Private Media Search on Public Databases

Giulia Fanti

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-230
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-230.html

December 10, 2012

Private Media Search on Public Databases

by

Giulia Cecilia Fanti

B.S. (Olin College of Engineering) 2010

A thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Engineering - Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Kannan Ramchandran, Chair
Gerald Friedland

Fall 2012

Private Media Search on Public Databases

by

Giulia Cecilia Fanti

Abstract

Private Media Search on Public Databases

by

Giulia Cecilia Fanti

Master of Science in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley

Kannan Ramchandran, Chair

Automated media classification techniques like speech processing and face recognition are becoming increasingly commonplace and sophisticated. While such tools can add great value to the public sphere, media searches often process sensitive information, leading to a potential breach of client privacy. Thus, there is great potential for applications involving privacy-preserving searches on public databases like Google Images, Flickr, or "Wanted Persons" directories put forth by various police agencies. The objective of this thesis is to argue that private media searches masking the client's query from the server are both important and practically feasible. The main contributions include an audio search tool that uses private queries to identify a noisy sound clip from a database without giving the database information about the query. The proposed scheme is shown to have computation and communication costs that are sublinear in database size. An important message of this work is that good private search schemes will typically require special algorithms that are designed for the private domain. To that end, some techniques used in the private audio search tool are generalized to adapt nearest-neighbor searches to the private domain. The resulting private nearest-neighbor algorithm is demonstrated in the context of a privacy-preserving face recognition tool.

Contents

Contents i

List of Figures iii

Acknowledgements vi

1 Introduction 1

1.1 Outline 5

2 Related Work 6

2.1 Cryptographic advances 6

2.2 Media recognition algorithms 8

2.3 Private Media-matching 10

2.3.1 Two-way private media matching 11

2.3.2 One-way private applications 12

3 Private Queries 15

3.0.3 Multi-Server Private Information Retrieval 15

3.0.4 Private Stream Search 17

4 Privacy-Preserving Audio Search 22

4.1 Audio search algorithm 22

4.2 Private audio search 24

4.2.1 PIR Version 25

4.2.2 PSS Version 28

5 Audio Search Tool Results 31

5.1 Audio Search Accuracy . . . 32
5.1.1 PSS Recognition Rates . . . 36
5.2 Security . . . 37
5.3 Resource Consumption . . . 39
5.3.1 Feature Distribution . . . 40
5.3.2 Communication Costs . . . 41
5.3.3 Computation Costs . . . 43

6 Generalized Privacy-Preserving Media Matching 52
6.1 Distance-preserving hashes via random projections . . . 54
6.2 Case Study: Face Recognition . . . 55
6.2.1 Fisherfaces . . . 55
6.2.2 Privacy-preserving adaptation . . . 57
6.3 Results . . . 57
6.3.1 Accuracy of Hashing Scheme . . . 58
6.3.2 runtime . . . 60

7 Conclusion 64
7.1 Computationally Efficient Two-Way Privacy . . . 64
7.2 Making PIR Practically Viable: A P2P Approach . . . 65
7.3 Remaining challenges . . . 67

Bibliography 69
References . . . 69

A Proofs for Claims 72
A.1 Proof of Observation 1 . . . 72
A.2 Proof of Observation 2 . . . 73

B Additional Algorithm Details 74

List of Figures

1.1 High-level diagram for a one-way-private face-recognition system. The client wishes to classify a media file in such a way that the server cannot learn information about the client's query. Red solid arrows signify data provided by the client, blue dotted arrows by the server, and purple dashed by a combination thereof. 2

2.1 Training face data (left) and corresponding eigenfaces (right). Images courtesy of Santiago Serrano with Drexel University, `http://www.pages.drexel.edu/\protect\unhbox\voidb@x\penalty\@M\{}sis26/Eigenface%20Tutorial.htm` 9

2.2 Media search techniques categorized in terms of privacy level, computational efficiency, and communication requirements. 10

3.1 Basic PIR scheme. Each of two servers computes the bitwise sum of a user-specified subset of database files. $\langle \cdot, \cdot \rangle$ denotes the inner product of two vectors. Because the two user-sepcified subsets differ only at the ith index, the binary addition of the results from both servers gives the original desired file. 16

3.2 Example of PSS search in which the user wants the ith file from a database. The server, which has only the public encryption key, computes n modular exponentiations and returns the product of all of these. The client then uses the private key to decrypt the results and obtain the desired file. 19

3.3 Example of PSS search in which the user wants all files containing the word "red". When the server returns the buffer, the user can decrypt bins 2 and 4 to determine the desired file. 20

4.1 Audio feature generation. The spectrogram of an audio clip is used to calculate the sum energy in frequency bands, and the band energy values are compared to one another. This same process is done on successive, overlapping time segments of audio to get a feature representation of an entire audio file. 23

4.2 Diagram of public audio search algorithm. The client successively sends each subfingerprint to the server, which returns all fingerprint blocks that match the query exactly at the specified position. The client checks the bit error rate (BER) between the query and the returned fingerprint block and declares a match if the BER is below a threshold. 25

4.3 Dual database structure consisting of a lookup table and a song directory. . 28

4.4 Private audio search using PSS. The client submits a new PSS query for each q_w in the query fingerprint block. This figure shows the client's PSS query for q_0 =0x0003. The server cycles through all the fingerprint blocks in the database and checks the first subfingerprint in the block. Two successive fingerprint blocks from the same song are just shifted by one subfingerprint, so they have 255 subfingerprints in common. 30

5.1 Precision-recall curves for subfingerprints with an overlap factor of $\frac{31}{32}$, marked with blue stars, $\frac{1}{2}$ shown with red circles, and $\frac{1}{16}$ shown with green squares, at $SNR = 15$ (solid lines), and $SNR = 5$ (dotted line). We observe that the recognition rates appear to be much more sensitive to SNR than overlap factor. 33

5.2 Impact of AWGN noise on error rate between the noisy query and the original song. As expected, the bit error rate curve is essentially the same for 16- and 32-bit fingerprints. However, 16-bit fingerprints have a higher number of exact subfingerprint matches. 34

5.3 Recognition rate as a function of SNR. The experimental results, plotted in red, illustrate the recognition rates from 95 trials, with 16-bit subfingerprints and $\epsilon = 0.45$. The adjusted experimental results, plotted in green, show the recognition rates if we strip off the 'false matches', i.e. the instances of an incorrect fingerprint block being matched to the query, but giving the correct result nonetheless because the incorrect block comes from the correct song. 36

5.4 Histogram showing how often each print occurs in the database of 96 songs. For this histogram, each print is chosen to overlap by a factor of $\frac{1}{2}$. Ideally, we would like this distribution to be uniform. 40

5.5 Worst-case resource consumption in AudioSearchPIR:FullSearch as a function of the proportion of subfingerprint bits kept private. 46

5.6 Mean runtime as a function of proportion of bits kept private. The times are normalized by the mean time for a non-private search. We see that PIR drastically outperforms PSS as the percentage of private bits increases. More importantly, for relatively low proportions of private bits (below about 0.7), the PIR scheme's runtime is within an order of magnitude of the non-private query. In these trials, $SNR = 15$, and averages are taken over 10 trials. . . 47

5.7 Toy example of server decoding partially revealed bits with infinite SNR, i.e. $p_e = 0$. The received codeword has four 'erasures', so the server can narrow down the possible matches to $2^4 = 16$ equally likely codewords in the decoding space, assuming codewords are drawn uniformly. 48

6.1 Method for converting arbitrary feature vectors into "subfingerprints". This facilitates a private search similar to the one used in the previously-described audio search. h_i denotes a randomly-drawn ℓ-dimensional hyperplane, where ℓ is the the original feature vector length. 55

6.2 Mean recognition rate as a function of total number of hash bits, parameterized by subfingerprint length k. 59

6.3 Graphical selection of subfingerprint size for a fixed number of hash bits. The selection technique shows how to choose k in order to upper bound by ϵ the probability of not getting any exact matches in the subfingerprint search. . 61

6.4 Mean runtime as a function of subfingerprint size k, parameterized by the total number of hash bits $n_s k$. The circled data points correspond to the circled points in Figure 6.2, to give an idea of the computational costs associated with a particular level of accuracy. 63

7.1 Peer-to-peer service providing search anonymity. 66

Acknowledgements

I am extremely grateful to a number of individuals, without whose support I could not have completed this work. To my adviser, Kannan Ramchandran, thank you so much for your patience and for your guidance; your help was invaluable when I found myself stuck in dead ends. To Matthieu Finiasz, cryptologist at CryptoExperts in Paris, France, my deepest thanks for your willingness to always discuss this project with me, at resolutions both high and low. Your patience and input were invaluable. To Gerald Friedland, thank you for helping me to think about the big picture more clearly and critically in this project, and also for your advice regarding the thesis itself. I am also very grateful for input and suggestions from Venkatesan Ekambaran and Jaeyoung Choi. Many thanks to the National Science Foundation for their support in the form of Graduate Research Fellowship DGE-1106400.

Thank you also to my parents, Monica and Paolo, and my sister, Victoria, for their constant and unwavering support. Thank you to Kevin for helping me through the rough spots.

Chapter 1

Introduction

Content-based media searches are growing to be a key component of many emerging applications, and they have the potential to change the way people interact with their environments (see e.g. [1], [2]). Examples of these technologies include voice and face recognition, location estimation, mood classification from videos or images, and internet searches that find images similar to an uploaded query (query by example). Recognition and classification are critical components of the decision-making process; as automated media recognition grows increasingly sophisticated, so too grows the ability of machines to make intelligent choices. However, practical automated recognition systems typically compare queries to existing databases that are stored remotely by necessity; small devices like cell phones cannot possibly store databases as large and comprehensive as those of major web services like Google or Facebook.

However, if a user wishes to classify a media file containing user-specific details, the server can often deduce information about the client, giving rise to significant privacy concerns [3]. For instance, in a photo-tagging service, the server could fairly easily use the tagged names of photograph subjects to map facial images to personal information like social security numbers [4]. This particular scenario may not be an imminent threat, but the fact that it is possible could give rise to security threats like fraud and identity theft, not to mention less malicious forms of manipulation in the form of advertising, for instance [5], [6].

Given the relative frequency with which websites are hacked, the public vulnerability stems not only from server-side attacks, but also from external attackers who somehow access the server's information [7]. Moreover, there are situations in which a server should not have any information about the client, such as when the client does not trust the server, or when the server does not wish to be liable for access to clients' private data.

The appropriate privacy settings in a system depend entirely on the problem definition. Broadly speaking, private media searches typically fall in one of two main categories: either the server *and* the client wish to keep their data secret, or only the client wishes to protect his/her data. A significant amount of work has been done on the former category. Specifically, advances have been made in a variety of biometric authentication/matching scenarios like facial image and fingerprint recognition on private databases, e.g. [8]–[11]. However, comparatively little work has been done on the latter scenario, which corresponds to one-way private queries on public databases. This may be partially due to the fact that large-scale public media databases like YouTube and Google Images have only recently become commonplace. This topic will be discussed in greater depth in Section 2.3.2.

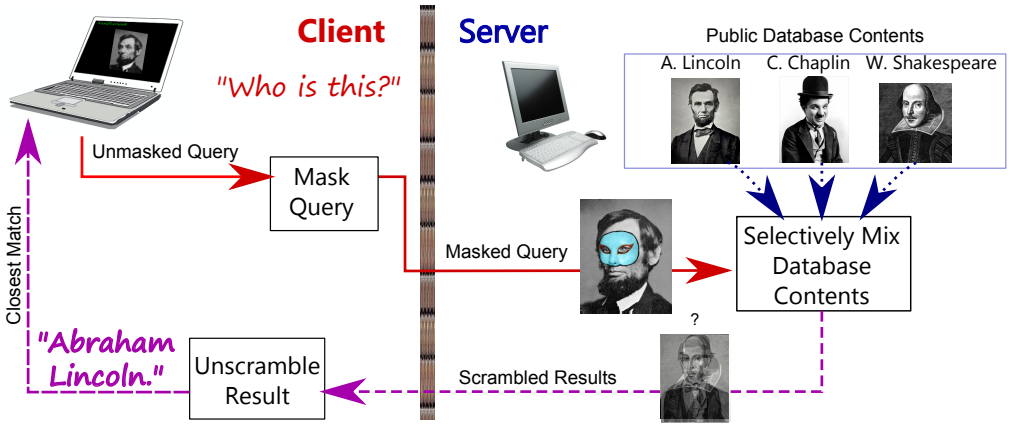


Figure 1.1: High-level diagram for a one-way-private face-recognition system. The client wishes to classify a media file in such a way that the server cannot learn information about the client's query. Red solid arrows signify data provided by the client, blue dotted arrows by the server, and purple dashed by a combination thereof.

To clarify the high-level picture, Figure 1.1 displays a sample one-way-private face-

recognition system, in which a client wishes to identify an image of Abraham Lincoln. From a design perspective, there are three major black boxes that must be addressed: masking the query, processing the query at the server, and deciphering the server's results. We will explore different ways to realize these functions later.

Despite the relative lack of attention from the research community to one-way privacy, content-based media similarity searches on public databases could enable many commercial applications, such as automatic subject- or location-tagging on private photo collections, or identification of suspected criminals in airports [12], [13]. The amount of publicly available media data is tremendous, ranging from Flickr and Facebook to video collections like YouTube and Vimeo to databases of wanted or missing persons published by various police agencies. In the commercial sphere, one could envision services in the healthcare domain that automatically classify medical data like DNA sequences or ECG signals. Patient monitoring devices could automatically and privately compare accumulated data against public databases of classification parameters; such tools could be useful in areas where medical access is sparse.

In a different vein, privacy-preserving recommendation systems could improve users' shopping experiences without revealing their preferences to companies that are likely to sell that information. For instance, companies like Redpepper [14] are starting to offer services that run face recognition software on every client that enters a network of subscribing stores—if a client is registered for Redpepper's service, he or she will receive personalized offers upon entering the store. We can think of this as a scenario warranting one-way privacy; if a store subscribes to the service, it has access to the individuals in Redpepper's database, making the database effectively public (from the store's point of view). However, Redpepper should not be able to track customers that are *not* enrolled in the database. These examples indicate that as collective public privacy concerns grow, it is important to be able to search public media databases in a way that does not compromise clients' private data; however, this can happen only if the associated performance losses (i.e. communication, computation) are minimal.

This technology has not yet been developed (in part) because traditional media recog-

nition techniques are difficult to adapt to the private domain. Cryptography tools are typically exact and intolerant of any loss in representation, while media objects are inherently distortion-tolerant, giving media classification algorithms considerable freedom to exploit this tolerance for performance gains. Offhand, the notion of media content recognition in the encrypted domain seems difficult, to say the least. Currently, the main obstacle to the adoption of private content matching is the severe inefficiency of known algorithms. The area has nonetheless received increasing attention as researchers continue to overcome technical limitations by exploring new mathematical tools and cleverly applying old ones. This trend will only strengthen as privacy concerns grow.

The goal of this work, at a broad level, is to examine more closely the potential of private queries as a tool for user-private content media searches. Specifically, we will present a private audio search tool that operates on a public database but keeps the user's query private. This system is meant largely as a proof of concept, since the application is somewhat contrived. The broader impact is that we will subsequently demonstrate how to generalize the concepts used in our audio search tool to think about the requirements of arbitrary private media recognition tools. Specifically, we will focus on privacy-preserving nearest-neighbor searches, and implement such a search in a face-recognition system based on the Fisherfaces algorithm.

The key observations of this research are threefold. First, private queries can be useful as a cryptographic primitive even when the client does not know offhand which file is desired. Second, even in a private search domain, private media searches can achieve adequate recognition rates with equal or only slightly degraded performance compared to public searches. Third and most important, private media-matching problems are often well suited to a signal processing perspective. That is, it is often suboptimal to use existing signal processing algorithms in the private domain by fitting cryptographic tools to non-private operations like computing the distance between two vectors; instead, it may make sense to actively modify existing algorithms or create new ones to be more suitable for privacy-preserving applications. We will explore this design philosophy in some detail and show how it can significantly reduce the costs of privacy-preserving media searches.

1.1 Outline

It is important to make a clear case why one-way private searches should be distinguished from two-way private ones. Chapter 2 opens with a discussion on this topic, including some speculation on the reasons for the engineering community's relative silence on this topic until very recently, and some historical perspective on relevant prior work. We will then discuss some important one-way privacy primitives like private information retrieval and private stream search in Chapter 3. Chapter 4 subsequently presents a proof-of-concept audio search tool based on private queries. The system takes a noisy sound clip of a song and identifies the song title by matching the clip to an external database of songs. All this is done without giving the server any information about the client's query. While the system is primarily intended as a building block for other categories of private media recognition (e.g. image, video), audio matching itself can be useful in certain applications, like location identification in images or videos [13]. Chapter 6 will then explain one method of generalizing the techniques used in the audio search system to other forms of media. As a demonstration of this generalization, we will present a sample private facial recognition system.

Chapter 2

Related Work

Privacy-preserving media-matching represents the intersection of many related research areas. On the cryptographic side, our system relies upon important tools in the field of private queries. On the signal processing side, these systems can be built only using content-recognition algorithms that are suitable for the private domain. We will describe some known media recognition algorithms that are commonly modified for privacy-preserving applications. Combining these signal processing and cryptography tools, most existing work on private media recognition is restricted to the symmetrically private domain. That is, the server learns nothing about the client's query, and the client learns nothing about the contents of the database except for the query result. This two-way variety of private media search is slightly different from our problem, but we will nonetheless provide a brief overview for completeness. Finally, in the area of one-way privacy, there are some practical applications in the literature, but they are largely unrelated to the media recognition problem.

2.1 Cryptographic advances

The relevant cryptographic tools for one-way privacy have been in development for decades. Search tools that protect the privacy of only the client are known as private

queries, and they span a number of capabilities. There are two types of privacy that can be achieved by these tools: information theoretic and computational. In the context of a private query, information theoretic security implies that even given infinite computational power, an adversary could never definitively learn the identity of the query. Meanwhile, computational security implies that an adversary cannot uniquely identify the query with bounded computational resources. Schemes offering both kinds of security will be considered.

The best-known variety of private query is called Private Information Retrieval (PIR); PIR allows a user to privately retrieve data at a known index in a database. This concept was first proposed in 1995 when Chor et al. presented an information-theoretically secure PIR algorithm with computation and communication costs linear in the database size [15]. The scheme, which will be explained in detail in Chapter 3, also requires at least two non-colluding servers to have identical copies of the database. Non-communicating servers is a weighty but plausible assumption, particularly given the emerging prominence of cloud data storage. The assumption is that if data is stored on cloud infrastructures managed by different services, those services are unlikely to cooperate with one another to correlate user behavior.

The work of Chor et al. in [15] spawned two main branches of research. One such branch attempted to find ways to efficiently execute PIR on a single server [16]. However, Beimel et al. subsequently showed that any single-server PIR scheme more efficient that simply transferring the whole database must rely on one-way functions, implying that only computational security can be achieved with a single server (barring full database transfer) [17]. Related to this, Sion et al. found that single-server PIR schemes are orders of magnitude less efficient than trivial database transfer on existing hardware [18]. Another branch of PIR research has focused more on reducing communication and computation costs by using multiple servers combined with database preprocessing [19], [20]; these schemes additionally achieve information theoretic security. Currently, there exist multi-server schemes that are sublinear in both computation and communication, but adding more elements to the database is quite inefficient [20]. In a related vein, Olumofin's and Goldberg's 2012 work

demonstrated that existing PIR schemes can be 2-3 orders of magnitude more efficient than simple database transfer in practice [21].

Another important branch of private queries is called Private Stream Search (PSS). PSS allows a client to search a database for all documents that contain at least one instance of a user-specified list of keywords. A scheme was first proposed by Ostrovsky and Skeith, and it also provides only computational security [22]. PSS relies on properties of homomorphic cryptosystems, which will be described later. In general, computationally secure systems tend to be impractical from a computational standpoint, and sometimes communication as well. However, recent work has significantly improved the efficiency of the original PSS scheme [23]; in fact, it was recently shown that the downlink communication cost of PSS can be made asymptotically equal to the downlink cost of the corresponding public search [24]; we will use such a construction in our audio search implementation in Chapter 4. We observe that PSS can be used as a computationally secure PIR scheme, as we will show in greater detail in Chapter 3.

2.2 Media recognition algorithms

As mentioned earlier, privacy tools tend to be intolerant of approximate matching and also quite inefficient. Because of this, sophisticated media recognition algorithms involving techniques like convex optimization are often poor candidates for privacy-preserving applications. Instead, most existing private media-matching research relies on well-known nearest neighbor methods for the signal processing aspect of the algorithm [9], [10], [25]. Nearest neighbor methods compute a feature vector for a query and then find the closest feature vector in the database by some metric—usually Euclidean distance. The nearest neighbor in the database is subsequently chosen as the most likely match.

The most prevalent example of such an algorithm in the media-matching sphere is Eigenfaces [26]. This algorithm represents images (both from the database and query) as linear combinations of basis vectors extracted from the facial training data. These basis vectors are found via singular value decomposition on a matrix representation of all the

Figure 2.1: Training face data (left) and corresponding eigenfaces (right). Images courtesy of Santiago Serrano with Drexel University, `http://www.pages.drexel.edu/~sis26/Eigenface%20Tutorial.htm`

images in the training set. The name 'eigenfaces' stems from the fact that the basis vectors, when rearranged into the rectangular dimensions of the training images, look like distorted facial features. See Figure 2.1 for some examples of database faces and the corresponding eigenfaces. By projecting each face onto this basis, the facial images are mapped to a lower-dimensional vector than the original image vector, which contained all the pixel intensities. That new vector acts as a feature vector, enabling a nearest-neighbor search. This nearest-neighbor search tends to be the most computationally and communication intensive portion of private adaptations [9], [10]. Nonetheless, Eigenfaces is one of the most common face-recognition algorithms used in the private domain, and we will use an extension of it in Chapter 6 to build a one-way private face recognition system.

