Private Information Retrieval with Access Control

by

Pawan Goyal

S.B., Computer Science and Engineering and Mathematical Economics,
Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Authored by:	Pawan Goyal Department of Electrical Engineering and Computer Science May 19, 2023
Certified by:	Sacha Servan-Schreiber Ph.D. Candidate, MIT CSAIL Thesis Supervisor
Certified by:	Srini Devadas Edwin Sibley Webster Professor of Electrical Engineering and Computer Science Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Thesis Committee

Private Information Retrieval with Access Control

by

Pawan Goyal

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Private Information Retrieval (PIR) allows a user to query for a record from a remote database without revealing the query to the database server. However, PIR does not provide access control guarantees, allowing any user access to any record. Moreover, the database server cannot check access permissions through conventional techniques as they are fundamentally incompatible with PIR.

In this thesis, we present Pirac—a novel framework for access control in PIR. In Pirac, only users who have permission to access a specific database record can retrieve it. Our constructions make black-box use of the underlying PIR schemes and therefore apply to both single-server and multi-server PIR.

We evaluate our open-source implementation of Pirac when applied to state-of-the-art PIR schemes. For databases with roughly one million 4 KiB records, adding access control via Pirac incurs a 2.6× server-side computational overhead in single-server PIR and 3.1× in multi-server PIR, while keeping user processing and communication overheads at a minimum.

We show that Pirac enables new applications of PIR, including privacy-preserving password breach lookups, multi-user databases with personal content, and private friend discovery, among others.

Thesis Supervisor: Sacha Servan-Schreiber
Title: Ph.D. Candidate, MIT CSAIL

Thesis Supervisor: Srini Devadas
Title: Edwin Sibley Webster Professor of Electrical Engineering
and Computer Science

Acknowledgments

I would first and foremost like to express my utmost gratitude to my direct thesis advisor, Sacha Servan-Schreiber, without whose support this thesis would not have been possible. He brought me up to speed with the latest advancements in cryptography and showed incredible patience in discussing ideas. At times, I would have good ideas, but he would ask the right questions to make them great. His passion for cryptography served as a constant source of inspiration, and he always went the extra mile to help me learn, handle failures, and rise again. Sacha, thanks for showing me the art of doing research. I learned a lot from working with you.

I would also like to extend my sincere thanks to Srini Devadas, my faculty supervisor, whose guidance and expertise have been invaluable during my pursuit of this research. I am deeply grateful for his feedback and advice.

To Kyle Hogan, Mayuri Sridhar, and Simon Langowski, I want to express my deep appreciation for offering valuable feedback and suggestions, in addition to creating a fun and stimulating atmosphere during the times I spent in their office.

To my friends: Arindam, thanks for being a friend I could always confide in and making me Cambridge's best cocktails every weekend. Giannis, thanks for playing ping pong with me and teaching me how to live an organized life. Tharindu, thanks for countless late-night philosophical and not-so-philosophical talks and helping me become a better version of myself. Yash, thanks for giving me a new perspective in life and teaching me how to live life to the fullest. Era, thanks for organizing the mafia nights and going on walks with me. To my brothers at my fraternity and my friends in Norfolk, thanks for keeping me sane during the pandemic and ensuring that each day is a new adventure.

Last but not least, I want to acknowledge my family and especially my sister Shristi for their relentless support. They are my rock, constantly motivating me through my life's highs and lows.

I extend my gratitude to all those mentioned above, as well as anyone else who has supported me along the way during my time here at MIT. Thank you all.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

User privacy is becoming an increasingly important consideration on the internet and new cryptographic protocols are rapidly being developed to meet these privacy demands. At the backbone of many such protocols is Private Information Retrieval (PIR) [30, 63]. Through PIR, a user can retrieve a record from a remote database without revealing the query to the database server.

PIR has seen a renewed interest thanks to several concrete applications, including private messaging [10, 11, 80], content discovery [15, 83, 88, 89], private friend discovery [39, 61], privacy-preserving advertising [12, 52, 60, 81, 91], blocklist lookups [62, 86], and media consumption [56].