While nearest-neighbor techniques can be difficult to adapt to the private domain, there do exist signal processing algorithms that inherently rely on finding exact matches, thereby rendering them very conducive to privacy-protection. We utilize such an algorithm for our audio search; the scheme was originally proposed by Haitsma and Kalker [27]. We will describe this scheme in much greater detail in Chapter 4, but the general idea is the following: Instead of searching for feature vectors that are close to the query vector in distance, find feature vectors that match the query vector *exactly* in at least one location (i.e. substrings of the corresponding bitstreams are identical), then compute the distance

of the resulting vectors. One can either select the closest such vector as a match, or choose the first vector whose distance from the query is below a certain threshold.

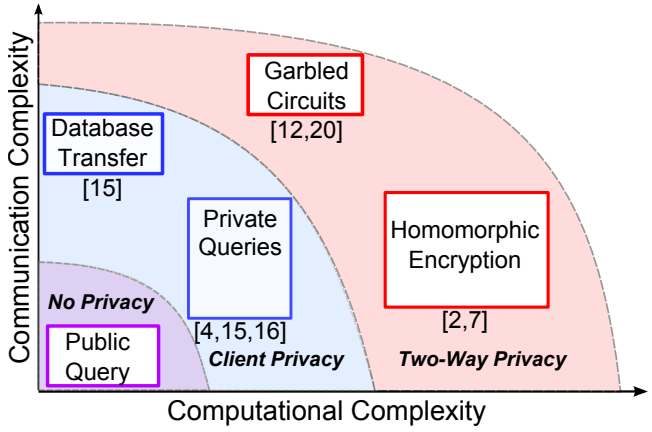


Figure 2.2: Media search techniques categorized in terms of privacy level, computational efficiency, and communication requirements.

2.3 Private Media-matching

At the intersection of these signal processing algorithms and cryptography techniques, we have private media searches. Private media searches ask for more from a search algorithm than public searches do. As such, they also cost more in terms of resources like communication and computation. Although the notion of private computing is garnering significant public interest, users are ultimately unlikely to tolerate the associated drawbacks if those drawbacks manifest themselves as significant delay and/or increased communication charges. Thus the primary goal of much private media matching research is to reduce the total resource demands of private search systems [10], [28].

One major obstacle to reducing resource costs is the fact that there is often a trade-off between computation and communication costs in private media search schemes. For instance, consider algorithms that rely on homomorphic encryption [9], [29]. In these algorithms, the client sends an encrypted query to a server, and due to mathematical properties of the cryptosystem, the server is able to do computations on the query despite not knowing the content of the query. Such algorithms are typically quite computionally heavy but the

communication costs can be fairly low. On the other hand, there are schemes that rely on garbled circuits, in which the server effectively sends a garbled version of the database to the client, and the client can extract a single desired element [10], [25]. These schemes require far less computation, but the communication costs are quite high for most practical circuit sizes.

To illustrate this point, some existing private media search techniques are displayed in Figure 2.2, depicted as a function of computation and communication costs. In practice, existing algorithms often combine several privacy primitives, so the schematic is only meant as a qualitative guide to the distribution and magnitude of costs for each approach. As mentioned earlier, most of the existing schemes fall into the outer band of this figure, offering two-way privacy protection. Since this variety of private media matching has received far more attention than the one-way variety used in our work, we will briefly present some primary contributions in the two-way area.

2.3.1 Two-way private media matching

Practical implementations of private media searches are usually presented in the context of two-way private biometric recognition; this can be for the purpose of user authentication or surveillance, among other possibilities. To the best of our knowledge, the first work specifically addressing privacy-preserving media recognition was conducted by Avidan and Butman in 2006 [30], in which they built a private face-matching system. This paper spawned a significant amount of related research using different techniques to do private media matching. The bulk of these works are related to private facial recognition [9], [10], [25], [31], [32], but there are also branches that are more focused on fingerprint image identification [29], speech processing [11], [33], and ECG signal classification [34]. These works (as well as others in the field) collectively managed to significantly reduce the computational and communication costs of running a two-way private media recognition system. For instance, subsequent papers like [25] are able to condense certain portions of the face recognition algorithm by as much as an order of magnitude in computation time and bandwidth compared to other private schemes. Other schemes, like the SCiFI system, build

an entire facial recognition search algorithm with built-in compatibility for privacy primitives [32]; we will discuss the need for this kind of design philosophy in Chapter 6. In any case, we will see later that even though these systems can achieve fairly low runtimes, their computational and communication loads are still linear in database size, which limits how efficient such systems can be on a large scale.

2.3.2 One-way private applications

We start by pointing out that two-way private searches are a generalization of one-way private searches. As such, symmetric private algorithms will always work when the database is public. However, the comparatively high cost of two-way private algorithms is our motivation for treating one-way private search algorithms as a separate entity. To our knowledge, there are no theoretical guarantees on the relative efficiencies of asymmetric and symmetric private search schemes, but in practice there are gains to be had by exploiting unilateral privacy. Applications of one-way privacy primitives are most common in the area of location-based services [35]. Location-based services are applications that use a client's physical location to provide services like displaying an appropriate map, or giving directions to the nearest bank. Such applications are particularly well suited to the user-private domain because the client's location is already known, meaning that the client is able to precisely describe what data it desires from the database. This fact melds nicely with the PIR setup described earlier, in which the client must know exactly which index it wishes to retrieve from the database [15]. Even in situations where the client requests all files in a database containing a list of keywords [22], the client must somehow know exactly which keyword to search for, which is typically a reasonable request in location-based services.

On the other hand, content-based media identification is a fundamentally different problem because the client has no prior information telling where in the database to search for the query. Along with some other factors, this is presumably why the topic has received so little attention in the signal processing community. In the past, a few authors of two-way private systems have briefly addressed the case when databases are partially or fully public; for instance, Erkin et al. discuss some small variations to their face recognition system if

the Eigenvectors for a database are public [9]. Similarly, Barni et al. briefly address this issue in their privacy-preserving ECG-classification system, giving small improvements if the server is willing to leak some information [34]. However, Shashank et al. built a fully one-way private image similarity search tool that requires a tree-structured database with files clustered by image similarity [28]. This work appears to be a fairly direct extension of [36], but to our knowledge, it is the only published work to date dealing exclusively and pointedly with one-way private media similarity searches. Indeed, their system manages to obtain satisfactory results at a much lower cost than comparable two-way private algorithms. In our research we present an alternate method of framing the one-way private media matching problems.

Temporal relevance

Despite the relative lack of attention to this area, we believe the time is ripe to pursue this research area. Cryptographically, the required tools have only recently matured enough to be considered viable. For a long time it was thought that practically-speaking, one-way privacy primitives are as inefficient as sending the whole database to the client. The issue was largely ignored until 2011, when Olumofin and Goldberg demonstrated that private queries can be as much as three orders of magnitude more efficient than database transmission in practice [37]. Many of the currently available multiserver PIR schemes have sublinear communication and computation costs, and the field is still active [19], [20]. There may very well come new innovations rendering private queries efficient enough to be practical.

On the other hand, public media databases have become commonplace only within the last decade (e.g. YouTube, Flickr, Facebook, Google Images), and these services did not use similarity searches until even more recently. The tremendous growth of public databases has proved increasingly useful for media matching applications and also heightened the potential privacy threat to the average user [13]. While the constant monitoring of people's online identities is perhaps not noticeable to the average user, it is absolutely constant, and varying in the degree of invasiveness. At least in the United States, the federal government and

private companies are actively monitoring mobile users, particularly with respect to location [38]. Just within the last two years, both Google and Facebook have had to settle lawsuits with the FTC for violating the privacy of users [39], and several companies (including Facebook) are moving steadily toward automated, ubiquitous face recognition [14]. In an orthogonal direction, a major search engine (Yahoo) has been the victim of a security breach, exposing the login credentials of many users—and this incident is by no means a unique case [7]. In short, online behavior gives an intimate understanding of people's daily lives. Whether this information is used with malicious intent or not, the issue of user privacy is a very relevant one, both from the client's and the server's perspective. The combination of these factors with the cryptographic advances mentioned earlier strongly suggest that user-private media searches can and should receive increased attention in coming years.

Chapter 3

Private Queries

We will start by explaining the fundamentals of one-way-private cryptographic tools, collectively termed 'private queries.' Private queries are searches on a database in which both the query and the result are masked from the server. The caveats are twofold: the client must know exactly what data is desired from the database, and the client may learn information about the database besides just the desired file during the search process. The latter condition is unimportant on public databases, but the fact that the user must know precisely what to request is problematic since media searches are inherently inexact. We will discuss ways to get around this stringent requirement in Section 4 and 6. The current section describes two representative private query methods used to privately retrieve items from a database.

3.0.3 Multi-Server Private Information Retrieval

Private Information Retrieval (PIR) allows a client to retrieve data at a particular index in a database without revealing the query (or the results) to the server. As mentioned above, there exist PIR implementations that require only one server [40], though in practice they are significantly less efficient than trivial database transfer [18]. We will emphasize multi-server schemes, which can achieve communication and computation that is sublinear in database size [19], [20]. Multi-server PIR schemes require the existence of at least two *non-*

colluding servers, each with a duplicate copy of the database. This assumption is strong, but not unreasonable; for instance, one could store data on clouds run by competing services, e.g. Amazon and Google.

We describe the basic PIR scheme from [15]. Both servers have copies of a database comprised of a binary string $x \in \{0,1\}^n$, and the user wishes to retrieve the ith bit, x_i. The user's request can be represented by $e_i \in \{0,1\}^n$, the indicator vector with a 1 at index i and 0's elsewhere. To disguise this query, the user generates a random string $a \in \{0,1\}^n$ with each entry a Bernoulli(1/2) random variable. The queries sent to servers 1 and 2 are $a \oplus e_i$ and a, respectively. Each server computes the inner product of its received query vector with the database x using bitwise addition (XOR) and returns a single-bit result. The user XORs the results from the two servers to get precisely x_i. The scheme is illustrated in Figure 3.1; if the database consists of indexed files rather than bits, the same process is carried out on each bit plane.

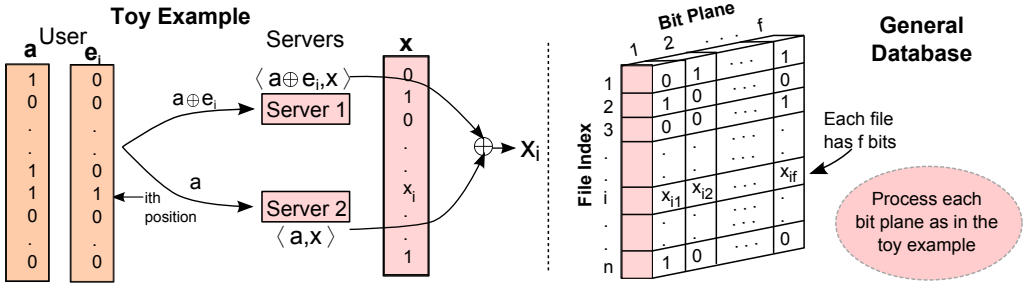


Figure 3.1: Basic PIR scheme. Each of two servers computes the bitwise sum of a user-specified subset of database files. $\langle \cdot, \cdot \rangle$ denotes the inner product of two vectors. Because the two user-sepcified subsets differ only at the ith index, the binary addition of the results from both servers gives the original desired file.

This multi-server PIR scheme is information theoretically secure, meaning that an adversary cannot break the scheme even with unlimited computing power. This is proved in Section 5.2. Communication-wise, it is also relatively light, even in the general case where the database is a list of files rather than a list of bits. The total communication cost in that case is only $2(n + f)$ bits. n bits are sent to each server, and two sets of f bits are received, where f is the file size (in the example above, $f = 1$ since the database is just

a binary string). When more servers are added, this technique has a general extension, which reduces the total number of communicated bits. For a general number of databases ζ, the total communication cost can be reduced to $2^d(1 + d\sqrt[d]{n})$, where $d \triangleq \log_2 \zeta$ [15]. The communication reduction comes from embedding the databases in $d-$dimensional cubes and sending randomized subsets to the various databases to be XOR'ed, much like the two-server example. From a practical standpoint, it is unclear where large public databases are likely to be duplicated on many non-colluding servers. Perhaps in a distributed storage situation, servers may be less likely to collude, but for databases as large as Google Images, this seems like a difficult condition to satisfy. In this paper, we will focus on the two-server case, though we consider a possible peer-to-peer PIR system relying on many proxy servers in Section 7.2.

3.0.4 Private Stream Search

We previously noted that PIR can be done with a single server at the expense of computation and communication; moreover, the resulting level of security is computational rather than information-theoretic, meaning that the scheme cannot be broken with known practical methods. We will start by presenting a single-server computationally-secure PIR scheme based on [22] for completeness, since it is similar in spirit to the privacy tools used in many two-way private schemes.

Private Stream Search (PSS) was originally designed to perform private keyword searches on databases [22]. For instance, one might want to privately retrieve all files in a text document database that contain the word "red". The problem statement is a bit different from that of a PIR query, since the client is no longer searching for a specific file by index. However, this setup can easily be framed as PIR by making the dictionary of keywords equal the list of document indices. We will explain the scheme from [22] in the context of PIR for ease of comprehension, though the original framing of PSS is also useful and represents a different variety of private query. Before detailing the algorithm, we introduce an important preliminary concept known as homomorphic encryption.

Homomorphic Encryption

Additively homomorphic cryptosystems are utilized to varying degrees in almost every current private media content retrieval scheme. In the general case, a public-key cryptosystem comprised of encryption function $\mathcal{E}(\cdot)$ and decryption function $\mathcal{D}(\cdot)$ is *homomorphic* if there exist operations $f_1(\cdot)$ and $f_2(\cdot)$ such that $f_1(x, y) = \mathcal{D}(f_2(\mathcal{E}(x), \mathcal{E}(y)))$ [41]. There are several cryptosystems with this property, but the additively homomorphic Paillier cryptosystem is used in most existing private media content retrieval applications [42]; it is homomorphic with f_1 corresponding to addition and f_2 to multiplication, i.e.

$$\mathcal{E}(x + y) = \mathcal{E}(x)\mathcal{E}(y) \tag{3.1}$$

This implies that multiplication by a constant c takes a particularly simple form:

$$\mathcal{E}(cx) = \mathcal{E}(x)^c \tag{3.2}$$

One important point is that the Paillier cryptosystem is a randomized mapping from a set of numbers to a larger set of numbers. Therefore, successive encryptions of a single number can take on different values, even if the same public key is used for all the encryptions.

PSS as Single-Server PIR

Now we apply this to the single-server PIR problem. Consider an ordered list of possible file indices, from 1 to n, with i being the desired index. Using an additively homomorphic cryptosystem, the client generates an encrypted query vector q of length n, with $\mathcal{E}(1)$ at index i and $\mathcal{E}(0)$ at the remaining indices. Because the Paillier cryptosystem is randomized, all the entries in this encrypted vector will be different with high probability. The encrypted query vector is sent to the server, as shown in Figure 3.2. Now the server goes through its database; for every file index, the server checks the corresponding entry in the encrypted query vector. This entry, which is either $\mathcal{E}(0)$ or $\mathcal{E}(1)$, is raised to the power of the whole file represented as a number. So if $i = 1$, then $q_1 = \mathcal{E}(1)$, giving $\mathcal{E}(1)^{f_1}$; by equation 3.2, this quantity is equivalent to the encryption of f_1. The same procedure is also done for all the other files f_j, $j \neq i$, but in that case, $\mathcal{E}(0)^{f_j} = \mathcal{E}(0)$. After going through the whole

database, the results of these exponentiations are multiplied together and returned to the client. Because of the homomorphic cryptosystem, multiplying ciphertexts corresponds to adding their arguments, so we get $\mathcal{E}(0 + ... + 0 + f_i + 0 + ... + 0) = \mathcal{E}(f_i)$. Thus the client decrypts precisely the file at the desired index, but the server learns nothing about the client's desired index.

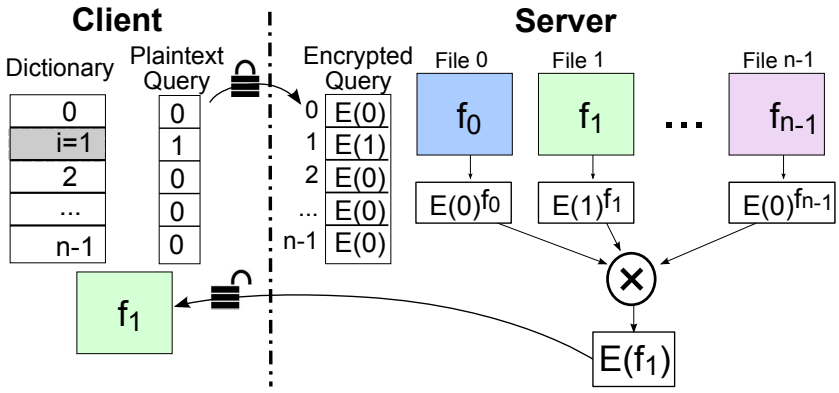


Figure 3.2: Example of PSS search in which the user wants the ith file from a database. The server, which has only the public encryption key, computes n modular exponentiations and returns the product of all of these. The client then uses the private key to decrypt the results and obtain the desired file.

PSS for Stream Search

Now let us consider the slightly more complicated original intent of PSS—keyword search—as it is presented in [22]. For this example, we wish to retrieve all files containing the word "red". Suppose there is an ordered list of possible search keywords. This list would include the word "red", among other keywords. The client generates an encrypted query vector, with $\mathcal{E}(1)$ at the indices of the desired keywords and $\mathcal{E}(0)$ at the remaining indices. Because the Paillier cryptosystem is randomized, all the entries in this encrypted vector will look different with high probability, even though each entry in the vector is either $\mathcal{E}(0)$ or $\mathcal{E}(1)$. This encrypted query vector is sent to the server along with the public key, as shown in Figure 3.3.

The server starts with a full database and a buffer, whose size is chosen according to

the expected number of matching documents per query; all buffer entries are initialized to the value 1. Upon receiving the encrypted query and public key from the client, the server goes through each file in its database. For every keyword in a given file, the server checks the entry in the encrypted query vector corresponding to that keyword, and multiplies the encrypted query entries for all the keywords in the file. By equation 3.1, the result is an encryption of the sum of the arguments. So if the word "red" is contained once in the document, we get $\mathcal{E}(0+0+...+1+...+0+0) = \mathcal{E}(1)$ (the zeros represent other keywords in the documents that were not "red"). If the file does not contain the desired keyword, the product of these query entries will be equivalent to $\mathcal{E}(0)$. This product is raised to the power of the whole file represented as a number. So if file 1 contains the word "red" once, we would get $\mathcal{E}(1)^{f_1}$, which is the encryption of f_1 by equation 3.2. This quantity is added (in plaintext) to a fixed number of randomly selected buffer bins (e.g. three bins per file). Here 'addition to the buffer' corresponds to modular multiplication in the encrypted domain, so the arguments get added. The same procedure is also done if the file does not contain the keyword, but $\mathcal{E}(0)^{f_1} = \mathcal{E}(0)$, so adding it to the buffer is essentially transparent. After going through the whole database, the buffer is returned to the client.

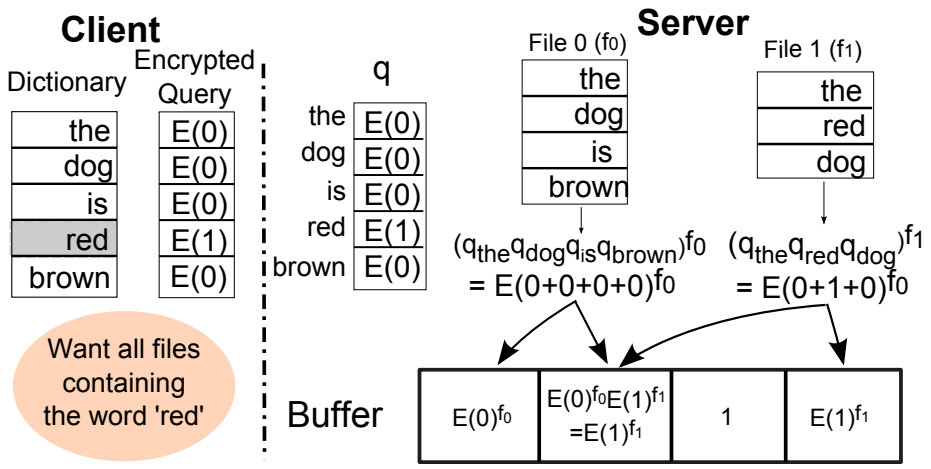


Figure 3.3: Example of PSS search in which the user wants all files containing the word "red". When the server returns the buffer, the user can decrypt bins 2 and 4 to determine the desired file.

Once the client receives the buffer, it must decrypt all the buffer bins. For simplicity,

we assume that only one keyword was sought in the query, e.g. "red". First of all, we note that files in which the query keyword(s) do not appear are effectively invisible to the client, as stated before. Second, suppose that two different files each contain the keyword once and (by chance) get added to the same buffer bin. Then when the client decrypts that buffer bin, the result will be the plaintext *sum* of those two files, so the client cannot recover the individual files. Therefore, the client must search for *singleton bins*, or bins that were assigned only one nonzero file by the server. When the client decrypts such a bin, it will obtain exactly that nonzero file.

In the PSS implementation of [22], only singletons can be decrypted, so the buffer size must be selected as $O(\log(n))$ in order to retrieve all results with high probability, where n is the number of documents. However, a scheme proposed by Finiasz and Ramchandran achieves optimal (i.e. identical to nonprivate) downlink communication by using techniques analogous to LDPC decoding [24]; each time a singleton file is decoded, all the other instances of that particular file in the buffer are removed by the client. This requires the client to know which bins contain a file, given the file content. One way to achieve this is by having the server seed its random number generator with the content of the file. In that way, the client can reproduce the "randomness" of the server, while preserving decodability in a probabilistic sense. We utilized this technique in our PSS implementation in Chapter 4.