Systems that use PIR as a building block cater to a large number of users. In the real world, different users might have different access rights to database records. For example, subscription services require memberships to access pay-walled content [56] and friend discovery services should only reveal account information to contacts to avoid publicly leaking email addresses or phone numbers [61, 73].

These examples motivate the development of access control in PIR and highlight the current gap between real-world databases which support access control (but have no query privacy) and databases that support private queries through PIR (but have no access control). Specifically, conventional access control is easy: it suffices for the server to check if the user is authorized to retrieve the requested record. However, enforcing access control in PIR, where the query must remain private from the server,

becomes a challenging problem [56, 57, 65, 66]. Conventional approaches to access control rely on the query, which puts them in fundamental conflict with the query privacy property of PIR [56].

With this in mind, we formally examine how access control can be applied to PIR, achieving a similar functionality to that of traditional databases.

Our contribution. This thesis contributes Pirac: a framework for adding access control to PIR. We realize Pirac via two different classes of constructions.

Construction I. Our first construction is realized using a PIR-by-keywords scheme. This approach requires no additional cryptographic assumptions but does require the PIR-by-keywords scheme to satisfy "keyword privacy," a property that we define in Section 6.1.1.

Construction II. Our second construction is realized using lightweight cryptography (symmetric-key encryption and pseudorandom functions) and is more general in that it makes no assumptions whatsoever on the underlying PIR scheme. At a high level, our construction encrypts the database such that only the authorized access key holder can decrypt the retrieved record. The user queries the database as before, but only authorized users can decrypt the result. However, we show that additional care is required to handle dynamic databases and changing access policies, concepts we elaborate on in Section 1.1.

The second construction is more concretely efficient and more widely applicable, making it the focus of this thesis and our evaluation. We show in Chapter 8, that Pirac imposes minimal computational overheads on the server and has *no* communication overheads. These features make it possible to deploy Pirac in exciting new contexts and applications, which we highlight in Chapter 9.

In summary, this thesis contributes:

1. Pirac: a framework for introducing access control to PIR with immediate real-world applications,
2. a construction using keyword-private PIR-by-keywords that makes no additional cryptographic assumptions,

3. a construction based on lightweight cryptographic primitives that applies to all PIR schemes, and
4. an open-source implementation of Pirac, which we extensively evaluate using both single-server and multi-server PIR schemes.

1.1 Setting and Threat Model

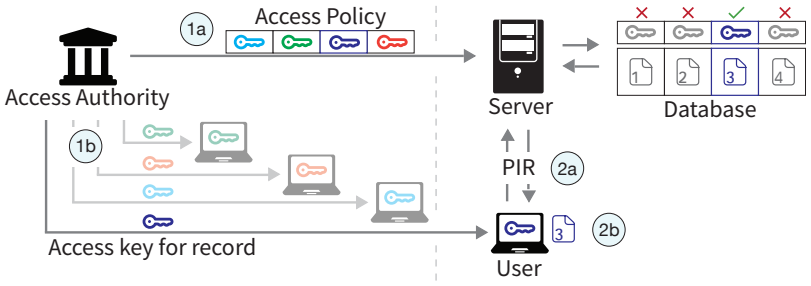

Figure 1-1: (1a) An access authority gives an access policy to the database server and (1b) distributes the access keys to users. (2a) a user queries the database with PIR and (2b) recovers the record, provided they have the correct access key.

Pirac is instantiated with one (or more) database server(s) and a set of users. For simplicity, however, we focus on the single-server PIR setting. The Pirac setting and parties involved are illustrated in Figure 1-1. The database server processes user queries and enforces access control policies over the queries. Similarly to prior work (e.g., [48, 65, 66]), we assume that the access policy is created and provided to the database server by an authority that also oversees granting access rights to users. For example, this authority can be the database owner or an external party like a bank or identity provider that outsources the PIR computation; in other cases, the access authority can be the database server itself (see Chapter 9).

The threat model and assumptions of Pirac reflect those of PIR, which requires query privacy for the user when interacting with a malicious database server (see Chapter 4 for a formal definition of PIR). However, in Pirac, we must additionally consider access control and record privacy for the database. Specifically, we must model a setting where *users* may try to learn information about records that they do not have permission to access.