Chapter 4

Privacy-Preserving Audio Search

As we briefly mentioned in Chapter 1, a practically viable private media search system will generally feature a customised search algorithm. In the early days of privacy-preserving signal processing research, the focus was mostly on pairing cryptography techniques with existing search algorithms [9], [10]. This approach can add unnecessary costs if the underlying search algorithm is not inherently privacy-accordant, but there do exist accurate search algorithms that *are* already compatible with private searches. We will start by presenting a private audio search tool based on precisely such an algorithm, i.e. the audio search of Haitsma and Kalker [27]. In this system, the client possesses a noisy sample of a song, such as what one might hear on the radio. The user wishes to identify the sample without giving any information about the noisy query to the server. The key idea is to integrate interdisciplinary techniques from both the cryptographic and signal processing communities. We begin by providing a brief overview of the non-private search algorithm.

4.1 Audio search algorithm

The underlying audio search algorithm we build upon was first proposed by Haitsma and Kalker [27]. This search tool relies on quantizing audio features and looking for exact matches between the query and the database. Their scheme represents audio files as collec-

tions of time-dependent, quantized audio features called subfingerprints. The quantization in these features allows different, noisy renditions of the same clip to map to the same subfingerprint, thereby adding robustness. Subfingerprints are found by first computing the spectrogram of a fixed-length audio segment. For example, for a 10-bit subfingerprint, the spectrum is divided into 11 frequency bands, and the total energy in each of these bands is integrated over the time interval of the audio segment. Each band is then compared to the frequency band below it; if the ith band has less total energy than the $(i+1)$th band, then the ith bit of the subfingerprint is assigned the value '0'; otherwise, it gets assigned the value '1'. This procedure is illustrated in Figure 4.1. An important point is that once the

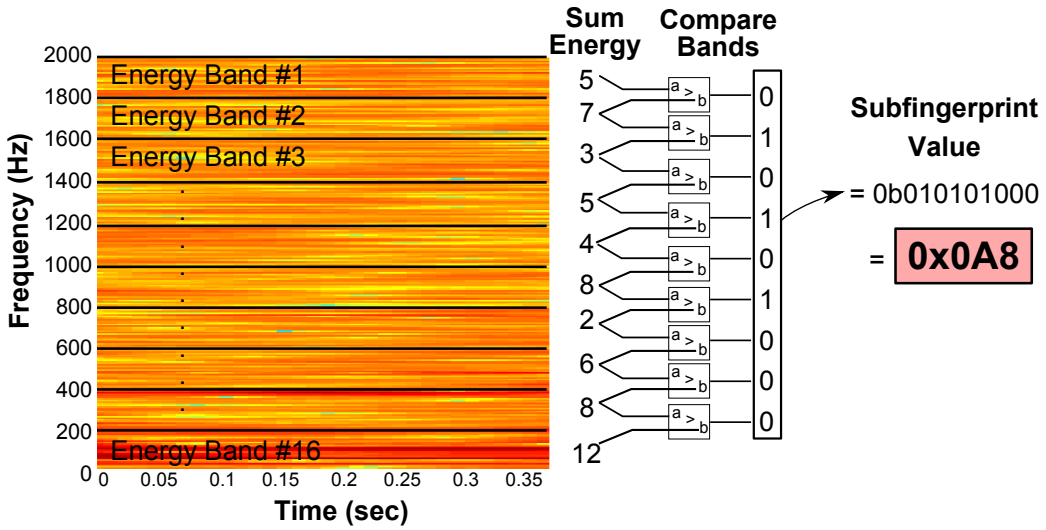


Figure 4.1: Audio feature generation. The spectrogram of an audio clip is used to calculate the sum energy in frequency bands, and the band energy values are compared to one another. This same process is done on successive, overlapping time segments of audio to get a feature representation of an entire audio file.

feature vectors are calculated, the rest of the algorithm is agnostic to the type of feature vector used. This speaks to the broader applicability of the private search scheme; our modular approach allows one to substitute images or video for audio, as long as there exists an appropriate feature vector extraction scheme for that media class. We will see later how to avail of this modularity by describing a scheme for nearest-neighbor matching, which we apply to the face-recognition problem.

Each song in the database is represented as a list of subfingerprints, which are found by executing the steps in Figure 4.1 on successive, overlapping, fixed-length time intervals. The longer a song is, the more subfingerprints are required to describe it. On the client end, a three-second query audio clip, potentially corrupted by noise or other distortions, is also converted to this subfingerprint representation—three seconds of audio map to 256 subfingerprints total, which are collectively termed a 'fingerprint block'.

The client sequentially sends each of the 256 noisy query subfingerprints to the server; for each one, it receives all fingerprint blocks in the database that contain the desired value at the desired position. For example, if the first query subfingerprint has the value 0x0003, then the algorithm will first find all fingerprint blocks in the database that start with the value 0x0003. If one of the returned blocks is very similar to the query block in Hamming distance, a match is declared. Otherwise, the algorithm submits the second query subfingerprint to the server, and so forth until it has tried all 256 query prints. This scheme is illustrated in Figure 4.2. Note that each time we search for the ith of the 256 query subfingerprints, the server returns all fingerprint blocks in the database that *exactly* match the query prints in the ith location; this property allows for modularity and sets the stage for an easy transition to the private domain.

4.2 Private audio search

In order to make the scheme private, we convert every query subfingerprint search to a private query, thereby hiding all information from the server. Two versions were implemented: one using two-server PIR, and one using single-server PSS. That being said, multi-server PIR is far more likely than single-server PSS to gain traction as a practical privacy tool. Computationally private schemes require the use of asymmetric encryption operations which, when used with secure key sizes, are very expensive. In practice, information-theoretically secure PIR can be at least 1000 times faster.

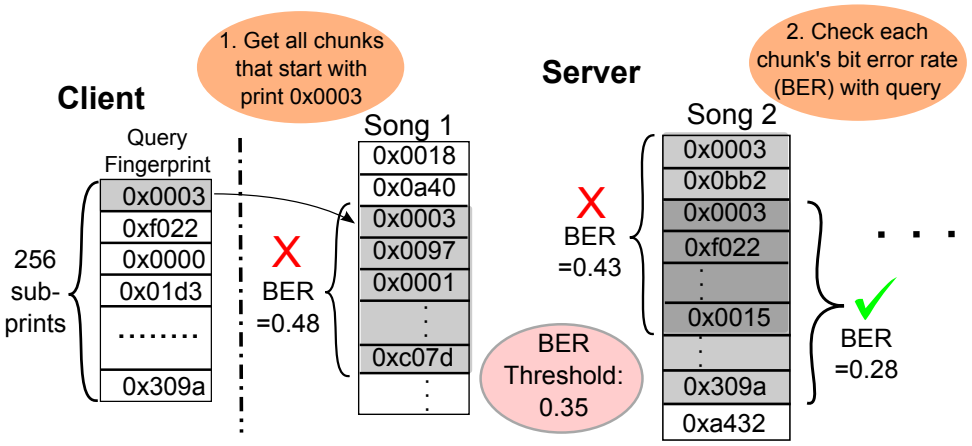


Figure 4.2: Diagram of public audio search algorithm. The client successively sends each subfingerprint to the server, which returns all fingerprint blocks that match the query exactly at the specified position. The client checks the bit error rate (BER) between the query and the returned fingerprint block and declares a match if the BER is below a threshold.

4.2.1 PIR Version

We will begin by describing a slightly simplified version of the PIR-reliant algorithm, dubbed 'AudioSearchPIR', for the sake of clarity. The search could use any PIR implementation, information-theoretically secure or computationally secure. The client begins by converting the noisy query to a list of 256 subfingerprints, just as before. Recall that in the non-private version of the algorithm, the client would send the first subfingerprint q_1 to the server to be matched (suppose again that the first subfingerprint takes the value q_1 ='0x0003'). Ideally, the client would know where in the database to search for instances of the subfingerprint 0x0003 and use PIR to retrieve all the corresponding chunks. However, the client does not know where in the database to find the desired subfingerprint.

There are many ways to address this, and search efficiency can be altered by using various database indexing schemes. One method to reduce the communication cost is to use an inverted database, which is indexed by subfingerprint. Essentially the database becomes a list in which the content at the q_wth index is the set of all fingerprint blocks that contain subfingerprint q_w. Now, if the client wishes to retrieve the fingerprint blocks in the database that start with subfingerprint 0x0003, it submits a PIR query to the server for the

data at index 0x0003. It receives all the fingerprint blocks containing 0x0003. These steps are outlined in Algorithm 4.1.

At this point, we consider two possible endings to the algorithm. The first possibility is dubbed 'AudioSearchPIR:FullSearch'; this approach compares the bit-error rate (BER) of each returned fingerprint block to the running minimum BER. So if a block has a lower BER than the running minimum, then that block becomes the new optimal solution. This approach implies that a PIR query is executed for each of the 256 query subfingerprints, giving the closest match possible. AudioSearchPIR:FullSearch is detailed in Algorithm 4.1.

Algorithm 4.1 AudioSearchPIR:FullSearch

function ExtractFeatures(NoisyAudio)
    Input: Noisy audio clip, at least 3 seconds long (NoisyAudio)
    Output: Vector of 256 query subfingerprints (q)
end function
function Pir($\mathcal{Q}, \mathcal{S}, \mathcal{P}$)
    Input: Server ID ($\mathcal{S}$), PIR protocol ($\mathcal{P}$), PIR query ($\mathcal{Q}$)
    Output: Mixed database contents (ScrambledResult)
end function
function ComputeBitErrorRate(q, r)
    Input: query fingerprint block (q), scrambled fingerprint block output of PIR (r)
    Output: Bit error rate between the two vectors (BitErrorRate)
end function

Input: Noisy audio clip (NoisyAudio)
Output: ID, content of closest match in Hamming distance (ClosestMatch)
MinBitErrorRate $\leftarrow 1$
$q \leftarrow$ ExtractFeatures(NoisyAudio)
for $w = 0 \rightarrow 255$ **do**
    Result $\leftarrow \mathbf{0}$
    for all non-colluding server $\mathcal{S}_v$ **do**
        $\mathcal{C}$ generate PIR query $\mathcal{Q}_v$ for subfingerprint q_w, to be sent to server $\mathcal{S}_v$
        ScrambledResult $\leftarrow$ Pir($\mathcal{Q}_v, \mathcal{S}_v, \mathcal{P}$)
        Result $\leftarrow$ Result $\oplus$ ScrambledResult
    end for
    BitErrorRate $\leftarrow$ ComputeBitErrorRate(q, Result)
    if BitErrorRate $<$ MinBitErrorRate **then**
        MinBitErrorRate $\leftarrow$ BitErrorRate
        ClosestMatch $\leftarrow$ Result
    end if
end for

Note that if the utilized PIR scheme $\mathcal{P}$ is information-theoretically secure, 'AudioSearch-

PIR:FullSearch' leaks no information whatsoever to any of the servers $\mathcal{S}_v$, provided the servers do not collude. We will prove this in Section 5.2. However, information theoretic security comes at the cost of inefficiency. For instance, we could avoid submitting all 256 PIR queries, and instead stop searching as soon as a fingerprint block is found with a BER below some threshold ϵ, as is done in the non-private algorithm. This approach is denoted 'AudioSearchPIR:ThresholdSearch', and is presented in full in Appendix B as Algorithm B.1. This modification impacts neither asymptotic nor worst-case communication and computation costs, but it does reduce average costs in practice. In Section 5.2, we will also show that this modification does not leak any 'significant' information to the sever. By 'significant', we mean information relevant to the semantic content of the audio file (as opposed to Gaussian noise). We will also discuss the security properties of this scheme in Section 5.2

Further refinement

The algorithms above have some inefficiencies built in. For instance, the downlink communication is higher than strictly necessary—even if the client only wants all fingerprint blocks that *start* with 0x0003, it will receive all blocks that *contain* 0x0003. This is done for a simple reason: if the first subfingerprint is not an exact match, then the client must search for all blocks that contain q_2 at the *second* subfingerprint, and so forth. Thus, having a separate database for all 256 subfingerprint positions is undesirable. One way to circumvent this issue is by using a dual database structure comprised of a lookup table and a song directory; this technique, dubbed 'AudioSearchPIR:LookupTable', is the approach we used in our implementation. We have included this Algorithm B.3 in Appendix B since it is similar in concept to the algorithms already listed.

In this new database structure of AudioSearchPIR:LookupTable, the song directory is just a list indexed by song number. Each song has an associated sequence of subfingerprints, as described earlier. If there are r songs in the database, this song directory will have r entries, and each entry will contain a different number of subfingerprints depending on the length of that particular song. The lookup table, on the other hand, is a list indexed by subfingerprint. So if each subfingerprint is 16 bits, the lookup table will have 2^{16} entries. The ith entry of the lookup table is a list of all the locations where subfingerprint i can be found in the database. A figure depicting this database setup is shown in Figure 4.3.

If the client is trying to find matches for the first subfingerprint 0x0003, it starts by

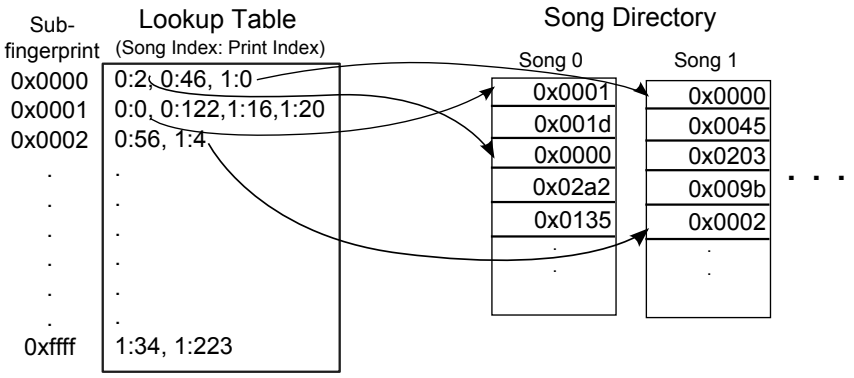


Figure 4.3: Dual database structure consisting of a lookup table and a song directory.

submitting a private query to the lookup table for index 0x0003. In return, the client will get a list of exact locations of 0x0003 in the database. Now if the client wants all fingerprint blocks that start with 0x0003, it will submit a series of new PIR queries, each of which retrieves a fingerprint block that starts precisely at one of the indices in the list of locations. This reduces the downlink communication cost, but it can also significantly increase the uplink cost, because the client has to submit a new private query for every instance of the desired subfingerprint in the database. Specifically, this becomes detrimental when the database is large and uplink communication in a given PIR query dominates over downlink.

4.2.2 PSS Version

As mentioned earlier, single-server PSS can also be used as a computationally secure PIR scheme by treating file indices as keywords. Using such a scheme, any of the algorithms listed above would work fine since they do not rely on a particular implementation of PIR.

However, one can also use PSS in the way it was originally designed, by designating keywords that reflect the content of an audio segment. We also chose to implement this latter version. Each subfingerprint can be thought of as a keyword, just as each subfingerprint block can be thought of as a document. Then we can do one of two things: One option is to submit a single PSS query for all documents that contain at least one of the subfingerprints from the query fingerprint block. For instance, if all 256 of the query subfingerprints happened to be either '0x0003', '0x0000', or '0x0001', then we would submit a single PSS query total with $\mathcal{E}(1)$ at the 0th, 1st, and 3rd indices, while the rest of the indices would contain $\mathcal{E}(0)$. PSS returns all documents that contain *at least one* of the desired keywords, so the client would receive all fingerprint blocks containing 0x0003, 0x0001, 0x0000, or any

combination thereof. This approach is appealing because there is only one PSS query for all 256 subprints, thereby significantly reducing uplink communication, which can be high for PSS. However, for reasons that will be explained shortly, it is difficult in this framework to implement partial privacy, in which a few bits from each subfingerprint are revealed to the server for efficiency gains.

Since partial privacy can give large efficiency gains overall, it is a feature we want to enable. To this end, we propose a different option dubbed 'AudioSearchPSS' —closer in spirit to the PIR implementations. This approach submits a separate PSS query for each of the 256 query subfingerprints and retrieves all fingerprint blocks that contain q_w at index w. The client has to send index w in plaintext to the server, but this doesn't leak any information. In our system, we chose to implement AudioSearchPSS, which is illustrated in Figure 4.4.

In this figure, the server receives 1) a PSS query Q generated for subfingerprint q_0 =0x0003, and 2) a plaintext value indicating that $w = 0$. Given this information, the server cycles through every fingerprint block in the database. For a given block, it retrieves the subfingerprint at $w = 0$ (suppose it is 0x0018, as in the figure), and finds the index of Q that corresponds to that subfingerprint (in our example, this would be $Q_{0\text{x}0018} = \mathcal{E}(0)$). As in the description of PSS in Section 3.0.4, the server takes this value and raises it to the power of the numeric representation of the fingerprint block, modulo the public key. In our depicted example, this would be $\mathcal{E}(0)^{f_0}$. This quantity randomly gets added to some number of buffer bins. Again, adding a value to a buffer bin corresponds multiplication modulo the public key in the encrypted domain. This process is done for every fingerprint block in the database, after which the buffer is returned to the client.

It is unclear offhand which method is better—the first technique has an increased downlink communication cost because the client ends up downloading all matches for all 256 subfingerprints, even if there was actually an exact match for q_0. Thus the buffer has to be very large for correct decoding, to fit all the expected matches. The amount of overhead in this case will depend on database size and the expected number of exact matches per subfingerprint. On the other hand, AudioSearchPSS, depicted in Figure 4.4, has a significantly increased uplink communication cost. If the subfingerprints are 16 bits long, each uplink query will require 2^{16} encrypted bits. Since around 3000 bit public encryption keys are needed for computational security [32], this yields about 195 MB per query. Since the latter approach can require up to 256 private queries, the client could upload as much as 50 GB in the worst case. Clearly neither of these options are desirable. However, at low

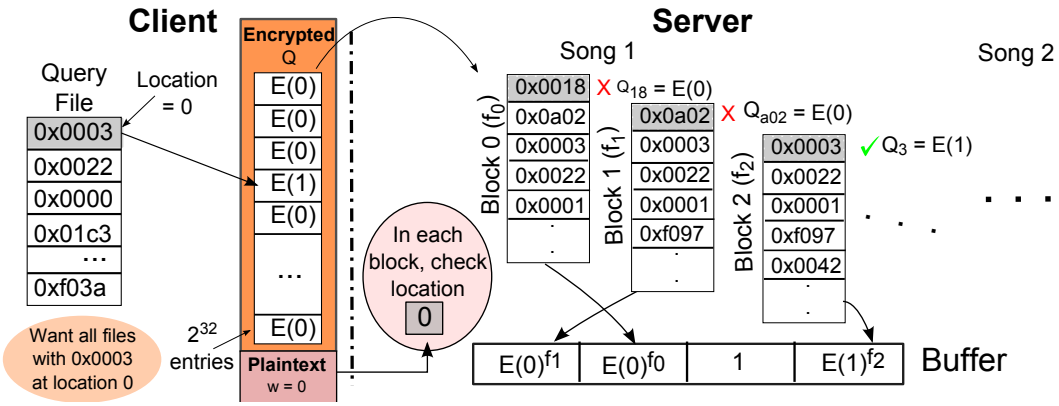


Figure 4.4: Private audio search using PSS. The client submits a new PSS query for each q_w in the query fingerprint block. This figure shows the client's PSS query for q_0 =0x0003. The server cycles through all the fingerprint blocks in the database and checks the first subfingerprint in the block. Two successive fingerprint blocks from the same song are just shifted by one subfingerprint, so they have 255 subfingerprints in common.

noise levels, it is likely that the system will find an exact subfingerprint match in one of the first subfingerprints. Indeed, our application space has relatively high SNR (FM radio typically has SNR levels around 70 dB [43]), so we chose to implement AudioSearchPSS. This Algorithm B.4 is also detailed in Appendix B.

Chapter 5

Audio Search Tool Results

There are three main areas of evaluation in a privacy-preserving media search system. The first is related to the accuracy of the algorithm—independently of the privacy aspect, we wish to evaluate how well the algorithm is able to identify media objects. In this chapter, we will explain why our algorithm gives recognition rates identical to those of the non-private algorithm and discuss what parameters affect those recognition rates. The second area of evaluation is the security of the system. That is, how much does the server learn from executing the protocol? We will show that our PIR-based algorithms are information theoretically secure. The third area is the efficiency of the system. We would like to identify a media file with sublinear communication and computation costs, ideally as close as possible to the non-private costs. We will show that by carefully selecting the size of our subfingerprints, we can reduce the overall communication and computation costs to be sublinear. However, this performance is still less efficient order-wise than the communication costs of the underlying non-private PIR scheme. Note that this scheme is presented in the context of audio matching, but as mentioned earlier, the actual search algorithm is agnostic to the type of feature vectors used. Therefore, the efficiency- and security-related performance characteristics can be thought of in more general terms.

We implemented both AudioSearchPIR:ThresholdSearch (Section 4.2.1) and AudioSearchPSS (Section 4.2.2) to empirically test the accuracy and practical efficiency. The search systems were written in Python, while various interfaces and test scripts were written in MATLAB. Tests were run on an Intel Core i7 machine with four 2.67 GHz processing cores and 3.86 GB of RAM. Our database consisted of 100 songs, representing a variety of music styles. However, because the algorithm divides songs into fingerprint blocks, we are actually dealing with upwards of 60,000 feature vectors (or fingerprint blocks). So if

we were dealing with images with one feature vector per image, this database size would correspond to about 60,000 images.