Threat model. The goal of our threat model is to capture the different settings and assumptions under which the server must enforce access control. Each threat model "tier" reduces the set of restrictions surrounding the database and access permissions (i.e., static or dynamic database records and permissions). These are summarized below and formalized in Chapter 5.

1. *Static-database authorization* applies only to static databases and static access policies. Assumes: records do not change over time, permissions for a record do not change, and a secret server state.
2. *Dynamic-database authorization* applies to *dynamic* databases, where records may change over time, and ensures that no information is revealed about a record to a user unless they have permission to access the record. Assumes: a static access policy and a secret server state.
3. *Forward-secret authorization* enhances dynamic-database authorization and applies to dynamic databases, where both records and access permissions change over time. This tier ensures that users that gain access to a record learn no information on previous versions of the record in the database *prior* to them gaining access [41, 54]. Assumes: only a secret server state.

Breach resilience. For all three tiers, we can additionally require breach resilience [9, 85], which guarantees that even if the entire state of the database server is compromised and revealed to users, access control is still guaranteed. This captures a threat model where a "snapshot" of the database and server state might be leaked to an adversary at various points in time and removes the assumption of a secret server state under all three authorization tiers.

Chapter 2

Overview

In this section, we describe a "strawman" approach to realizing access control in PIR and cover some limitations associated with it. We then overview our approaches to realizing Pirac for dynamic databases in Section 2.2.

2.1 The Strawman Approach

A natural idea for realizing access control in PIR is using encryption: simply encrypt each record using its access key! We describe this folklore strawman approach in Construction 1.

Limitations of the strawman approach. The strawman approach is only secure under the *static-database authorization* model described in Section 1.1. We explain below why this construction fails to meet the required authorization properties when considering dynamic databases and access policies.

Not dynamic-database compatible. The first limitation of Construction 1 is that it may reveal when a record is updated. Consider malicious users that repeatedly query the database for a record that they do not have permission to access. Depending on the returned (encrypted) record, the user can determine if the record was updated or not by inspecting the ciphertext: the ciphertext will be different if the server replaced the i-th record with the updated (encrypted) record. This is metadata leakage (ob-

Strawman Pirac with static-database authorization

To set up an access policy as the access authority:

Step 1. Generate n encryption keys $\Lambda := (\kappa_1, \ldots, \kappa_n)$.
▷ Let κ_i be the access key for the i-th record in the database.

Step 2. Give access key κ_i to users who are authorized to access the i-th record in the database and send Λ to the database server.

To set up access control for the database as the server:

▷ Let database $\mathcal{DB} := (\mathsf{x}_1, \ldots, \mathsf{x}_n)$, where x_i is the i-th record.

Step 1. Encrypt record x_i for $i \in \{1, \ldots, n\}$ with access key κ_i, using symmetric-key encryption, to obtain ciphertext $\tilde{\mathsf{x}}_i$.

Step 2. Replace the i-th record in the database with $\tilde{\mathsf{x}}_i$.

To retrieve record x_i as an authorized user:

Step 1. Retrieve the encrypted record $\tilde{\mathsf{x}}_i$ from the server via PIR.

Step 2. Decrypt the retrieved record using the access key κ_i and recover the (plaintext) database record x_i.

Construction 1: Static-database Pirac from encryption.

servers learn information about the database even without seeing the *contents* of the database) that can have important downstream ramifications in many systems.

Not forward secret. The strawman scheme does not provide *forward secrecy*. Suppose a user does *not* have access to the i-th record at time t_0. Then, at a future time t_1, the user is given access to the i-th record following an update to the database. If the user queried and stored the (encrypted) record at time t_0, then they can recover the old record after gaining access to the updated record at time t_1.

Not breach resilient. The strawman scheme does not provide security against a snapshot adversary that obtains a copy of the server state at different points in time [9]. Specifically, even though the data is encrypted, the server can have the encryption keys in the clear (e.g., for the purpose of encrypting record updates). As such, it is possible to decrypt all the records with a copy of the server state.