5.1 Audio Search Accuracy

The recognition rate of the algorithm is totally unrelated to the privacy aspect, at least for the version that uses PIR. This is because PIR schemes deterministically return the desired information, irrespective of implementation. As such, the success rate of the search algorithm is solely a function of parameters like bit error rate threshold and temporal overlap between successive feature vectors. We will begin by discussing the impact of these two factors.

Recall that subsequent subfingerprints are calculated from highly overlapping time windows. This property is good for recognition rates, but it also makes the system more inefficient, since each song is represented by a higher number of subfingerprints. As will be discussed later, the communication and computation costs in this scheme are very heavily impacted by the number of subfingerprints per song; adding more subfingerprints increases both the expected number of exact matches and the number of full fingerprints in the database, which in turn increases the cost of privately querying the database. Thus it is ideal to reduce the number of subfingerprints needed to represent a song as much as possible.

In Haitsma's original scheme the overlap between successive subfingerprints was a factor of 31/32 (i.e. if each subfingerprint lasts 0.37s, then the subfingerprints are separated in time by $\frac{1}{32} \cdot 0.37s \approx 11.6$ ms). However, we found experimentally that much lower levels of overlap yield equally good recognition while significantly speeding up the search. Figure 5.1 shows average precision-recall curves for a variety of overlap factors and query SNRs; these curves are obtained by varying the BER threshold between 0.1 and 0.5. Here we can see that even overlap factors as low as 1/16 do not significantly impact the recognition rate. Rather, the SNR has a much more pronounced effect on the recognition rates. From this data, we also find that BER thresholds of 0.45 or 0.46 tend to give optimal precision-recall performance.

In order to generate these precision-recall plots, we slightly modified the algorithm so that it would always do a worst-case search; that is, instead of recording only the first match below threshold, the system searched through all 256 of the query subfingerprints and recorded the identity of every song that was within the BER threshold. Since the

precision recall curves seem to be much more sensitive to SNR than to overlap factor in subfingerprint generation, we used a coarser subfingerprint (overlap factor of $\frac{1}{16}$) in order to reduce the total amount of data in the database.

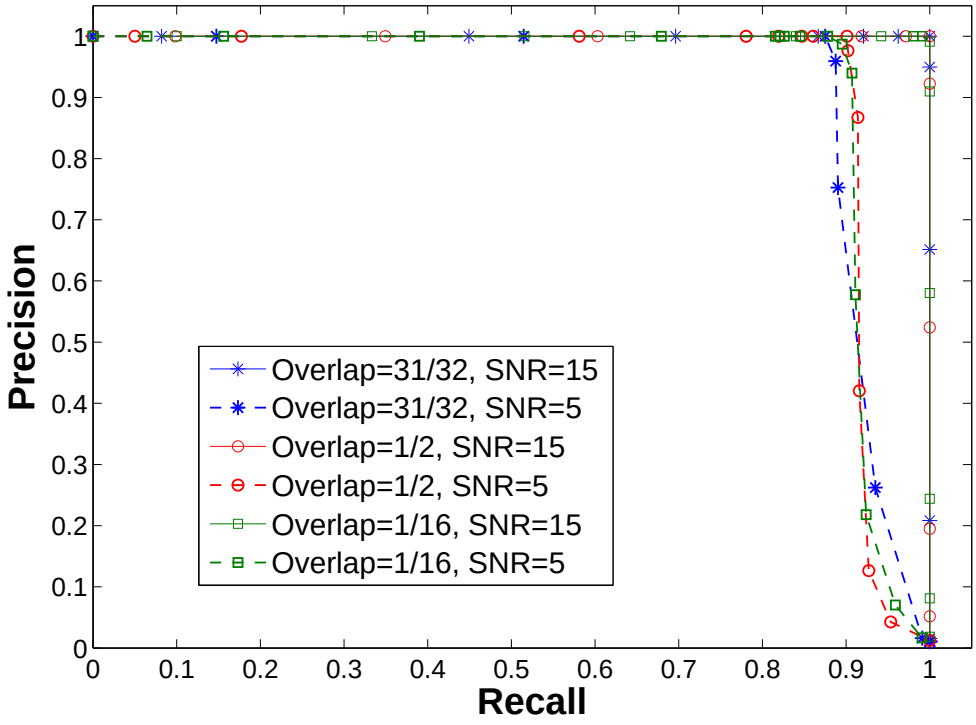


Figure 5.1: Precision-recall curves for subfingerprints with an overlap factor of $\frac{31}{32}$, marked with blue stars, $\frac{1}{2}$ shown with red circles, and $\frac{1}{16}$ shown with green squares, at $SNR = 15$ (solid lines), and $SNR = 5$ (dotted line). We observe that the recognition rates appear to be much more sensitive to SNR than overlap factor.

Once the subfingerprints and BER threshold are fixed, a query can be recognized only if there is at least one exact subfingerprint match. Figure 5.2 gives the empirically determined average number of exact matches and the BER between noisy queries and clean audio samples as a function of query SNR. As expected, the BER depends exclusively on SNR, not the number of bits per subfingerprint. On the other hand, the number of exact matches depends on subfingerprint length as well as SNR—a shorter subfingerprint is more likely to result in an exact match since fewer bits must remain uncorrupted, so the expected number of exact matches is higher.

The average BER allows us to estimate the overall algorithm's recognition rate. The noisy query q can be thought of as the output of a noisy channel. That is, we treat the

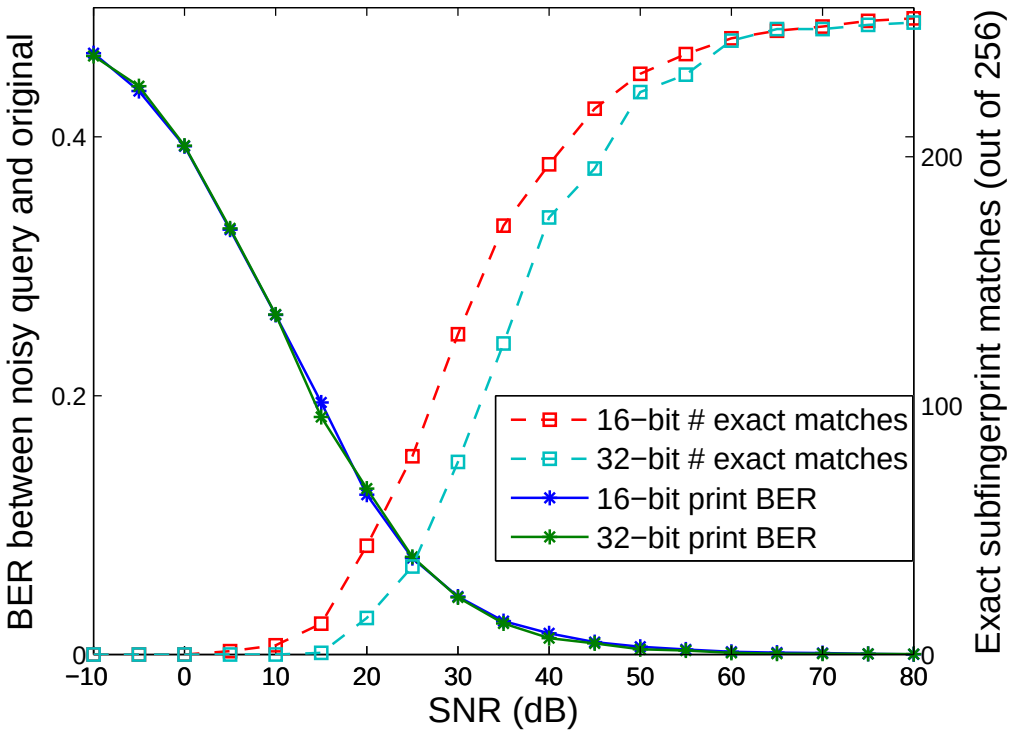


Figure 5.2: Impact of AWGN noise on error rate between the noisy query and the original song. As expected, the bit error rate curve is essentially the same for 16- and 32-bit fingerprints. However, 16-bit fingerprints have a higher number of exact subfingerprint matches.

clean audio fingerprint block as a message f^* that gets passed through a binary symmetric channel (BSC). The transition rate in this BSC is exactly the average BER, denoted by p_e. Note that we can also think of this as $q = f^* \oplus \eta$, where η denotes noise in the form of a Bern(p_e) random process that corrupts the fingerprint block. Then the probability that the source audio track matches the query in at least one subfingerprint is a function of the subfingerprint bit length $k \in \{16, 32\}$, the number of subfingerprints in a query fingerprint $n_s = 256$, and p_e. Define $\text{Sim}(q, f^*)$ as the number of subfingerprints that match exactly between fingerprint blocks q and f^*. So $0 \leq \text{Sim}(q, f^*) \leq 256$. Then the probability of finding at least one exact match between the query and the true match is

$$P(\text{Sim}(q, f^*) \geq 1) = 1 - (1 - (1 - p_e)^k)^{n_s}. \tag{5.1}$$

Recall that having at least one exact subfingerprint match in the query means that the true source audio will be returned by the server as a possible source fingerprint, along with

many other false fingerprint matches. If the total BER between the query and a source audio fingerprint is below the error threshold ϵ, then a match is declared. Given that the query contains at least one exact subfingerprint match, we treat the remaining subfingerprint bits in the fingerprint as independent of those in the exact-match subfingerprint. Each of the remaining bits is independently corrupted with probability p_e, so we must check that the total proportion of corrupted bits is less than the threshold ϵ. Since one full subfingerprint (comprised of k bits) is uncorrupted, the remaining $(n_s - 1)k$ bits can tolerate at most $x = \epsilon n_s k$ errors. For $\epsilon < n_s/(n_s - 1)$ (i.e. essentially always, since ϵ is strictly less than 0.5), it will hold that

$$x < (n_s - 1)k.$$

So if we define $d_b(q, f^*)$ as the number of *bit* errors between q and f^*, then the probability that the remaining $(n_s - 1)k$ bits contain at most x errors is precisely

$$P\Big(d_b(q, f^*) \leq x | \text{Sim}(q, f^*) \geq 1\Big) = \sum_{i=0}^{\lfloor x \rfloor} \binom{(n_s - 1)k}{i} p_e^i (1 - p_e)^{(n_s-1)k-i} \tag{5.2}$$

Combining expressions 5.1 and 5.2, we get the probability that the true source can be identified as a match by the algorithm. In these calculations, we assume that the BER threshold is low enough that the probability of false matches is negligible. This gives

$$P(\text{recognition}) = \Big(1 - (1 - (1 - p_e)^k)^{n_s}\Big) \sum_{i=0}^{\lfloor x \rfloor} \binom{(n_s - 1)k}{i} p_e^i (1 - p_e)^{(n_s-1)k-i} \tag{5.3}$$

The threshold ϵ is encapsulated in the maximum allowable number of bit errors, x. Equation 5.3 represents an upper bound on the probability of finding an exact match, since there is always the possibility of a false match being found before the true match gets returned to the client. Note that for practical purposes, the binomial cdf can be replaced by a Gaussian approximation without much loss of accuracy, since the number of total bits is quite high.

If ϵ is set high, our model loses some accuracy because the probability of a false match is high, whereas we assumed it to be negligible. This is clear in Figure 5.3, which illustrates empirical and theoretical recognition rates as a function of SNR for $\epsilon = 0.45$. This threshold is chosen as the optimal threshold in our precision-recall curves from earlier (Figure 5.1); however, it turns out that at such a high threshold and low SNR, the noise in the system occasionally maps the wrong fingerprint block to the query, but the BER is still within threshold. Interestingly, this phenomenon usually gives a correct answer, despite matching the query to the wrong subfingerprint block. We explain this by observing that most

music contains patterns, meaning that there may be several similar audio segments in a song. Demonstrating the extent of this phenomenon, the 'Experimental, Adjusted' curve in Figure 5.3 represents the recognition rate when the query was matched to the correct fingerprint block in the database. At higher SNR levels, this is always the case for a given match, and indeed we observe that our theoretical estimate of recognition rates represent an upper bound on the observed rates.

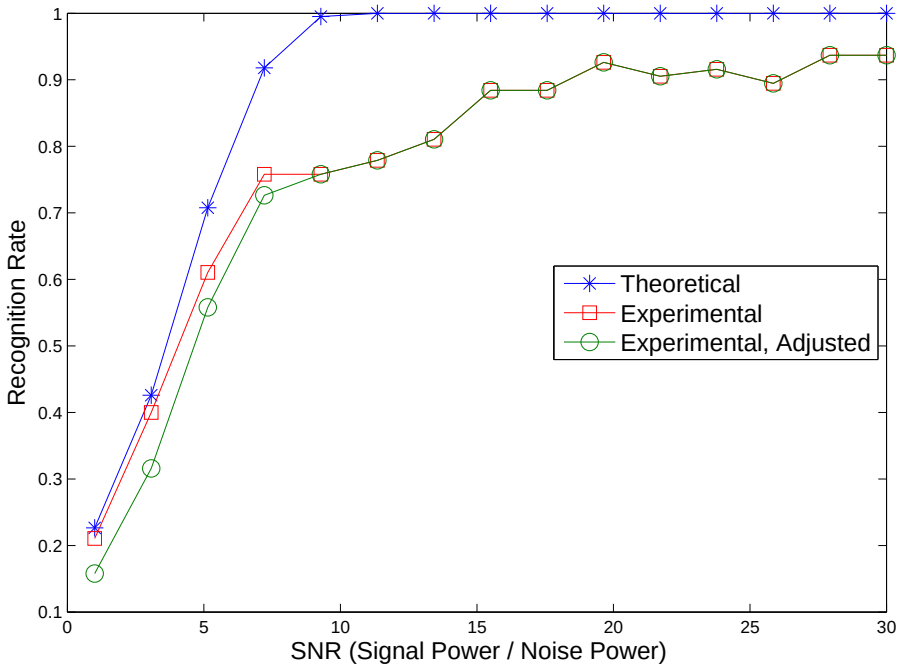


Figure 5.3: Recognition rate as a function of SNR. The experimental results, plotted in red, illustrate the recognition rates from 95 trials, with 16-bit subfingerprints and $\epsilon = 0.45$. The adjusted experimental results, plotted in green, show the recognition rates if we strip off the 'false matches', i.e. the instances of an incorrect fingerprint block being matched to the query, but giving the correct result nonetheless because the incorrect block comes from the correct song.

5.1.1 PSS Recognition Rates

In the PSS scheme discussed earlier, the probability of successfully recovering all matching documents is lower than 1 if there is more than one match in the database. However, the probability of successfully returning all files can be made arbitrarily close to one at the expense of increased downlink communication. Specifically, if we expect to see m matches

in the database, then the return buffer must be of size γm where γ is a reliability parameter; Finiasz and Ramchandran showed that γ can be chosen as order $O(1)$ while still recovering all matches with high probability [22]. Note that if one elects to use the PSS scheme as a proxy for single-server PIR, there is never more than one result, so full recovery is guaranteed with a single buffer bin.

5.2 Security

We now provide a security proof for a two-server PIR implementation of algorithm AudioSearchPIR:FullSearch (Section 4.2.1). We will then show that under certain conditions, AudioSearchPIR:ThresholdSearch is also information theoretically secure. The proofs for the other modifications we've mentioned (like AudioSearchPIR:LookupTable or AudioSearchPSS) are analogous, though the PSS version can at most be computationally secure. As with most privacy-preserving search systems of this variety, we assume a *semi-honest*, passive adversary, who follows protocol exactly, but may attempt to gain additional knowledge by analyzing received information. As one might expect, the proof stems entirely from the security of PIR. In this case, we have two parties: the client $\mathcal{C}$ and the servers $\mathcal{S}_v$, where $v \in 1, 2$.

Observation 1. *The 2-server PIR Scheme in Section 3.0.3 is information-theoretically secure.*

The proof of this is quite straightforward, but it is provided in Appendix A for completeness. This observation allows us to show the following:

Observation 2. *Under a semi-honest adversary model, AudioSearchPIR:FullSearch (Algorithm 4.1) evaluates without leaking any information about the query to any server $\mathcal{S}_v$.*

The proof of this is also included in appendix A.

As we saw earlier, the approach of AudioSearchPIR:FullSearch is somewhat inefficient, since in practice we can stop searching once a match within a threshold BER is found. Recall that we called this alternate approach of threshold matching 'AudioSearchPIR:ThresholdSearch'. However, threshold-matching does leak some information because each server learns how many query subfingerprints were sent by the client before finding an exact match. This is reflective of the amount of noise in the system, as well as the susceptibility of the client to noise. Such information may not be important in practice, but it is nonetheless leaked, so we should give it full consideration.

For instance, for a given SNR level (which the server might know from other sources), suppose a client does not find an exact match until the 254th subfingerprint. The server might deduce that the frequency content of the client's noise-free audio clip was particularly uniform over the frequency bands, thereby causing the system noise to alter the relative energies in the different bands more than usual, and skewing the query subfingerprints. Knowing the frequency-domain characteristics of the client's sound clip could in turn help the server identify the query song from the database.

We will now show that under AudioSearchPIR:ThresholdSearch, no semantic information is leaked if a prior assumption about the robustness of subfingerprints is met: For a given noise level, the probability a client sees an exact match on a given subfingerprint should be independent of the underlying audio file. Of course, this assumption is unlikely to hold in practice, since it would imply a perfect feature representation. However, we experimentally observe this property to hold on average—that is, there may be individual fingerprint blocks that are noticeably more or less robust than the majority, but on average, noise resistance is constant. In any case, if this assumption is too stringent and the leaked information is unacceptable, one can always revert to AudioSearchPIR:FullSearch for information theoretic security.

Definition Let fingerprint block f^* be the closest match in database $\mathcal{DB}$ to query fingerprint block q. Then we define *semantic information* as any information X such that $I(X, f^*|\mathcal{DB}) > 0$.

Observation 3. *Suppose that for a given noise level, the probability a noisy query subfingerprint is an exact match is independent of the original (non-noisy) audio segment. Then under a semi-honest adversary model, AudioSearchPIR:ThresholdSearch (Algorithm B.1) evaluates without leaking any semantic information about the query to any server $\mathcal{S}_v$.*

Proof. Any semi-honest server $\mathcal{S}_v$ in the system is able to view each input message Q^v from the client, as well the full database. We wish to prove that given this information, each server $\mathcal{S}_v$ learns no semantic information from $\mathcal{C}$. From observation 1, we know an information-theoretically secure PIR scheme exists; let $\mathcal{P}$ denote such a scheme. From observation 2, we know that each individual PIR query leaks no information to the server.

Once a match is found with BER below threshold ϵ (at query subfingerprint index $w^* \leq 255$), $\mathcal{S}_v$ does learn that the first $w^* - 1$ of $\mathcal{C}$'s query subfingerprints are not exact matches. We call this information Y and show that Y is disjoint from the semantic information of the query signal, i.e. that $I(Y, f^*|\mathcal{DB}) = 0$.

It holds that $I(f^*, Y|\mathcal{DB}) = H(f^*|\mathcal{DB}) - H(f^*|Y, \mathcal{DB})$. Recall our model, in which we represent the query vector as $q = f^* \oplus \eta$. We will not use this model to prove security because we want to be more general, but we will retain the notion of independent noise. Suppose that η_R is some independent, real-valued noise of arbitrary distribution that gets added to the noise-free audio file f_R^*. Note that f_R^* is the underlying audio signal, not the feature representation of the audio signal. Then the client generates a quantized feature vector q by doing some signal processing on the received signal $f_R^* + \eta_R$. This more general model of the system better reflects what is happening in real life, though our simplified model was experimentally shown to adequately upper bound system performance in practice.

Then we have $H(f^*|Y, \mathcal{DB}) \geq H(f^*|\eta_R, Y, \mathcal{DB})$ because conditioning on more variables reduces entropy, so

$$\begin{aligned} I(f^*, Y|\mathcal{DB}) &\leq H(f^*|\mathcal{DB}) - H(f^*|\eta_R, Y, \mathcal{DB}) && (5.4) \\ &= H(f^*|\mathcal{DB}) - H(f^*|\eta_R, \mathcal{DB}) && (5.5) \\ &= I(f^*, \eta_R|\mathcal{DB}) = 0 && (5.6) \end{aligned}$$

where 5.5 results because Y can be completely reconstructed from η_R and $\mathcal{DB}$, and 5.6 results because η_R is independent of f^*. Note that $\mathcal{DB}$ consists of all the information accessible to the server, including the original audio files of all the songs. This is an upper bound on the server's information, since it would most likely only store the feature vectors for each song. Thus we have

$$\begin{aligned} &0 \leq I(f^*, Y|\mathcal{DB}) \leq I(f^*, \eta_R|\mathcal{DB}) = 0 \\ \implies &I(f^*, Y|\mathcal{DB}) = 0, \end{aligned}$$

and $\mathcal{S}_v$ learns no semantic information from the client in the process of operating AudioSearchPIR:ThresholdSearch. □

5.3 Resource Consumption

The point of implementing this system is to show that it is possible to obtain good results without accruing computation and communication costs linear in database size. The previous section illustrated that we can achieve recognition rates comparable to the non-private algorithm; in this section we will examine the associated additional costs.

5.3.1 Feature Distribution

The speed of this algorithm as a private media search tool will depend in part on the distribution of features in the appropriate vector space. In the public domain, an algorithm can handle searching through a larger pool of feature vectors with relative ease, but in the private domain, each query takes a much longer time. If most of the files have similar feature vectors, then the expected number of exact matches per subfingerprint will be $O(n)$, with a leading constant tending to 1; in the limit as the distribution of feature vectors converges to an impulse, the number of false exact matches will be exactly $n - 1$, which is the worst efficiency possible. For this reason it is critical that each subfingerprint have the lowest number of matches possible, and the distribution that minimizes the maximum pmf value is the uniform distribution. Figure 5.4 shows the distribution of the 2^{16} possible subfingerprints in our database. This histogram indicates that the subfingerprints are not biased at a large scale, but there are certainly some subfingerprints that happen much more often than others, and this can lead to long search times. In designing such a system, it is worth spending some time to make sure that whatever feature vectors are being used have a distribution that is as uniform as possible.

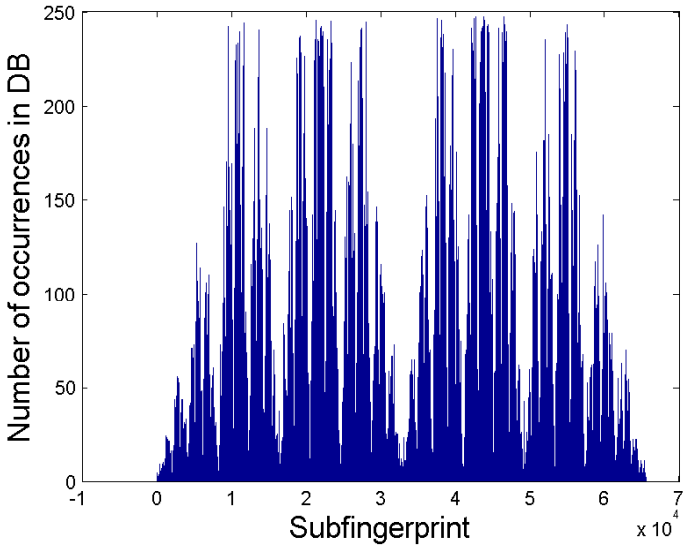


Figure 5.4: Histogram showing how often each print occurs in the database of 96 songs. For this histogram, each print is chosen to overlap by a factor of $\frac{1}{2}$. Ideally, we would like this distribution to be uniform.

5.3.2 Communication Costs

Once we select operating parameters for the audio search scheme, the goal is to make the search as efficient as possible. For PIR, we will constrain the theoretical discussion to AudioSearchPIR:FullSearch, with the understanding that one can improve efficiency by leaking some noise-related information.

In its basic form, the privacy-preserving schemes that rely on PIR have a worst-case (and average) communication complexity that is $O(\max(p(k), m(n))$, where n is the database size and $m(n)$ is the expected number of exact subfingerprint matches in the database; k is the number of bits in each subfingerprint, and $p(k)$ is the total communication complexity of a PIR search on a list of k-bit subfingerprints. If k is constant, then the expected number of matches will be $O(n)$ assuming subfingerprints are distributed uniformly; in this case the total communication is dominated by the downlink cost of sending matching fingerprints back to the client. That being said, it is even better to choose k according to database size; if k is chosen as $O(\log n)$, then the expected number of matches will scale as $O(1)$ with database size, and the dominant communication cost will come from the uplink private queries. Note that even in the non-private domain, this is useful because it means that the server need only process a constant-order number of files, which is a very desirable property for low latency. Thus we will use this assumption in all of our successive cost calculations. Using the two-server PIR scheme from 3.0.3, $p(k)$ has complexity $O(2^k)$. That means the total communication is still linear in n, but the point is that more efficient PIR schemes can reduce the communication to polynomial levels of degree strictly less than one. A similar effect is exploited in [28], leading to their sublinear communication complexity.

The communication in the PSS scheme is much higher than that in the PIR scheme. However, as we have mentioned earlier, [24] shows that it is possible to frame the PSS problem in such a way that the downlink communication cost is asymptotically equal to the non-private downlink communication, i.e. $O(m(n))$. We will use the communication costs from [24] for our analysis, though our implementation used a slightly suboptimal version of their scheme (regular LDPC codes rather than irregular).

On the communication side, the worst-case scenario is if we search through all the subfingerprints. In that case, assuming we have a public key with $N = 4096$ bits and $k = 16$ bit subfingerprints, our uplink communication will be $4096 \cdot 2^{16} \cdot 256 = 68$ gigabits. Note that for practical security, we would need encryptions of 3000 bits [32]. On the downlink, the expected number of matches per subfingerprint is $m(n) = n/2^k$, assuming a uniform

distribution of subfingerprints in the database. If we then use Finiasz and Ramchandran's scheme, the communication cost reduces to $(N \cdot n_s \cdot 2^k + n_s \cdot n/2^k)$. The dominant factor here depends on the database size n and how k is selected. If the number of subfingerprint bits k is kept constant, the linear dependence on n will dominate, giving $O(n/2^k)$. On the other hand, if we choose the number of subfingerprint bits as order $\log n$ as before, then $m(n)$ will be $O(1)$, causing uplink costs to dominate. This gives a total communication cost of $O(2^k) = O(n)$. As before, we get a communication cost linear in n, but in this case, the constant factor N is extremely large and adds at least three orders of magnitude to the cost.

We can lower bound the communication costs of the privacy-preserving scheme by thinking about the communication complexity of a non-private scheme. Irrespective of implementation, any non-private scheme (using these features) must ask the client to transmit enough information uplink about the query fingerprint block to allow the server to find a corresponding match in its database with BER below the threshold. The server must then transmit enough information downlink to tell the client which song corresponds to the client's query. The dowlink cost can be thought of as constant—the length of a song title does not vary with database size. In [27], the uplink cost is $256k$ bits; however, under the assumption that the query is highly correlated with a fingerprint block in the database, information theory tells us that the uplink cost can be reduced. Recall that we modeled the query fingerprint block q as the result of passing a fingerprint block f^* in the database through a BSC with bit transition probability p_e. In this case, the Slepian-Wolf theorem implies that the uplink communication need only be $H(q|f^*) = H(p_e) \cdot k \cdot n_s$. Again, this assumes that successive bits in the fingerprint block are iid, which is not the case in practice; as such, the required communication could theoretically be reduced even further. However, lacking a good model of the fingerprint block statistics, this estimate is an upper bound on the minimum possible uplink communication. In any case, the order of required total communication remains unchanged: the non-private scheme must transmit $O(k)$ bits. For a fair comparison with the private scheme, we again assume that k is again chosen as $O(\log(n))$. A logarithmic scaling may be superfluous in the non-private scheme, but at the very least, k should scale with database size in order to retain the discriminative power of features. That is, if k remains constant as the database grows, at a certain point, the feature space will become too densely populated with database entries for accurate classification.

A comparison of the communication (and computation) costs in the PIR, PSS, and public cases is shown in Table 5.1. The key observation is that using current technology,

Privacy	Privacy Scheme	Communication	Computation
PIR	2-server PIR, basic [15]	$O(n)$	$O(n)$
	2-server PIR, covering codes [15]	$O(\sqrt[3]{n})$	$O(n)$
	2-server PIR, [19]	$O(\sqrt[3]{n})$	$O\left(\frac{n}{\log^2 n}\right)$
PSS	Single-server PSS [24]	$O(n)$	$O(n \log^2 n \log\log n)$
Public	n/a [27]	$O(\log n)$	$O(\log n)$

Table 5.1: Comparison of communication and computation costs for privacy-preserving schemes and the original non-private version in [27]. For consistency, we choose $k = \log n$ in both the private and non-private versions. We note that current PIR schemes have communication costs at best polynomial in n compared to logarithmic scaling in the non-private case.

neither of the privacy-preserving schemes can approach the $O(\log n)$ communication costs of the non-private scheme, even order-wise. However, the communication in PIR schemes is dominated by the cost of PIR, so the existence of PIR schemes with very slow polynomial growth could allow privacy-preserving searches to be viable on relatively small scales. The computation complexities provided are explained in greater detail in Section 5.3.3.

5.3.3 Computation Costs

The computation in PIR-based schemes is dominated by server-side operations. This is better than high client-side computation for two reasons. First, servers are better suited than clients to handle heavy computation, particularly because the required computations can be easily parallelized. Second, weighty private queries can be made more efficient with better PIR schemes, whereas portions of the algorithm unrelated to privacy cannot be trimmed as easily. Since PIR schemes are an active area of research, it makes sense to try to offload as many resources as possible to the private query portion of the algorithm, under the assumption that these algorithms are likely to improve with time.

The total computation cost using PIR is $O(max(p_c(n), m(n) \cdot k))$, where $p_c(n)$ is the computational cost of the PIR scheme used, and $m(n)$ is still the expected number of exact matches per subfingerprint. The latter term reflects client-side computation; the client must calculate the BER between each returned fingerprint block and the query fingerprint block. For each private query, the client receives $m(n)$ expanded fingerprint blocks of size $2n_s - 1$, where $m(n) \in O(1)$ when $k \in O(\log(n))$. Extracting the correct fingerprint block from the expanded fingerprint block requires only memory access because the client knows

exactly where in the expanded fingerprint block the desired subfingerprint should appear (i.e. it will always be precisely in the middle, by construction of the database). Computing the BER between two fingerprint blocks is an operation requiring $O(2n_sk)$ FLOPS, so the total client-side computation is $O(k) = O(\log n)$. The value of $p_c(n)$ depends on which PIR scheme is used. For the basic one presented in Section 3.0.3, the computation arises from projecting the length-2^k query vector onto the database contents, which requires the bitwise addiction of about half the files in the database. This gives a complexity of $O(n)$. However, schemes such as those proposed in [19] provide lower-complexity PIR schemes that preserver sublinear computation (in addition to sublinear communication), as shown in Table 5.1.

The real bottleneck in PSS comes from the computational costs. Where the PIR scheme executes an XOR operation, the PSS scheme executes an expensive modular exponentiation. We estimate the cost of a modular multiplication as $O(N^2)$ for an N-bit public key, and a modular exponentiation as $O(N^2 \log n)$. Our PSS scheme requires one modular exponentiation per fingerprint block, and each fingerprint block contains n_sk bits. In our implementation, we divided fingerprint into chunks of 2 subfingerprints to make the modular exponentiation more manageable; these separate chunks were stored in aligned sub-buffers. This gives 128 modular exponentiations on items of length $2k$ bits. Thus, adding a single fingerprint block to the buffer requires $(4k^2 \log(2k) \cdot n_s/2)$ FLOPS. There are n fingerprint blocks in the database, so each private query requires $O(k^2 \log(2k)n)$, of which there are n_s in the worst case scenario. If we again assume that $k \in O(\log n)$, then the total server-side computation is $O\left(n \log^2 n \log\log n\right)$. As in the PIR scheme, the client-side computation is $O(\log n)$. This is clearly heavily outweighed by the server side computation, so the total PSS computation is $O\left(n \log^2 n \log\log n\right)$.

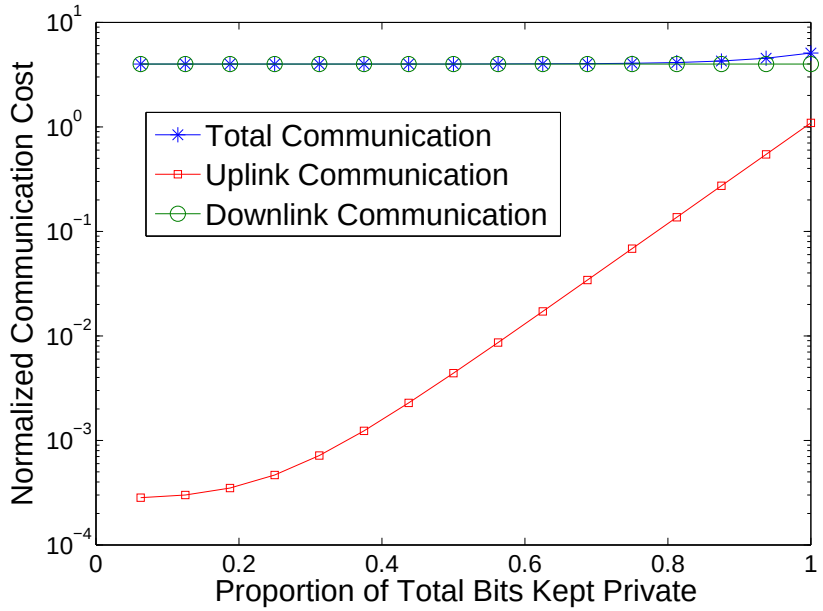
Now we compare this computation to the non-private case. Non-private computation costs occur entirely at the server side. We expect to see $m(n)$ matches per query subfingerprint, which is reduced to $O(1)$ by the same argument as before, and the BER must be calculated between the query and each match. As we saw in the PSS case, calculating BER is $O(k)$ in complexity, and this operation is expected to occur a constant number of times. Thus the complexity of the public scheme is $O(k)$, which is $O(\log n)$ by our assumption .

Trading Privacy for Performance

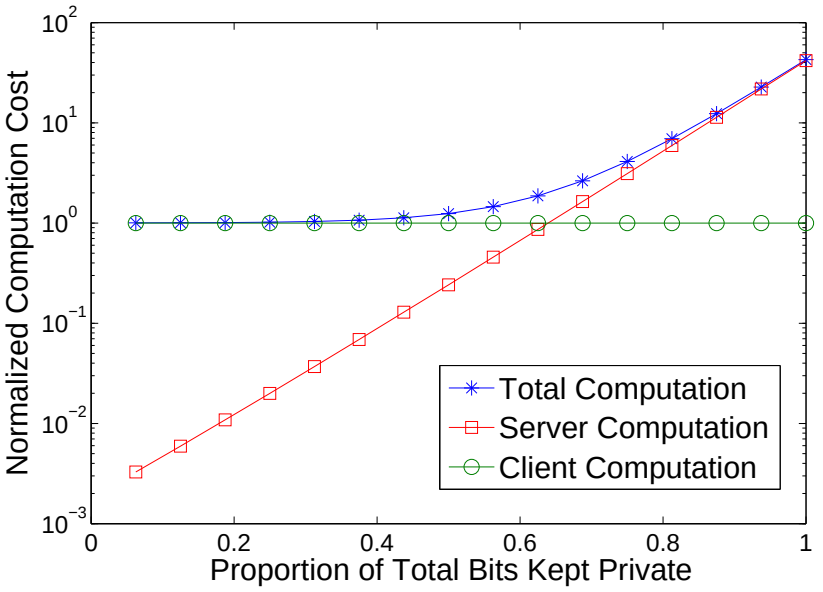
As one might expect, there is a tradeoff between cost and privacy, and we can obtain efficiency gains by reducing privacy and revealing a number of subfingerprint bits to the server. This effectively reduces the size of the database by reducing the number of possible query targets, which in turn reduces both the communication and computation required. For instance, reconsider the example of AudioSearchPIR:ThresholdSearch from Section 4.2.1, in which our first query subfingerprint was '0x0003'. We can reduce the privacy of the PIR query by telling the server in plaintext that the first two hexadecimal digits of our desired subfingerprint are '00'. Then the effective size of the database is reduced to 2^8, since the server only needs to search through subfingerprints of the form '0x00??'. This reduces the size of the uplink PIR query to a search for the last two symbols '0x03', which is good because uplink PIR is the overall communication bottleneck, as we saw in Section 5.3.2. Since the effective size of the database is reduced, the server also has to do less computation to process each PIR query, so the computation costs are also dramatically reduced.

Figure 5.5 shows the worst-case communication and computation in AudioSearchPIR:ThresholdSearch (the values are the same for AudioSearchPIR:FullSearch) as a function of the proportion of private bits to total bits in each subfingerprint. Both communication and computation are normalized by the worst-case cost of running the non-private algorithm. It is important to note that we can achieve communication and computation costs that are within an order of magnitude of a public search by keeping only a fraction of the bits private. This fraction will vary with the application, but the point is that we do not necessarily always need full privacy. For instance, in Figure 5.5, we see that the total system computation is the same order of magnitude as non-private computation if we reveal at least half the bits to the system.

Now we will briefly mention the impact of this technique on the PSS implementation. Partial bit-sharing motivated our design for AudioSearchPSS; recall that the algorithm submits a separate PSS query for each of the 256 noisy subfingerprints instead of just submitting one big PSS query for all the subfingerprints at once. This is because all the subfingerprints will almost certainly have different leading bits, so revealing them to the server would not allow the server to execute the private search on a subset of the database—it would essentially just leak information. That is, the server would still be forced to search through the entire database sequentially, which is very inefficient. For this reason, AudioSearchPSS just submits one partially masked PSS query for each subfingerprint until



(a) Worst-case communication cost as a function of private bits.



(b) Worst-case computation cost as a function of private bits.

Figure 5.5: Worst-case resource consumption in AudioSearchPIR:FullSearch as a function of the proportion of subfingerprint bits kept private.

it finds a match. Each of these masked queries can be comparatively cheap because the server only needs to search in a small subset of the database.

To illustrate the benefits of this approach in practice, Figure 5.6 gives mean experimental run times from 10 trials of both AudioSearchPIR:ThresholdSearch and AudioSearchPSS as the proportion of private subfingerprint bits grows. The results in Figure 5.6 clearly reinforce the notion that PIR-based schemes are significantly more efficient than PSS-based schemes in practice. However, the most salient point of this figure is that the PIR scheme's runtime is within one order of magnitude of the public runtime when the proportion of private bits is kept below about 0.7. From a practical standpoint, this already starts to suggest ultimate feasibility, via better PIR schemes, for example. However, this technique should be applied with extreme caution. As we will see shortly, even revealing a few bits in plaintext can have a significant (negative) impact on information-theoretic privacy, though the threat may be negligible in practice.

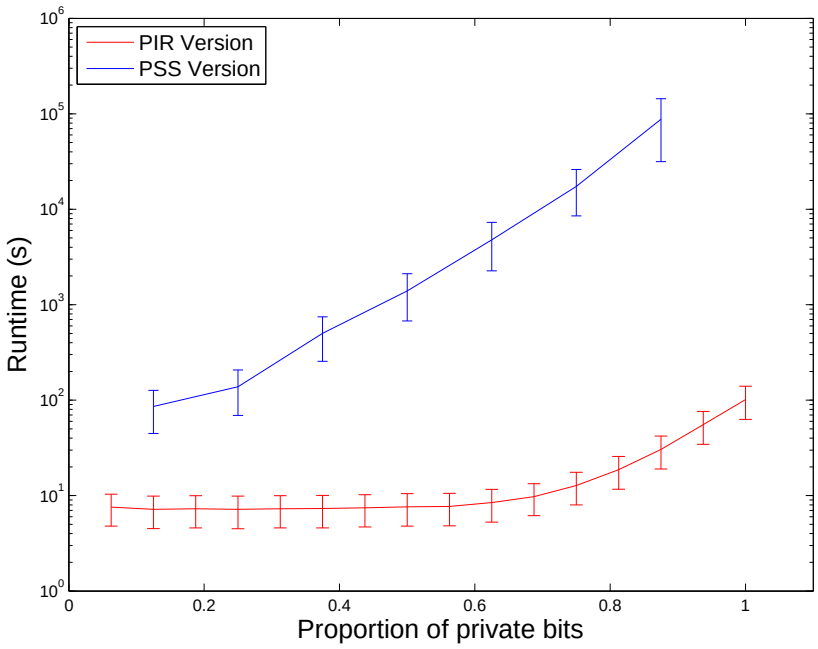


Figure 5.6: Mean runtime as a function of proportion of bits kept private. The times are normalized by the mean time for a non-private search. We see that PIR drastically outperforms PSS as the percentage of private bits increases. More importantly, for relatively low proportions of private bits (below about 0.7), the PIR scheme's runtime is within an order of magnitude of the non-private query. In these trials, $SNR = 15$, and averages are taken over 10 trials.

Impact of bit-sharing on privacy

Note that reducing the number of private bits has an unclear effect on overall privacy, because the fingerprint blocks will most likely not be uniformly distributed for real datasets. It could be the case that certain fingerprint blocks are completely defined by their leading bits, and this will ultimately depend on the database and the type of feature used (since this algorithm can also be used on media that is not audio). We can derive bounds on system parameters in order to ensure a certain level of client privacy by using our prior model. Again, think of each noise-free fingerprint block in the database as a distinct codeword in a code, and the client's query block as a noise-corrupted version of that codeword that it sends to the server. Then we can use results from coding theory to think about the ability of the server to accurately decode f^* based on leaked bits. The server will treat received bits as a (possibly noisy) subset of codeword bits, and it will treat the private bits as channel erasures. This interpretation is shown in Figure 5.7 for a toy example in which there are two subfingerprints of four bits each. $\mathcal{C}$ reveals two bits per subfingerprint to the client, so the client receives a partial codeword. Given that there is no noise in the system, i.e. $p_e = 0$, the number of codewords that could have generated q are $2^4 = 16$.

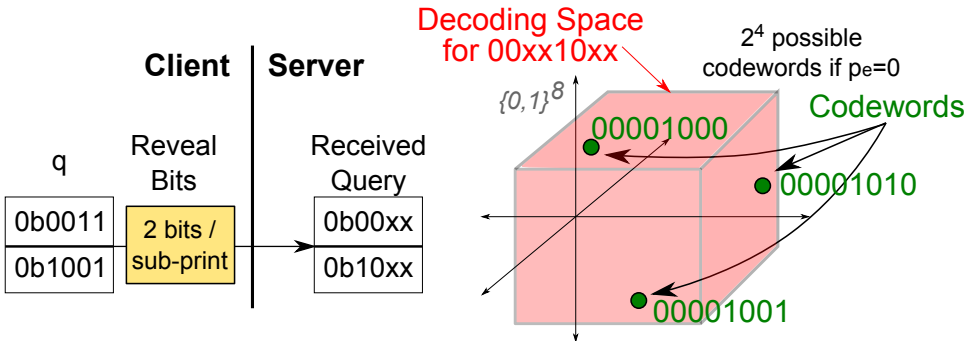


Figure 5.7: Toy example of server decoding partially revealed bits with infinite SNR, i.e. $p_e = 0$. The received codeword has four 'erasures', so the server can narrow down the possible matches to $2^4 = 16$ equally likely codewords in the decoding space, assuming codewords are drawn uniformly.

An $(n_s k, \log n)$ code C is defined as a subspace of $n_s k$-dimensional space in which we embed a $\log n$-dimensional message; $2^{\log n} = n$ is the number of possible messages in the code, and $2^{n_s k}$ is the total number of possible strings in the space available for our code.