2.2 Overview of Pirac Constructions

We provide two classes of constructions for Pirac.

Pirac via PIR-by-keywords

Our first construction is based on keyword-private PIR-by-keywords, a notion that we define in Section 6.1.1 and achieved by several existing PIR-by-keyword schemes [16, 71] (also see Appendix A). Our Pirac construction is essentially a reduction from keyword-private PIR-by-keywords to Pirac, which we describe in Section 6.1.2, and achieves *forward-secret authorization* by default.

Pirac via encryption

Our second construction is realized by upgrading the strawman Pirac (Construction 1) to handle authorization with dynamic databases and (optionally) forward secrecy. At a high level, we show that if the server refreshes the key with the help of a PRF and re-encrypts each record between queries, we can achieve (1) metadata privacy between updates and (2) forward secrecy as access rights change (see Section 6.2).

Upgrade to breach resilience. In Section 6.2.3, we show how to upgrade our encryption-based construction to achieve breach resilience. We do so through public-key encryption (as opposed to using symmetric-key encryption). Unfortunately, public-key encryption incurs a large concrete overhead making breach resilience primarily of theoretical interest on larger databases (see Chapter 8 for evaluation).

2.3 Efficiency Considerations

We highlight some efficiency properties that we consider when defining and constructing Pirac.

- *Server-side overhead.* One important property is the server-side overhead: Pirac should not impose significant server-side work, ideally remaining on-par with the work required to process the baseline PIR query.

- *User-side overhead.* Similarly, Pirac should not impose significant user-side computational overheads and must remain on-par with the work required to generate the baseline PIR query.
- *Communication overhead.* Finally, we must ensure that the communication between the user and the server in Pirac does not significantly increase, relative to the baseline PIR scheme.

When defining Pirac, we will allow for a polynomial factor (in the security parameter) overhead in all three of these properties. However, we note that our constructions achieve minimal overheads that are at worst a linear factor in the security parameter.

Chapter 3

Related Work

In this section, we survey related work on access control and database privacy in PIR.

Symmetric PIR [47] (SPIR) is a solution to the database privacy problem (users only gain one database record per query). Specifically, SPIR *rate-limits* users to only one record per query, but does not provide *access control*, since any user is still allowed to retrieve any record from the database.

Access control in single-server PIR. Layouni [65] is the first to consider access control for the SPIR protocol of Lipmaa [69]. Layouni et al. [66] improve the construction by making it black-box with respect to the underlying SPIR scheme and considering a setting with multiple access authorities. Neither work provides an implementation and their constructions are mainly of theoretical interest given the heavy cryptographic primitives involved, including the use of a linear number of bilinear pairing operations.

Access control in multi-server PIR. Henry et al. [57] build access control into the Percy++ PIR scheme [48]. Their construction results in large computational overheads for the servers *and* the user (due to zero-knowledge proofs, polynomial commitments, and bilinear pairings). Additionally, the communication and computation for the user are linear in the number of records.

For PIR based on function secret sharing (FSS) [16], Servan-Schreiber et al. [82] develop access control for FSS, which they show can be used to instantiate efficient

access control in two-server PIR (multi-server FSS schemes are not as efficient [16, 17, 20, 33]).

We compare both of these approaches to Pirac in Chapter 8 and note that neither of these approaches to access control generalize to other multi-server PIR schemes.

Access control for Oblivious Transfer. A separate line of work explores access control for Oblivious Transfer (OT) [6, 8, 21–24, 35, 40, 67, 90], which is related to PIR. Like PIR, OT [79] is a protocol designed to privately retrieve a bit from a sender (e.g., a database). However, unlike PIR, OT does not achieve the low communication overhead required of PIR and is overall a weaker primitive compared to (symmetric) PIR.

Among current OT schemes, Camenisch et al. [21] utilize anonymous credentials and zero-knowledge proofs to enforce access control in Oblivious Transfer. In their approach, the access policy is public, and users must convince the database that they possess the credentials for all the required categories to access a specific entry. The work of Camenisch et al. [24] extend this line of research by introducing hidden access policies. However, to the best of our knowledge, none of the aforementioned works provide an implementation of their schemes and are primarily of theoretical interest. We consider these approaches as orthogonal but note that our Pirac constructions can be applied to OT as well.