Definition The *minimal distance* (d_{min}) associated with code C is defined as the smallest

distance between any two codewords:

$$d_{min} := \min\{d(v, w) : v, w \in C, v \neq w\}.$$

Observation 4. *Suppose $q = f^* \oplus \eta$, with $q, f^*, \eta \in \{0,1\}^{n_s \times k}$ and binary noise vector η having some probability mass function $P_\eta(\eta)$. Suppose r plaintext bits of query q are leaked to server $\mathcal{S}$. Define d^*_{min} as*

$$\begin{aligned} d^*_{min} := \underset{d_{min}}{\textit{maximize}} \quad & d_{min} \\ \textit{subject to} \quad & \sum_{i=0}^{d_{min}-1} \binom{n_s k}{i} \leq 2^{n_s k - \log n} \\ & 0 \leq d_{min} \leq n_s k. \end{aligned}$$

Then as long as $r < n_s k + 1 - d^*_{min}$, $\mathcal{S}$ cannot decode f^*.

Proof. $\mathcal{S}$ receives some subset of r bits from q, and the rest are unknown. Independently of the decoding algorithm, S can decode any number of errors v and erasures e as long as the condition

$$d_{min} \geq 2v + e + 1 \tag{5.7}$$

is satisfied. This is because a code with $n_s k$-bit codewords and minimal distance d that undergoes e erasures can be thought of as a shorter codeword with $n_s k - e$ bits and minimal distance $d - e$. Then by geometry, the shortened code can only correct $(d - e - 1)/2$ transmission errors.

Now, there are $n_s k - r$ erasures total. We also have a distribution on the number of errors v. Given d_{min}, we can learn exactly how many revealed bits r are sufficient for $\mathcal{S}$ to decode f^* with a given probability; that probability stems entirely from the randomness of η. However, we wish to make the system immune to η; $\mathcal{S}$ should always be unable to decode. To this end, we should keep enough erasures in the query print that $\mathcal{S}$ can never correct the codeword, regardless of the number of bit errors. To do so, we choose d_{min} as large as possible, since that is a worst-case scenario for privacy; the codewords are far apart, so each leaked bit gives maximal information about the underlying codeword. The distribution of codewords that maximizes the minimum distance between codewords is the uniform distribution. Thus we have $2^{n_s k} - n$ strings that are not occupied by codewords. We can find an upper bound on d_{min} for *any* code by simply counting the number of items at Hamming distance 1, 2, etc. from each uniformly spaced codeword, until all the unused strings are accounted for. For instance, there are $\binom{n_s k}{1} n$ strings at a distance of 1 from some

codeword, and so forth. This gives

$$\begin{aligned} d^*_{min} = \underset{d_{min}}{\text{maximize}} \quad & d_{min} \\ \text{subject to} \quad & \sum_{i=0}^{d_{min}-1} \binom{n_s k}{i} \leq 2^{n_s k - \log n} \\ & 0 \leq d_{min} \leq n_s k. \end{aligned}$$

Since we want to prevent decoding even in the absence of noise, we choose $v = 0$ and get that as long as

$$r < n_s k + 1 - d^*_{min}, \tag{5.8}$$

$\mathcal{S}$ can never decode f^*. This holds for any code, which means that there are no restrictions on the distribution of fingerprint blocks or the nature of the noise η. □

Computing the solution to this optimization problem for practical system parameters ($k = 16$, $n_s = 256$, $n = 1 \times 10^6$) is somewhat tricky because the combinatorials get very large very quickly. However, we can approximate the solution with Stirling's Approximation, which states that $\log n! \approx n \log n - n$. Since a sum of combinatorials will be dominated by the largest term in the summation, we can apply this aproximation to the largest combinatorial term. This gives a different optimization problem that is computationally tractable:

$$\begin{aligned} d^*_{min} \approx \underset{d_{min}}{\text{maximize}} \quad & d_{min} \\ \text{subject to} \quad & n_s k \log(n_s k) - n_s k + \log n \geq \\ & (n_s k - d_{min} + 1)\log(n_s k - d_{min} + 1) + (d_{min} - 1)\log(d_{min} - 1) \\ & 0 \leq d_{min} \leq n_s k. \end{aligned}$$

For the suggested system parameters, this approximation gives $d_{max} \approx 1881$, so we take $r \leq n_s k + 1 - d_{min} = 2216$. That is, as long as we leak $r < 2216$ bits total or 8.6 bits per subfingerprint, then the server can *never* decode f^*, irrespective of the number of bit errors in the shared bits. In our 16-bit subfingerprint, sacrificing 8 bits per subfingerprint is more than enough to reap significant practical gains in both computation and communication. If the client sends more bits than this bound, it is not necessarily true that the server will be able to determine f^*. There may be too many bit errors for $\mathcal{S}_v$ to decode, especially if p_e is close to 0.5, and this probability is easy to compute. However, this will depend on the noise distribution.

Now, even if the server cannot decode exactly, it may have a good and narrow idea of which codewords could be the possible sources. To briefly address this, we assume that η is

a Bern(p_e) random process. Suppose that $0 < p_e < 0.5$. Then using a maximum-likelihood decoder (since we assume the channel is BSC), $\mathcal{S}_v$'s best estimate of f^* is any element from the set of codewords with minimum Hamming distance from the received vector. The server obtains from all this a likelihood function for q parameterized by f^*. Let $\tilde{q}$ denote the bits of q that get sent in plaintext, and let $\tilde{\eta}$ and $\tilde{f}$ denote the bits of η and f, respectively, at indices corresponding to the plaintext bits of q. Let $w(\cdot)$ be the weight function of a binary vector, or the sum of all the elements in the vector. Then

$$\begin{aligned}\mathcal{L}(f|q) &= \Pr_{\tilde{\eta}}[\tilde{\eta} = \tilde{f} \oplus \tilde{q}] \\ &= (p_e)^{w(\tilde{f}\oplus\tilde{q})}(1-p_e)^{r-w(\tilde{f}\oplus\tilde{q})}.\end{aligned}$$

The server can estimate f^* with the maximum likelihood codeword (at least in principle—this is a very computation-intensive operation in practice), and also determine the likelihoods of all the other codewords. In particular, all codewords f resulting in the same value of $w(\tilde{\eta})$ have the same likelihood, so the server cannot distinguish between them. For each value of $w(\tilde{\eta})$, there are $\binom{r}{w(\tilde{\eta})}2^{(n_s k - r)}$ binary strings f that give the same likelihood; for optimal server confusion, the sets with higher likelihood should contain as many codewords as possible. But this corresponds to grouping codewords closer together, which reduces the discriminative power of the algorithm.

It may be worthwhile to sacrifice this extra bit of privacy in favor of practically viable run-times. Broadly, the privacy threat we are considering is that servers (or third parties with access to server records) will deduce private information about users, essentially by mining search data from clients. Because these searches collectively represent a tremendous amount of data, even partial masking can be useful in practice. That is, it may not be necessary to completely block the server from knowing the nature of the query; rather, it may be sufficient to simply render the task of deducing the client's query very expensive. This is the ultimate justification for partial privacy. For instance, maximum likelihood decoding in this case reduces to a nearest-neighbor problem, which currently has solutions that are at best exponential in the dimension of the space. This offers some form of computational security, even when theoretical guarantees may be too weak to be satisfactory.

Chapter 6

Generalized Privacy-Preserving Media Matching

The Haitsma and Kalker search algorithm is a useful example because only minimal changes are required to make the algorithm privacy-preserving. This situation is by no means typical, and good systems will usually require special search algorithms offering both private-domain compatibility and resource efficiency. Designing such algorithms can involve introducing quantized features that are conducive to exact comparison, changing actual search mechanisms, or both. In this chapter, we will start by discussing some techniques of feature vector and search algorithm modifications that can make media search algorithms easier in the private domain; this is followed by a sample face recognition tool that relies on these techniques and also meshes nicely with the audio search tool just presented.

As discussed in Chapter 2, many feature-based media recognition algorithms search for nearest-neighbor feature vectors in a database [9], [10], [25]. Our private audio search took the slightly different approach of searching for all database feature vectors that match the query vector exactly in at least one index. Exact matches are convenient for the private query aspect of the algorithm, but it is intuitively clear that nearest-neighbor searches will generally give better recognition results. This is because for most nearest-neighbor searches, the optimal vector will almost never match the query vector exactly at any of its indices. This raises an important question: Since nearest-neighbor search algorithms are not naturally suitable for the private domain, is it possible to slightly alter them for private query compatibility without suffering in performance? It turns out that the answer is yes, to some extent. In [9], [10], [25], the most computationally intensive portion of the algorithm is

the Euclidean distance calculation and comparison. In the user-private scenario, we could (in principle) let the client execute all these computations, since the database is public. However, the client will most likely have limited computational capabilities compared to the server, making this solution impractical.

A potential solution to this question originates from the signal processing community. A popular research topic in recent years aims to represent arbitrary feature vectors in a way that reduces nearest-neighbor searches to a Hamming distance comparison, which is computationally lighter in both the private and non-private domain. A lot of this work relies on locality-sensitive hashing techniques that provide probabilistic noise resistance. There are different ways to implement these hashes. For instance, [44] and [45] have explored a simple but effective hashing technique: Given two normalized vectors with some distance between them, project both vectors onto a set of random hyperplanes and record the sign (+/-) of each projection for each vector. It turns out that the Hamming distance of these binary hash vectors is a probabilistic estimate of Euclidean distance. Indeed, Min et al. found this technique to give 95 percent accurate nearest-neighbor estimates at a fraction of the cost for high-dimensional features like GIST [45]. We will examine this technique in greater detail shortly. Other less-studied feature modifications may be introduced specifically for a given application; examples include the quantization techniques used in [34] and distance-preserving hashing in [33] These examples are drawn from the two-way privacy literature since there are not many one-way private systems in existence.

Another (complementary) tactic for improving privacy-preserving media searches is to modify or specifically design search algorithms for private search. This is done to varying degrees in most privacy-preserving media search systems [9], [10], [25]. An excellent example of an algorithm that was fully designed specifically for the private domain is the SCiFI system by Osadchy et al. [32]. By defining faces as a collection of items from dictionaries of face parts and part locations, the authors facilitate comparison in the encrypted domain. Specifically, the recognition step is defined by whether a set difference (a Hamming distance calculation) is above or below a threshold. This kind of search algorithm overhaul can lead to better recognition rates and fewer unnecessary algorithm inefficiencies. One important point is that the SCiFI algorithm is tailored to a particular domain, i.e. face recognition. From a global point of view, designing algorithms for general media recognition will most likely involve a combination of 1) transforming arbitrary media information to a privacy-conducive format (e.g. hashed feature vectors) and 2) finding efficient ways to search through transformed data. While SCiFI is not a general purpose media search

tool, its underlying principles are similarly aligned with this design philosophy. We will now present an example of a potentially more general application for the one-way private domain, implemented in the form of a nearest-neighbor face recognition system. Before presenting that system, we will examine the distance-preserving hash of [44] in greater detail.

6.1 Distance-preserving hashes via random projections

Suppose we want to hash two vectors v_1 and v_2, with $v_i \in \mathcal{R}^\ell$, $i \in \{1, 2\}$, and a Euclidean distance between the normalized vectors of $\delta_{1,2}$. The goal is to set up a system in which we have noise-resistant "subfingerprints" just as in the audio search tool, so we will try to obtain n_s subfingerprints with k bits in each, as before. We begin by projecting both vectors onto a set of $n_s k$ random hyperplanes h_i, with $i \in \{1, ..., n_s k\}$ and $h_i \in \mathcal{R}^\ell$. We then record the sign (+/-) of each projection. It is straightforward to show that the probability that v_1 and v_2 have different hash bits for a single random projection is $p_{1,2} = \frac{2}{\pi} \sin^{-1} \frac{\delta_{1,2}}{2}$. After $n_s k$ projections onto random hyperplanes, each original vector v_i is hashed to a new $n_s k$-bit feature ($H(v_1)$ and $H(v_2)$).

The overall objective is to use this hash to determine which feature vector from the set $\{v_i | i = 1, ..., n\}$ is closest in Euclidean distance to query feature vector v_q; recall that n is the database size. By projecting both the database and the query hash vectors onto the same $n_s k$ random hyperplanes, the ratio of Hamming distance to total number of projections is a probabilistic estimate of $p_{i,q}$, call it $\hat{p}_{i,q}$.

$$\hat{p}_{i,q} = \frac{\text{Hamm}(H(v_i), H(v_q))}{n_s k}, \quad i \in \{1, ..., n\}$$

With some manipulation, we see that our best estimate of $\delta_{i,q}$ given $\hat{p}_{i,q}$ is

$$\hat{\delta}_{i,q} = 2 \sin(\frac{\pi}{2} \hat{p}_{i,q}) \tag{6.1}$$

In Equation 6.1, $\hat{\delta}_{i,q}$ is monotonically increasing on the interval $\hat{p}_{i,q} \in [0, 1]$, so it must hold that if the pairs of vectors (v_i, v_q) and (v_j, v_q) have different separation distances $\delta_{i,q} \neq \delta_{j,q}$, then $\hat{p}_{i,q} > \hat{p}_{j,q} \iff \hat{\delta}_{i,q} > \hat{\delta}_{j,q}$. This is convenient for our private search, because it means we can simply compare Hamming distances of hashes rather than estimated Euclidean distances, without altered results.

After determining the $n_s k$ hash bits for all images in the system, we can rearrange these bits into a matrix by subdividing the hashes into n_s subfingerprints of length k bits each,

and search for exact matches. It can be ensured that there will be at least one exact match with a given probability by choosing n_s and k appropriately, as discussed in Section 6.3.1. As before, the distance metric is Hamming distance, thanks to our distance-preserving hashing. So Haitsma and Kalker's algorithm can be used just as before to conduct private face recognition. A graphical representation of this hashing process is provided in Figure 6.1.

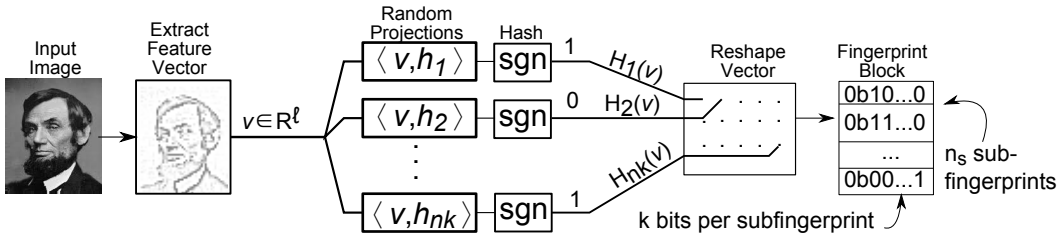


Figure 6.1: Method for converting arbitrary feature vectors into "subfingerprints". This facilitates a private search similar to the one used in the previously-described audio search. h_i denotes a randomly-drawn ℓ-dimensional hyperplane, where ℓ is the the original feature vector length.

6.2 Case Study: Face Recognition

This hashing and subfingerprinting technique can be used on any algorithm that requires a nearest-neighbor search on a feature vector space. To demonstrate this, we implemented a basic face recognition system that relies on random projection hashes in the context of the Fisherfaces algorithm [46]. This system was executed on a smaller scale than the audio search tool and meant primarily as a proof of concept, so a MATLAB implementation was sufficient for both desired speed and database size. We implemented only a PIR-based version, and our algorithm is analogous to AudioSearch:ThresholdSearch from Section 4.2.1.

6.2.1 Fisherfaces

We will start by describing the implementation and motivation for Fisherfaces. The Fisherfaces algorithm was designed because Eigenfaces has poor performance on varied lighting conditions, facial expressions, and occlusions like hair and glasses. Ideally, we would like a facial recognition algorithm to classify the same person consistently, even under different circumstances.

To do this, first of all, the training data in Fisherfaces should contain multiple images from each individual under different conditions. Next, recall that the Eigenfaces algorithm determined the basis of eigenfaces by taking the singular value decomposition of the training data. Mathematically, this gives the projection directions that maximize the scatter across all images, and is identical in concept to principal component analysis (PCA). We will provide here the explanation given in [46]. Consider N sample images $\{x_1, \ldots, x_N\}$ belonging to C classes $\{X_1, \ldots, X_C\}$. Define the total scatter matrix S_T as

$$S_T = \sum_{k=1} N(x_k - \mu)(x_k - \mu)^T$$

where $\mu \in \mathcal{R}^N$ is the mean image for the samples. In Eigenfaces, these image vectors are projected onto a projection matrix W, giving dimensionality reduction. This matrix is chosen as that which maximizes the determinant of the total scatter of the features, i.e. it must solve the optimization problem

$$W_{opt} = \underset{W}{\text{argmax}} |W^T S_T W|.$$

However, in practice, we observe that the appearance of a face varies much more across different lighting conditions than across people. Therefore, Fisherfaces proposes a slightly different projection matrix, that maximizes the scatter *between classes*. To do this, the algorithm tracks the means of the training data in each class $\mu_1, \ldots, \mu_C$, as well as the total mean μ. Then it defines scatter matrices for within-class scatter S_W, and between-class scatter S_B:

$$S_W = \sum_{k=1}^{C} N_i(\mu_i - \mu)(\mu_i - \mu)^T$$

$$S_B = \sum_{k=1}^{C} \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T$$

where N_i is the number of items in class i. Then the projection matrix for Fisherfaces is chosen as follows:

$$W_{opt} = \underset{W}{\text{argmax}} \frac{|W^T S_B W|}{|W^T S_W W|}.$$

The basis W_{opt} has the property of minimizing the within-class variance while maximizing the between-class variance, which is exactly what we want. This approach has been shown to increase the recognition rate significantly with respect to Eigenfaces [46]. However, we want to point out that once the new basis is found, the algorithm is identical to Eigenfaces—all images are projected onto this basis, after which the server conducts a nearest-neighbor search in the database.

6.2.2 Privacy-preserving adaptation

For our privacy-preserving version, we extract the Fisherfaces feature vectors and hash them as shown in Figure 6.1. Note that this essentially gives us fingerprint blocks just as in the audio search tool. With these features, we use the same search technique, relying on AudioSearchPIR:FullSearch. The choice to use a full search rather than a threshold search was made because the noise in a face recognition system is less regular than in the audio application; some people's appearances are more homogeneous than others' in the training set. If we were to use a threshold, we would have to train different thresholds for different classes. For this reason, we simply cycle through all the subfingerprints and choose the closest match.

For the face recognition tool, we only implemented the PIR version by modifiying the Face Recognition Evaluator Toolbox for MATLAB written by Brian Becker and Enrique Ortiz [1]. This was applied to the AT&T database of faces [47], which contains a database of 40 faces over a spread of 10 images each, totaling 400 images in the database.

6.3 Results

Once the feature vectors are hashed, the computation and communication demands of the privacy-preserving face recognition scheme are identical to those of the privacy-preserving audio search tool, so we will not cover those concepts again. However, unlike the PIR-based audio search tool, our privacy-preserving algorithm does not guarantee equivalent recognition rates to the non-private version. The random projection technique can be probabilistically driven to obtain recognition rates that are close—but not identical—to the nearest-neighbor approach; this is done by increasing the total number of random projections $n_s k$. After executing the random projections, we are also subdividing the hashes into $k-$bit subfingerprints and then searching for exact matches. Using the properties of the random hashes, we can consider the probability of finding an exact match as a function of the number of subfingerprints and subfingerprint length k. In this section, we will discuss the factors that impact both the accuracy of our hashed scheme and the overall runtime.

[1] `http://www.briancbecker.com/bcbcms/site/proj/facerec/fbeval.html`

6.3.1 Accuracy of Hashing Scheme

Stepping back to the random hashing scheme, the accuracy of the algorithm depends entirely on how good our estimate of $p_{i,j}$ is. To some extent, we can quantify the quality of this estimate. The XOR of hash vectors $H(v_i)$, $i \in \{1, ..., n\}$ and $H(v_q)$ can be thought of as a Bernoulli random process with mean $p_{i,q} = \frac{2}{\pi} \sin^{-1} \frac{\delta_{i,q}}{2}$ and variance $\sigma^2 = p_{i,q}(1 - p_{i,q})$. Thus we can use the Central Limit Theorem (CLT) to determine how good our estimate of $p_{i,q}$ is for a given number $n_s k$ of realizations. Again, $\text{Hamm}(H(v_i), H(v_q))$ denotes the Hamming distance between the two hashes; then the CLT tells us that

$$\hat{p}_{i,q} = \frac{\text{Hamm}(H(v_i), H(v_q))}{n_s k} \sim \mathcal{N}(p_{i,q}, \frac{p_{i,q}(1 - p_{i,q})}{n_s k}) \tag{6.2}$$

Due to the variable nature of $p_{i,q}$, the variance (and mean) of this random variable will depend on the original distance between the two vectors, so the same number of hashes $n_s k$ will yield tighter confidence intervals for certain distance estimates than others. However, we can upper bound the variance with the quantity $\sigma_{p_{i,q}} = \frac{1}{4n_s k}$, which is obtained by setting $p_{i,q} = \frac{1}{2}$, thereby maximizing the quantity $p_{i,q}(1 - p_{i,q})$. For the database vectors closest to the query vector, the separation distances will be relatively small, so $p_{i,q}$ will most likely be much smaller than $\frac{1}{2}$, yielding an even tighter confidence interval. It is unclear how this translates into a distribution on $\delta_{i,q}$ in general. However, for sufficiently small $p_{i,q}$, we can approximate that

$$\sin\left(\frac{p_{i,q}\pi}{2}\right) \approx \frac{p_{i,q}\pi}{2}, \qquad \text{for } p_{i,q} \ll \frac{1}{2}$$

which in turn implies that

$$\delta_{i,q} \approx p_{i,q}\pi, \qquad \text{for } p_{i,q} \ll \frac{1}{2}$$

Thus for database feature vectors close to the query feature vector in Euclidean distance, we can approximate the distribution of the distance estimate as follows, using an upper bound on variance:

$$\hat{\delta}_{i,q} \sim \mathcal{N}(\pi p_{i,q}, \frac{\pi^2}{4n_s k}) \qquad \text{for } p_{i,q} \ll \frac{1}{2}$$

This expression indicates that we can estimate the Euclidean distances between normalized feature vectors as accurately as desired, at least for database vectors that are close to the query. However, this model does not allow a direct theoretical comparison between the hashed and unhashed Fisherfaces algorithms, because in the original Fisherfaces algorithm, the projected vectors are *not normalized*. Normalization does not significantly affect the results, since scaled versions of the same feature vector simply reflect different intensities in the original image; in practice, it is ultimately the angle of the feature vector that gives

the most information about an image's composition. This is easily observed by running the Fisherfaces algorithm on both unnormalized and normalized projection vectors, and observing that the recognition rates are almost identical. However, the small difference in recognition rates makes direct comparison difficult. It also ensures that even as the number of hashed bits grows unboundedly, the hashed Fisherfaces algorithm will never perform quite as well as the regular Fisherfaces algorithm.

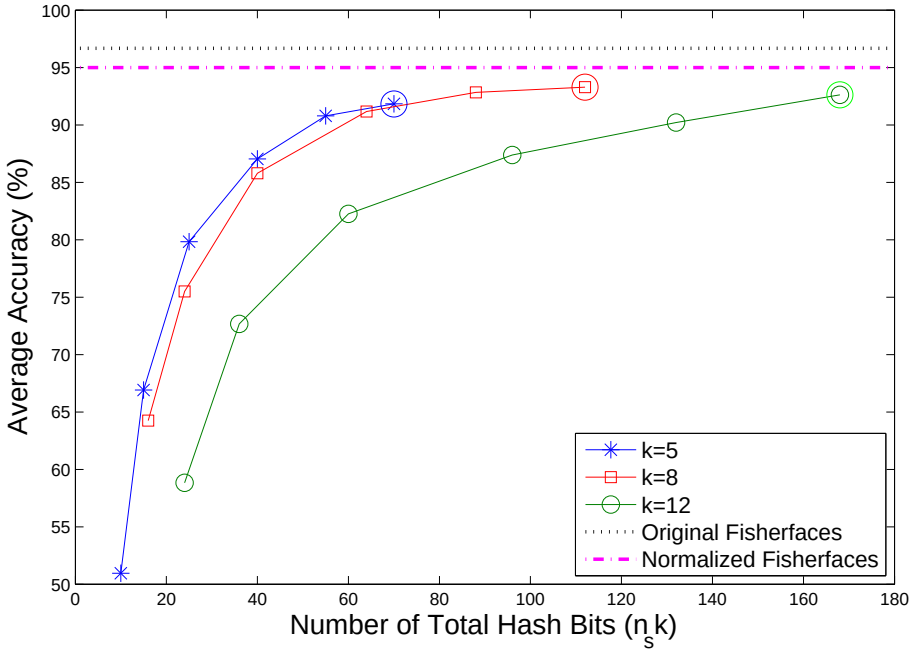


Figure 6.2: Mean recognition rate as a function of total number of hash bits, parameterized by subfingerprint length k.

Figure 6.2 shows the empirically-determined probabilities of finding the correct match using both the original Fisherfaces and hashed Fisherfaces algorithms. We randomly divided the dataset into 70 percent training data and 30 percent test data. The circled data points are markers to facilitate comparison between the accuracy plots in Figure 6.2 and the time plots in Figure 6.4. As expected, increasing the number of total hash bits drives recognition rates toward the non-private, normalized rates. For a fixed number of hash bits $n_s k$, larger subfingerprints (i.e. larger values of k) drive down the recognition rate by reducing the probability of finding an exact match. However, as will be shown later, it can be useful to increase k in order to reduce the runtime. One important point is that given enough subfingerprints, these random projections can actually achieve comparable performance to the original Fisherfaces algorithm with a notable overall reduction in the number of bits per

feature. For instance, in our setup when $k = 8$ and $n_s k = 112$, we achieve average accuracy levels within 4 percentage points of the public scheme, while using only 8.9 percent of the bits per feature vector (112 hash bits per feature, versus 39 Fisherfaces x 32 bits each = 1248 bits per feature for the unhashed version).

Choosing fingerprint block dimensions

In the audio search tool, the fingerprint block dimensions were chosen according to the Haitsma and Kalker specifications. However, in the hashing scheme, the designer must choose subfingerprint size k and number n_s. As illustrated in Figure 6.2, these choices do affect overall accuracy.

Observation 5. *Let $N = n_s k$, and suppose we want the probability of getting no exact matches to be at most $\epsilon > 0$. This holds if and only if*

$$\frac{1 - p_e}{\epsilon^{1/N}} > (\epsilon^{-k/N} - 1)^{1/k}. \tag{6.3}$$

This is easy to show using the probability expression from equation 5.1. One way to use this expression in practice is to solve the inequality graphically for a given N. This approach is shown in Figure 6.3 for $\epsilon = 0.0001$, the probability of error $p_e = 0.25$, and the number of total hash bits $N = 120$. The figure indicates that in order to satisfy the desired error probability, it must hold that $k \leq 5$.

6.3.2 runtime

The theoretical computational and communication complexity is the same as before, since the search algorithm itself remains unchanged. The results from Chapter 5 still apply. However, we would like to provide a very brief comparison with some benchmark privacy-preserving face recognition benchmark systems. Table 6.1 gives asymptotic communication and computation costs for this face recognition scheme as well as two benchmark two-way privacy schemes [10], [32]. Strictly speaking, this comparison is not fair since the two-way private schemes solve a fundamentally harder problem than ours. In addition, the SCiFI system also has much higher accuracy than the eigenfaces algorithm. Nonetheless, we wish to highlight that order-level gains in asymptotic efficiency can be had by exploiting one-way privacy.

Finally, we demonstrate the effect of feature hashing on the query time. The test runtime is primarily limited by two main factors: the expected number of matches per PIR query,

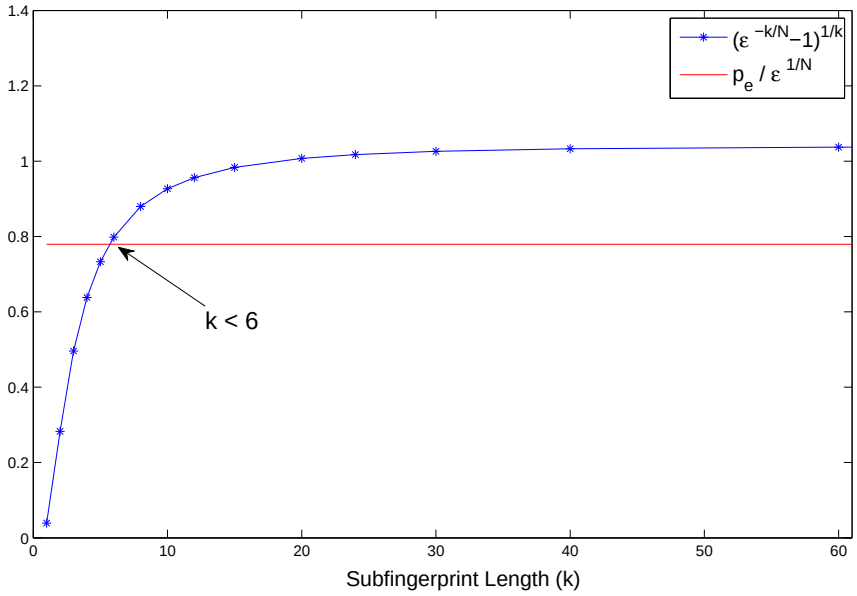


Figure 6.3: Graphical selection of subfingerprint size for a fixed number of hash bits. The selection technique shows how to choose k in order to upper bound by ϵ the probability of not getting any exact matches in the subfingerprint search.

and the uplink size of the PIR query. A high expected number of matches will increase the downlink communication of false matches to the client, which increases the amount of unnecessary computation. The expected number of exact matches is a function of both database size and subfingerprint size, since smaller subfingerprints increase the likelihood of finding an exact match, whether correct or otherwise. To reduce these inefficiencies, one can use larger subfingerprints in order to make each subfingerprint more discriminative. However, this has the effect of reducing the expected number of exact matches, but it also increases the size of the PIR query exponentially. The larger the PIR query, the more XORs that the server must execute, which increases the computational loads on the server. Thus, for a given number of total hash bits, inefficiencies result from overly large or overly small subfingerprints; the key is to choose k between the two extremities. Also, recall that larger subfingerprints can drive down recognition rates for a fixed number of hash bits.

The average normalized runtime over 10 trials is shown in Figure 6.4 for different levels of constant total hash bits. The times are normalized by the runtime for the public algorithm. To give an idea of the actual runtimes, each non-private query took an average of 2.92×10^{-4} s to run, so even private tests running 3 orders of magnitude slower still required less than a second to run. We can clearly see the trend mentioned above; very small and very

Algorithm	**Privacy Scheme**	**Communication**	**Computation**
Hashed Eigenfaces	2-server PIR, [19]	$O(\sqrt[3]{n})$	$O\left(\frac{n}{\log^2 n}\right)$
Eigenfaces [10]	HE, Garb. Circ. [10]	$O(n)$	$O(n)$
SCiFI [32]	Oblivious Transfer [48]	$O(n)$	$O(n)$

Table 6.1: Face recognition asymptotic communication and computation costs for privacy-preserving face recognition; 'Hashed Eigenfaces' refers to our suggested scheme, and we also compare this to the online complexities of two benchmark face recognition systems [10], [32]. We chose $k = \log n$ in the hashed scheme, where k is subfingerprint length and n is the number of faces in the database.

large subfingerprints drive up the runtimes significantly. The circled data points represent the same fingerprint block configurations as the circled points in Figure 6.2; under these conditions, we can expect a level of accuracy within 5 percentage points of the unmodified Fisherfaces algorithm. Note that for a given plot line in Figure 6.4 (i.e. for a fixed number of hash bits $n_s k$), moving to the left—using smaller subfingerprints—will improve the accuracy by increasing the probability of finding exact matches. Thus for a fixed number of total hash bits, it is best to use the smallest subfingerprints possible while maintaining runtimes within tolerance. As demonstrated in Chapter 5, these runtimes can be further reduced by allowing only partial privacy, i.e. by revealing some bits to the server, while masking the rest. Again, the effects of partial privacy are the same as in the audio scenario, since the search algorithm on the hashed image feature vectors is identical to the search algorithm on the audio feature vectors.

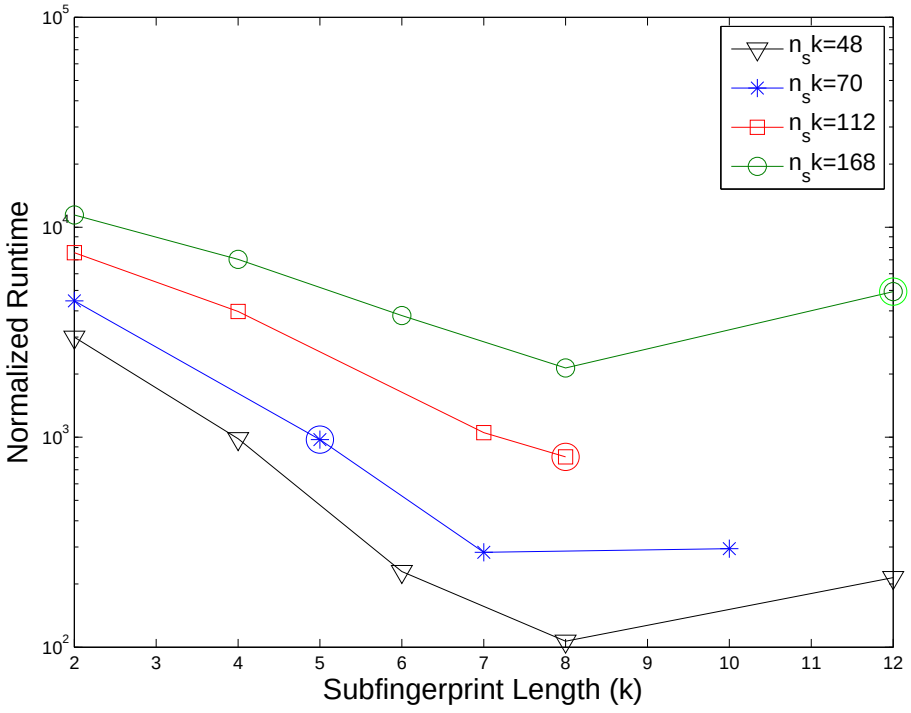


Figure 6.4: Mean runtime as a function of subfingerprint size k, parameterized by the total number of hash bits $n_s k$. The circled data points correspond to the circled points in Figure 6.2, to give an idea of the computational costs associated with a particular level of accuracy.

Chapter 7

Conclusion

This work features a variety of privacy-preserving media recognition tools that rely on private queries. The first contribution is a privacy-preserving audio recognition tool that adapts an existing audio search algorithm to operate with one-way privacy tools. Specifically, it was demonstrated that such a tool can operate within an order of magnitude as fast as a non-private scheme by using partial privacy settings—that is, by slightly reducing the privacy afforded to the client. Subsequently, themes from this audio search tool were used to design a general scheme for privacy-preserving nearest-neighbor searches. This approach was implemented in a sample face-recognition tool relying on the Fisherfaces algorithm. We demonstrated that our privacy-preserving nearest-neighbor search can lead to relatively efficient search times, with only small degradations to detection accuracy. The current section will first describe two potential directions for the continuation of this work, related to the notion of making these techniques practically viable. Then we will conclude with a discussion of challenges that remain in the area of one-way privacy-preserving media searches, with a brief overview of some interesting research directions with promising results.

7.1 Computationally Efficient Two-Way Privacy

A good portion of this thesis was spent describing the importance of one-way privacy. However, there are certainly many applications for two-way privacy as well. The approach proposed in this work could easily be extended to give two way privacy, as long as the basis vectors are public. Recall that in the audio search algorithm, the server only leaks information to the client by sending falsely matching fingerprint blocks to the client. However, suppose one could prevent the client from reading a fingerprint block unless it was within

the specified threshold of the client's noisy query. This setup lends itself well to a coding solution.

The Slepian-Wolf theorem tells us that we can encode correlated sources with fewer than the bits required to encode each source individually. In fact, in the language of the model from before, if f^* is the optimal matching fingerprint block in the database, then there exists a code that allows the server to transmit as few as $H(f^*|q) = H(\eta)$ bits while still achieving reliable communication. To achieve such rates, capacity-approaching codes like LDPC codes can come arbitrarily close to this bound. Meanwhile, the client can (with high probability) only decode a block efficiently if it is within a specified BER from the client's own query block. This offers the server security in a probabilistic sense.

This approach is particularly appealing, because it allows for harnessing the improved efficiencies of PIR schemes, but it nonetheless gives two-way privacy. Recall from Table 6.1 that the proposed PIR-based scheme achieves asymptotically lower communication and computation than existing two-way private schemes. While existing schemes are linear in database size, our scheme can achieve sublinear communication and computation by using more efficient varieties of PIR.

7.2 Making PIR Practically Viable: A P2P Approach

A major and valid criticism of privacy-preserving searches is that currently, there is little to no incentive for web servers to alter their systems to accommodate private queries. In fact, there is a disincentive for them to do so, because servers presumably increase advertising revenue by knowing what clients search for. Given this situation, is there any hope for introducing PIR on a wide scale? To address this question, we should mention a fact that was omitted earlier. Most PIR schemes with many servers do not require zero collusion. Instead, they typically guarantee information-theoretic security as long as no more than s of the d servers collude [15]. Given the rise of distributed peer-to-peer networks, one could imagine many small, client-run proxy nodes acting as PIR servers, with guarantees of information-theoretic security as long as no more than s of these proxies collude. For the moment, let's consider a system that deals with text-based queries, unrelated to media searches.

The first obvious question is how to get the relevant data on the cache servers. The search databases are tremendously large, and one cannot hope to store all that information

on a cache node. However, caches can use existing servers like Google, Bing, and Yahoo to field queries for files they do not possess. For instance, Figure 7.1 illustrates how this might work in a two-cache system. For instance, consider a 3-item database, and the client wants the i=1 index. The client generates a normal PIR request for the '1' index, just as in the PIR example from Section 3.0.3. However, the masked requests go to two proxy servers rather than two main servers. If the proxy node already has all the files specified by the query vector, it just returns the appropriate XOR of those files. Otherwise, it requests the missing files from the main servers, then computes the XOR.

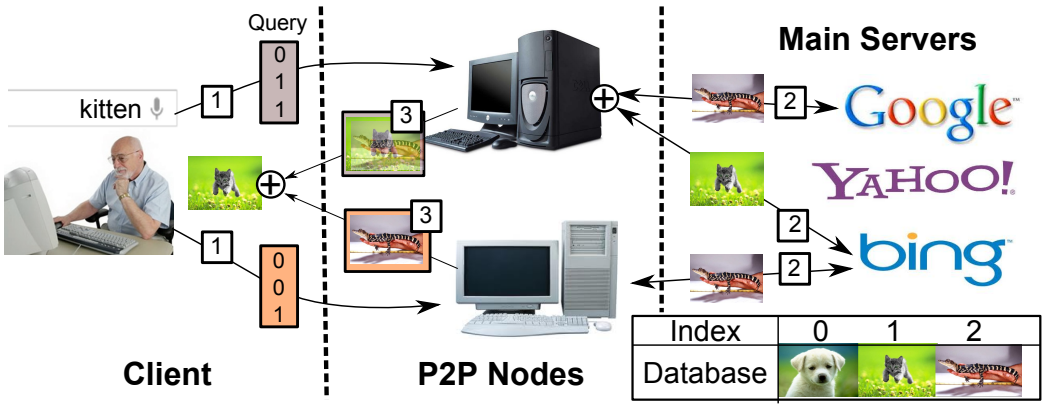


Figure 7.1: Peer-to-peer service providing search anonymity.

The main question, of course, is how to deal with the heavy bandwidth and computation requirements, from the viewpoints of the client, proxy nodes, and main servers. The broad scheme described above is not an adequate solution on its own—if each proxy node had to query half the database contents each time, the bandwidth usage would quickly overwhelm the system. However, it would be possible to split up the database into smaller chunks and assigning each proxy node to one such chunk. For instance, one proxy might be in charge of all queries about movies, while another might receive all queries about medical conditions. This reduces the privacy level per query, but is likely sufficient in most cases, especially if these chunks contain documents spanning a variety of topics. Such a splitting mechanism would require well-thought-out load management. If one file in a chunk is more popular than all the others, then the proxy server can guess which file is desired by the client. However, if all the popular files are given to a single proxy node, then that node will have to use more resources than the other nodes. Database segment allocation to proxy nodes is an interesting question with many implications for the overall fairness and efficiency of the system.

Another source of significant gains depends on the number of proxy PIR servers; a

peer-to-peer system is well-suited to exploit the communication and computation gains resulting from having many servers. For instance, if the database has n files, the client-side communication can scale with $\sqrt[d]{n}$, where d is the number of total servers [15]. There are similar gains that can be had computationally, using schemes like [19], though the scaling laws are different. Thus it would be possible to significantly reduce the client-side communication and server-side computation by having a very large number of servers.

It is not clear whether such a P2P setup could ultimately be practical. However, variations of this idea have been considered in the past by Papadopoulos et al. [49] and Müller et al. [50], for instance. However, [50] addresses the fairly specific use case of 'privacy of ideas'; that is, if a client searches a database for a unique or uncommon combination of keywords, the server could infer the client's thoughts regarding a new patent or stock market trading, for example. The resulting PIR scheme is designed around this notion. While this may be a valid concern, we would ideally like to make a system that is more general, along the lines of [49]. However, even Papadoloulos et al. rely on computational PIR schemes, which ultimately are much less efficient than information-theoretic schemes and do not harness the order-level gains that can be had with multiple servers. Specifically, it would be interesting to explore the possibility of a system in which the communication and computation costs are as far sublinear as possible.

7.3 Remaining challenges

One-way privacy techniques in general are still a long way away from public use, due primarily to the inefficiency of known one-way privacy techniques. However, as media recognition becomes a mainstream feature and privacy concerns grow, this increasingly relevant area of study is likely to grow in demand [14], [51]. Two major remaining issues include 1) designing good cryptographic techniques for private signal processing and 2) building good algorithms to utilize them. On the cryptographic front, there are a few research areas that would facilitate the commercialization of one-way private queries. One of the most appealing is the development of fully homomorphic encryption schemes. Fully homomorphic schemes would permit computation of encrypted domain functions involving multiplications *and* additions, rather than just one operation [52]. This option is currently far from viable due to efficiency limitations, but it is nonetheless an active field. There is also ongoing work on improving classical private queries through techniques like preprocessing [19] and distance-preserving encryption [53]. In a completely different vein, recent research on quan-

tum private queries has suggested as much as an exponential reduction in computational and communication efficiency with respect to existing classical implementations [54]. Quantum private queries are still in the very early stages of development, but there have been some physical experiments in the optical domain with positive results [55].

On the other hand, efficient privacy primitives are only useful in the media-recognition domain if there exist efficient signal processing techniques that can utilize the primitives. Given current technical limitations, this translates into algorithms that can be made to rely on private queries. For instance, several existing private face-matching schemes rely on nearest-neighbor feature vector searches. As shown in this thesis, this approach can be recast for the one-way private domain by hashing feature vectors via projections onto random hyperplanes and then searching for exact partial matches within the database. That being said, many of the most successful media-matching algorithms are computationally complex and require tools that are too heavy for the encrypted domain, like ℓ_1-optimization. If the engineering community wishes to pursue private media search as a viable technology–and there is strong incentive to do so–then there must exist sufficiently accurate classification algorithms that can be easily adapted to the encrypted domain. This compatibility requirement will play a large role in determining what kinds of applications can actually be supported in a private setting.