Chapter 4

Preliminaries

In this section, we describe the notation we use throughout the thesis and provide background on the cryptographic protocols used in the construction of Pirac.

4.1 Notation

We describe the database as a vector $\mathcal{DB} := (\mathsf{x}_1, \ldots, \mathsf{x}_n)$, where each $\mathsf{x}_i \in \{0,1\}^\ell$. We index into vectors as $\mathcal{DB}[i]$ such that $\mathcal{DB}[i] = \mathsf{x}_i$, modeling the syntax of an array access. Occasionally, we denote the vector of tuples $((w_1, \mathsf{x}_1), \ldots, (w_n, \mathsf{x}_n))$ as $\mathcal{DB}_\mathsf{W}$ and index into $\mathcal{DB}_\mathsf{W}$ as $\mathcal{DB}_\mathsf{W}[w_i] = \mathsf{x}_i$, modeling the syntax of a key-value store instead of an array access. We use $\mathbb{N}$ to denote natural numbers. Assignment to a variable x from a possibly randomized algorithm Alg is denoted $x \leftarrow \mathsf{Alg}$ and assignment from a value y is denoted $x := y$. Sampling a random element x from a distribution $\mathcal{S}$ is denoted $x \leftarrow_R \mathcal{S}$. By *efficient*, we mean any (possibly non-uniform) probabilistic polynomial time (PPT) algorithm. We use the symbol $\perp$ to represent "null" and $\mathsf{poly}(\cdot)$ to denote a fixed polynomial and $\mathsf{negl}(\cdot)$ to denote a negligible function. We use $\approx_c$ to denote computational indistinguishability between two distributions.

4.2 Private Information Retrieval

PIR is instantiated between a client and a remote database server. The database $\mathcal{DB}$ consists of n ℓ-bit records $(\mathsf{x}_1, \ldots, \mathsf{x}_n)$. The client has an index i and must retrieve

$\mathcal{DB}[i]$ through the server, without revealing i to the server in the process.

Definition 4.1 (Private Information Retrieval [28, 30])**.** Fix security parameter $\lambda \in \mathbb{N}$ and database parameters $n, \ell \in \mathbb{N}$. Let $\mathcal{DB}$ be a n-record database with ℓ-bit records. PIR consists of three efficient (possibly randomized) algorithms $(\mathsf{Query}, \mathsf{Answer}, \mathsf{Recover})$, with the following syntax.

- $\mathsf{Query}(1^\lambda, i) \to \mathsf{q}$. Takes as input the security parameter λ and an index $i \in \{1, \ldots, n\}$. Outputs a query q for index i.
- $\mathsf{Answer}(\mathcal{DB}, \mathsf{q}) \to \mathsf{c}$. Takes as input a database $\mathcal{DB}$ and query q. Outputs a query answer c.
- $\mathsf{Recover}(\mathsf{c}) \to \mathsf{x}$. Takes as input the query answer c. Outputs database record x.

The above functionality must satisfy: *correctness*, *privacy*, and *efficiency*.

- **Correctness.** For all databases $\mathcal{DB}$ and for all indices $i \in \{1, \ldots, n\}$,

$$\Pr\left[\begin{array}{l} \mathsf{q} \leftarrow \mathsf{Query}(1^\lambda, i) \\ \mathsf{c} \leftarrow \mathsf{Answer}(\mathcal{DB}, \mathsf{q}) \\ \mathsf{x} \leftarrow \mathsf{Recover}(\mathsf{c}) \end{array} : \mathsf{x} = \mathcal{DB}[i]\right] = 1 - \mathsf{negl}(\lambda).$$

- **Privacy.** For all indices $i \in \{1, \ldots, n\}$, there exists an efficient simulator $\mathcal{S}$ such that $\mathsf{Query}(1^\lambda, i) \approx_c \mathcal{S}(1^\lambda)$. That is, the query reveals no information on the index i to computationally bounded adversaries (i.e., the server).
- **Efficiency.** PIR is efficient if for all databases $\mathcal{DB}$, the size of the query and response is $O(n^{\epsilon_1}\ell^{\epsilon_2})$, for any $\epsilon_1 < 1$, $\epsilon_2 \leq 1$, that are possibly dependent on n.