Aside from the performance of our particular approach, the most important message of this work is that while privacy tools can be difficult to mesh with existing search algorithms, a signal processing approach to the problem can simplify matters significantly. In this case, our modification of the original Fisherfaces algorithm with random projection hashing made the problem much more tractable from a privacy standpoint, allowing the search to be framed as a set of PIR queries. It is uncommon in the literature to find schemes that modify the signal processing portion of a privacy-preserving media search in order to facilitate adaptation to the private domain. However, the presented results suggest that this is an approach worth considering, particularly given the inefficiency of existing techniques.

References

[1] N. Singer, "The human voice, as game changer," New York Times http://www.nytimes.com/2012/04/01/technology/nuance-communications-wants-a-world-of-voice-recognition.html?pagewanted=all, march 2012.

[2] T. Singer, "New police scanner raises facial profiling concerns," NPR http://www.npr.org/2011/08/11/138769662/new-police-scanner-raises-facial-profiling-concerns, August 2011.

[3] M. Barbaro and T. Zeller, "A face is exposed for user 4417749," *New York Times Technology*, August 2006, retrieved from http://www.nytimes.com/2006/08/09/technology/09aol.html?pagewanted=all.

[4] A. Acquisti and R. Gross, "Predicting social security numbers from public data," *preprint.* [Online]. Available: http://www.pnas.org/content/106/27/10975.short

[5] G. Friedland, "Privacy concerns in multimedia and their solutions," in *Tutorial at the 20th ACM International Conference on Multimedia (slides)*, 2012.

[6] J. Valentino and J. Singer-Vine, "They know what you're shopping for," *The Wall Street Journal*, December 2012, retrieved from http://online.wsj.com/article/SB10001424127887324784404578143144132736214.html.

[7] N. Perlroth, "Yahoo breach extends beyond yahoo to gmail, hotmail, aol users," *New York Times Technology*, July 2012, retrieved from http://bits.blogs.nytimes.com/2012/07/12/yahoo-breach-extends-beyond-yahoo-to-gmail-hotmail-aol-users/.

[8] M. Barni, T. Bianchi, D. Catalano, M. D. Raimondo, R. Labati, P. Failla, D. Fiore, R. Lazzeretti, V. Piuri, A. Piva, and F. Scotti, "A privacy-compliant fingerprint recognition system based on homomorphic encryption and fingercode templates," in *Proc. IEEE Intl. Conf. on Biometrics: Theory Applications and Systems*, 2010, pp. 1–7.

[9] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft, "Privacy-preserving face recognition," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, I. Goldberg and M. Atallah, Eds., 2009, vol. 5672, pp. 235–253.

[10] A. Sadeghi, T. Schneider, and I. Wehrenberg, "Efficient privacy-preserving face recognition," in *Information, Security and Cryptology ICISC 2009*, ser. Lecture Notes in Computer Science, D. Lee and S. Hong, Eds., 2010, vol. 5984, pp. 229–244.

[11] P. Smaragdis and M. Shashanka, "A framework for secure speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 4, pp. 1404–1413, 2007.

[12] O. Bowcott, "Interpol wants facial recognition database to catch suspects," The Guardian, October 2008.

[13] G. Friedland and R. Sommer, "Cybercasing the joint: Privacy implications of geo-tagging," in *Proc. USENIX Workshop on Hot Topics in Security*, 2010.

[14] S. Sengupta and K. O'Brien, "Facebook can ID faces, but using them grows tricky," *New York Times*, September 21 2012, retrieved from http://www.nytimes.com/2012/09/22/technology/facebook-backs-down-on-face-recognition-in-europe.html.

[15] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *IEEE Symposium on Foundations of Computer Science*, 1995, pp. 41–50.

[16] E. Kushilevitz and R. Ostrovsky, "Replication is not needed: single database, computationally-private information retrieval," in *Annual Symposium on IEEE Symposium on Foundations of Computer Science*, 2012, pp. 364–373.

[17] A. Beimel, Y. Ishai, E. Kushilevitz, and T. Malkin, "One-way functions are essential for single-server private information retrieval," in *Thirty first annual ACM symposium on theory of computing*, 1999, pp. 89–98.

[18] R. Sion and B. Carbunar, "On the computational practicality of private information retrieval," in *Proc. Network and Distributed System Security Symposium*, 2007.

[19] A. Beimel, Y. Ishai, and T. Malkin, "Reducing the servers' computation in private information retrieval: PIR with preprocessing," *J. Cryptology*, vol. 17, no. 2, pp. 125–151, 2004.

[20] D. Woodruff and S. Yekhanin, "A geometric approach to information theoretic private information retrieval," in *Proc. IEEE Conf. on Computational Complexity*, 2005, pp. 275–284.

[21] F. Olumofin, "Practical private information retrieval," Ph.D. dissertation, University of Waterloo, August 2011.

[22] R. Ostrovsky, W. Skeith, and O. Patashnik, "Private searching on streaming data," *Journal of Cryptology*, vol. 20, pp. 397–430, 2007.

[23] J. Bethencourt, D. Song, and B. Waters, "New constructions and practical applications for private stream searching," in *Security and Privacy, 2006 IEEE Symposium on*, may 2006, pp. 134 –139.

[24] M. Finiasz and K. Ramchandran, "Private stream search at the same communication cost as a regular search: Role of ldpc codes," in *Proc. IEEE Int. Symposium on Information Theory*, 2012, pp. 2556–2560.

[25] Y. Huang, L. Malka, D. Evans, and J. Katz, "Efficient privacy-preserving biometric identification," in *Proc. Network and Distributed System Security Symposium*, 2011.

[26] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of Cognitive Neuroscience*, vol. 3, no. 1, p. 7186, 1991.

[27] J. Haitsma and T. Kalker, "A highly robust audio fingerprinting system," in *Proc. the Intl. Symposium on Music Information Retrieval*, 2002, pp. 107–115.

[28] J. Shashank, P. Kowshik, K. Srinathan, and C. Jawahar, "Private content based image retrieval," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[29] M. Barni, T. Bianchi, D. Catalano, M. di Raimondo, R. D. Labati, and P. Failla, "Privacy preserving fingercode authentication," in *Proc. ACM Workshop on Multimedia and Security*, 2010.

[30] S. Avidan and M. Butman, "Blind vision," in *LNCS*, vol. 3953, no. 2006, 2006, pp. 1–13.

[31] B. Moskovich and M. Osadchy, "Illumination invariant representation for privacy-preserving face identification," in *Computer Vision and Pattern Recognition Workshops*, 2010, pp. 154–161.

[32] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich, "SCiFI - a system for secure face identification," in *IEEE Symposium on Security and Privacy*, 2010, pp. 239–254.

[33] M. Pathak and B. Raj, "Privacy-preserving speaker verification as password matching," in *Proc. Intl. Conf. on Acoustics, Speech, and Signal Processing*, 2012.

[34] M. Barni, P. Failla, R. Lazzeretti, A. Sadeghi, and T. Schneider, "Privacy-preserving ECG classification with branching programs and neural networks," *IEEE Trans. on Information Forensics and Security*, vol. 6, no. 2, pp. 452–468, 2011.

[35] G. Ghinita, "Private queries and trajectory anonymization: a dual perspective on location privacy," *Trans. on Data Privacy*, vol. 2, no. 1, 2009.

[36] P. Lin and S. Candan, "Secure and privacy-preserving outsourcing of tree structured data," vol. 3178, no. 2004, pp. 155–176, 2004.

[37] F. Olumofin and I. Goldberg, "Revisiting the computational practicality of private information retrieval," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, G. Danezis, Ed., 2012, vol. 7035, pp. 158–172.

[38] P. Maass and M. Rajagopalan, "That's no phone. that's my tracker." *New York Times*, July 13 2012, retrieved from http://www.nytimes.com/2012/07/15/sunday-review/thats-not-my-phone-its-my-tracker.html.

[39] J. Pepitone, "Facebook settles FTC charges over 2009 privacy breaches," *CNN, Online Edition*, November 10, 2011, retrieved from http://money.cnn.com/2011/11/29/technology/facebook_settlement/index.htm?iid=EL.

[40] R. Ostrovsky and W. Skeith, "A survey of single-database private information retrieval: Techniques and applications," in *Public Key Cryptography PKC 2007*, ser. Lecture Notes in Computer Science, T. Okamoto and X. Wang, Eds., 2007, vol. 4450, pp. 393–411.

[41] R. Rivest, L. Adleman, and M. Dertouzos, "On data banks and privacy homomorphisms," in *Foundations of Secure Computation*, R. DeMillo, D. Dobkin, A. Jones, and R. Lipton, Eds. Academic Press, 1978.

[42] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology EUROCRYPT 99*, ser. Lecture Notes in Computer Science, S. Jacques, Ed., 1999, vol. 1592, pp. 223–238.

[43] A. Aude, "Audio quality measurement primer," http://www.intersil.com/content/dam/Intersil/documents/an97/an9789.pdf, 1998.

[44] C. Yeo, P. Ahammad, H. Zhang, and K. Ramchandran, "Rate-efficient visual correspondences using random projections," in *Proc. IEEE Intl. Conf. on Image Processing*, 2008, pp. 217–220.

[45] K. Min, L. Yang, J. Wright, L. Wu, X. Hua, and Y. Ma, "Compact projection: Simple and efficient near neighbor search with practical memory requirements," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2010, pp. 3477–3484.

[46] P. Belhumeur, J. Hespanha, , and D. Kriegman, "Eigenfaces vs. fisherfaces: recognition using class specific linear projection," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, pp. 711–720, 1997.

[47] A. L. Cambridge, "The database of faces," Retrieved from http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html, 2002.

[48] M. Rabin, "How to exchange secrets with oblivious transfer," *Technical report TR-81, Aiken Computation Lab, Harvard University*, 1981.

[49] S. Papadopoulos, S. Bakiras, and D. Papadias, "pCloud: A distributed system for practical PIR," *Dependable and Secure Computing, IEEE Transactions on*, vol. 9, no. 1, pp. 115 –127, jan.-feb. 2012.

[50] W. Muller, A. Heinrich, and M. Eisenhardt, "Privacy of ideas in P2P networks," Tech. Rep.

[51] S. Sengupta, "Parenting dilemmas in the age of facial recognition," *New York Times*, July 2012, retrieved from http://bits.blogs.nytimes.com/2012/07/23/parenting-dilemmas-in-the-age-of-facial-recognition/.

[52] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009.

[53] W. Lu, A. Varna, A. Swaminathan, and M. Wu, "Secure image retrieval through feature protection," in *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, 2009, pp. 1533–1536.

[54] V. Giovannetti, S. Lloyd, and L. Maccone, "Quantum private queries: Security analysis," *Information Theory, IEEE Transactions on*, vol. 56, no. 7, pp. 3465 –3477, July 2010.

[55] F. DeMartini, V. Giovanetti, S. Lloyd, E. Nagali, L. Sansoni, and F. Sciarrino, "Experimental quantum private queries with linear optics," *Phys. Rev. A*, vol. 80, no. 1, 2009.

Appendix A

Proofs for Claims

A.1 Proof of Observation 1

Proof. Let Q^v denote the PIR query vector to the vth server $\mathcal{S}_v$. A sufficient condition for information-theoretic security is that the mutual information between the client's queries (Q^1 and Q^2) to the two servers and the client's desired file index (i) given the database contents is zero, i.e.

$$I(Q^v, i|\mathcal{DB}) = H(Q^v|\mathcal{DB}) - H(Q^v|i, \mathcal{DB}) = 0 \qquad \text{for } v \in 1, 2$$

where $H(\cdot)$ denotes the entropy function. Recall that the query vectors are generated as $Q^0 = (e^i \cdot r) \oplus a$ and $Q^1 = (e^i \cdot (1-r)) \oplus a$, where a is a Bernoulli random process of length n with parameter $\frac{1}{2}$ and $r \sim \text{Bern}(\frac{1}{2})$ is a random flag indicating which server gets $e^i \oplus a$ and which gets a. e^i is the ith canonical vector, with a 1 at the ith index and zeros elsewhere. Thus it suffices to show that $I(a \oplus e^i, i|\mathcal{DB}) = 0$ and $I(a, i|\mathcal{DB}) = 0$. For the latter we have

$$\begin{aligned} I(a, i|\mathcal{DB}) &= H(a|\mathcal{DB}) - H(a|i, \mathcal{DB}) \\ &= H(a) - H(a) = 0 \end{aligned}$$

because the entries of a are generated independently of i. The same is true for $I(a \oplus e^i, i|\mathcal{DB}) = 0$, since the bitwise sum of a $\text{Bern}(\frac{1}{2})$ random process with any other binary random process is an iid $\text{Bern}(\frac{1}{2})$ random process. So $I(a \oplus e^i, i|\mathcal{DB}) = 0$. Thus $I(Q^0, i|\mathcal{DB}) = I(Q^1, i|\mathcal{DB}) = 0$; without collusion, $\mathcal{S}_v$ cannot possibly learn any information about the desired query index i from Q^v. Since this is the only information received by the server, the scheme is information-theoretically secure. □

Note that the server may have side information, as mentioned in the proof—the point is that the client's message to the server does not leak any *additional* information.

A.2 Proof of Observation 2

Proof. Any semi-honest server $\mathcal{S}_v$ in the system is able to view each input query vector Q^v from the client, as well the full database. We wish to prove that given this information, each server $\mathcal{S}_v$ learns no information from $\mathcal{C}$. Let $Q^{v,w}$ denote $\mathcal{C}$'s PIR query vector to $\mathcal{S}_v$ for the wth noisy subfingerprint q_w, and let $\{Q^{v,w}\}_{0 \leq w \leq 255}$ denote the set of all PIR queries sent to $\mathcal{S}_v$ during AudioSearchPIR:FullSearch; we will abbreviate this $\{Q^{v,w}\}$. To show that $\mathcal{S}_v$ gains no information about q, we must show that $I(q, \{Q^{v,w}\}|\mathcal{DB}) = 0$.

Starting with $w = 0$ and using scheme $\mathcal{P}$, $\mathcal{C}$ sends query $Q^{v,0}$ to $\mathcal{S}_v$. By Observation 1, $\mathcal{S}_v$ learns nothing about the nature of q_w. More precisely, $I(Q^{v,0}, q_w|\mathcal{DB}) = 0$. Since $Q^{v,0}$ was generated independently of all the other subfingerprints in q, we have by extension that $I(Q^{v,1}, q|\mathcal{DB}) = 0$. The same is true for all the remaining $w > 0$. Since $Q^{v,i}$ was generated independently of $Q^{v,j}$ for all $i \neq j$, we have that

$$H\big(\{Q^{v,w}\}|\mathcal{DB}\big) = H(Q^{v,0}|\mathcal{DB}) + \ldots + H(Q^{v,255}|\mathcal{DB}).$$

This in turn implies

$$\begin{aligned}
I(q, \{Q^{v,w}\}|\mathcal{DB}) &= H\Big(\{Q^{v,w}\}|\mathcal{DB}\Big) - H\Big(\{Q^{v,w}\}|q, \mathcal{DB}\Big) \\
&= H(Q^{v,0}|\mathcal{DB}) + \ldots + H(Q^{v,255}|\mathcal{DB}) - H(Q^{v,0}|q, \mathcal{DB}) - \ldots - H(Q^{v,255}|q, \mathcal{DB}) \\
&= H(Q^{v,0}|\mathcal{DB}) - H(Q^{v,0}|q, \mathcal{DB}) + \ldots + H(Q^{v,255}|\mathcal{DB}) - H(Q^{v,255}|q, \mathcal{DB}) \\
&= I(q, Q^{v,0}|\mathcal{DB}) + \ldots + I(q, Q^{v,255}|\mathcal{DB}) \\
&= 0,
\end{aligned}$$

since we already showed that $I(q, Q^{v,w}) = 0$ for all $0 \leq w \leq 255$. Thus AudioSearch-PIR:FullSearch leaks no information to the server about the client's query. □

Appendix B

Additional Algorithm Details

Algorithm B.1 AudioSearchPIR:ThresholdSearch

```
1: function ExtractFeatures(NoisyAudio)
2:     Input: Noisy audio clip, at least 3 seconds long (NoisyAudio)
3:     Output: Vector of 256 query subfingerprints (q)
4: end function
5: function Pir(Q, S, P)
6:     Input: Server ID (S), PIR protocol (P), PIR query (Q)
7:     Output: Mixed database contents (ScrambledResult)
8: end function
9: function ComputeBitErrorRate(q, r)
10:    Input: query fingerprint block (q), scrambled fingerprint block output of PIR (r)
11:    Output: Bit error rate between the two vectors (BitErrorRate)
12: end function
13:
14: Input: Noisy audio clip (NoisyAudio)
15: Output: ID, content of closest match in Hamming distance (ClosestMatch)
16:
17: MinBitErrorRate ← 1
18: q ← ExtractFeatures(NoisyAudio)
19: for w = 0 → 255 do
20:     ResultList ← 0
21:     for all non-colluding server S_v do
22:         C generate PIR query Q_v for subfingerprint q_w, to be sent to server S_v
23:         ScrambledResult ← Pir(Q_v, S_v, P)
24:         ResultList ← ResultList ⊕ ScrambledResult
25:     end for
26:     for all Result in ResultList do
```

Algorithm B.2 AudioSearchPIR:ThresholdSearch, Continued

```
27:         BitErrorRate ← ComputeBitErrorRate(q, Result)
28:         if BitErrorRate < ϵ then
29:             MinBitErrorRate ← BitErrorRate
30:             ClosestMatch ← Result
31:             break
32:         end if
33:     end for
34:     if MinBitErrorRate < 1 then
35:         break
36:     end if
37: end for
```

Algorithm B.3 AudioSearchPIR:LookupTable

```
1: function PIR(Q, S, P, LocationFlag)
2:     Input: Server ID (S), PIR protocol (P), PIR query (Q), Flag indicating whether the PIR query is for a subfingerprint location list (1) or a fingerprint block (0) (LocationFlag)
3:     Output: Scrambled database contents of either the lookup table entries or fingerprint blocks (ScrambledResult)
4: end function
5:
6: Input: Noisy audio clip (NoisyAudio)
7: Output: ID, content of closest match in Hamming distance (ClosestMatch)
8:
9: MinBitErrorRate ← 1
10: q ← ExtractFeatures(NoisyAudio)
11: for w = 0 → 255 do
12:     Result ← 0
13:     for all non-colluding server S_v do
14:         C generate PIR query Q_v for subfingerprint q_w, to be sent to server S_v
15:         ScrambledLocationResult ← PIR(Q_v, S_v, P)
16:         LocationResult ← LocationResult ⊕ ScrambledLocationResult
17:     end for
18:     for all Location of q_w in LocationResult do
19:         for all non-colluding server S_v do
20:             C generate PIR query Q_v for Location, to be sent to server S_v
21:             ScrambledResult ← PIR(Q_v, S_v, P)
22:             Result ← Result ⊕ ScrambledResult
23:         end for
24:         BitErrorRate ← ComputeBitErrorRate(q, Result)
25:         if BitErrorRate < Threshold then
26:             MinBitErrorRate ← BitErrorRate
27:             ClosestMatch ← Result
28:             break
29:         end if
30:     end for
31:     if MinBitErrorRate < 1 then
32:         break
33:     end if
34: end for
```

Algorithm B.4 AudioSearchPSS

```
1: function ExtractFeatures(NoisyAudio)
2:     Input: Noisy audio clip, at least 3 seconds long (NoisyAudio)
3:     Output: Vector of 256 query subfingerprints (q)
4: end function
5: function Pss(Q, S)
6:     Input: PSS query (Q), Single server (S)
7:     Output: Buffer with mixed results (Buffer)
8: end function
9: function DecryptBufferBin(BufferPosition,PrivateKey)
10:    Input: Index of buffer to decrypt (BufferPosition), Private encryption key (PrivateKey)
11:    Output: Decypted buffer bin contents (Result)
12: end function
13: function IsSingleton(Result)
14:    Input: Decrypted buffer bin contents (Result)
15:    Output: Indicator reflecting whether Result was a singleton (1) or a superposition of several fingerprint blocks (0)
16: end function
17: function ComputeBitErrorRate(q, r)
18:    Input: query fingerprint block (q), scrambled fingerprint block output of PSS (r)
19:    Output: Bit error rate between the two vectors (BitErrorRate)
20: end function
21: function RemoveSingleton(Result, &Buffer)
22:    Input: Plaintext fingerprint block (Result), Result buffer (Buffer)
23:    All instances of Result are removed from Buffer
24: end function
25:
26: Input: Noisy audio clip (NoisyAudio)
27: Output: ID, content of closest match in Hamming distance (ClosestMatch)
28:
29: MinBitErrorRate ← 1
30: q ← ExtractFeatures(NoisyAudio)
31: for w = 0 → 255 do
32:    C generate PSS query Q for subfingerprint q_w
33:    Buffer ← Pss(Q, S)
34:    BufferPosition ← 0
35:    while BufferPosition ≤ length(Buffer) do
36:        Result ← DecryptBufferBin(BufferPosition,PrivateKey)
37:        if not IsSingleton(Result) then
38:            BufferPosition ← BufferPosition + 1
39:            continue
40:        end if
```

Algorithm B.5 AudioSearchPSS, Continued

```
41:         BitErrorRate ← ComputeBitErrorRate(q, Result)
42:         if BitErrorRate < Threshold then
43:             MinBitErrorRate ← BitErrorRate
44:             ClosestMatch ← Result
45:             break
46:         end if
47:         RemoveSingleton(Result, Buffer)
48:         BufferPosition ← 0
49:     end while
50:     if MinBitErrorRate < 1 then
51:         break
52:     end if
53: end for
```