Remark 1 (Multi-server PIR Syntax)**.** Definition 4.1 captures the syntax of single-server PIR schemes. However, we will also use the definition and syntax to describe multi-server PIR. In this case, we treat each server as an individual instance using the above syntax (and assume a local user state that links the two instances). Our Pirac constructions will not require explicit multi-server PIR syntax.

4.2.1 PIR by Keywords

Definition 4.1 captures the syntax for querying the database at a particular *index* (e.g., accessing an element in an array). A more general definition captures the notion of querying the database on a *keyword* (e.g., accessing an element in a key-value store). Syntactically, $\mathsf{PIR.Query}$ takes a keyword $w \in \{0,1\}^k$ (instead of an index $i \in \{1,\ldots,n\}$ as defined in Definition 4.1) and $\mathsf{PIR.Answer}$ takes a keyword-value-pair database denoted by $\mathcal{DB}_{\mathsf{W}}$. All other properties of PIR remain the same.

4.3 Symmetric and Public-key Encryption

Here, we define symmetric-key encryption in Definition 4.2 and present its semantic security in Definition 4.3.

Definition 4.2 (Symmetric-key Encryption [14, 51])**.** Fix a security parameter $\lambda \in \mathbb{N}$. A symmetric-key encryption scheme consists of three efficient (possibly randomized) algorithms $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda) \to \mathsf{sk}$. Takes as input a security parameter λ. Outputs a new secret key sk.
- $\mathsf{Enc}(\mathsf{sk}, m) \to c$. Takes as input the secret key sk and message m. Outputs a ciphertext c.
- $\mathsf{Dec}(\mathsf{sk}, c) \to m$. Takes as input a secret key sk and ciphertext c. Outputs a plaintext message m.

The above must satisfy *correctness* and *semantic security*. Correctness holds if $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{sk}, m)) = m$.

For convenience, we will use the simulation-based definition of semantic security [50, 68].

Definition 4.3 (Semantic Security [50, 68])**.** Fix security parameter $\lambda \in \mathbb{N}$ and message length $\ell \leq \mathsf{poly}(\lambda)$. A symmetric-key encryption scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$

is said to be *semantically secure* if for all $p \leq \mathsf{poly}(\lambda)$ and messages $m_1 \dots m_p \in \{0,1\}^\ell$, there exists an efficient simulator $\mathcal{S}$ such that for

$$\mathsf{View} := \left\{ c_1, \dots, c_p : \begin{array}{l} \mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda), \\ \mathsf{c}_i \leftarrow \mathsf{Enc}(\mathsf{sk}, m_i) \end{array} \;\middle|\; i \in \{1, \dots, p\} \right\},$$

it holds that $\mathsf{View} \approx_c \mathcal{S}(1^\lambda, \ell, p, z)$, where $z \in \{0,1\}^*$ denotes arbitrary auxiliary information. In words, a ciphertext reveals no information on the encrypted message m.

Public-key encryption. Public-key encryption shares the syntax of symmetric-key encryption described in Definition 4.2, except that KeyGen outputs a *key pair* $(\mathsf{pk}, \mathsf{sk})$ and Enc takes the public key pk instead of the secret key sk. The correctness and semantic security definitions are then extended in a natural way.

Chapter 5

Pirac Definitions

In this chapter, we define Pirac. Our definition shares the syntax of PIR (Definition 4.1) and introduces additional functionality to support access authorization. In Section 5.1, we define access authorization to complement our Pirac definition. Specifically, we define static-database, dynamic-database, and forward-secret authorization, which were highlighted in Section 1.1.

Authorization Predicate. We start by defining an authorization predicate $\mathsf{AuthVerify}$, that determines authorization with respect to an access policy $\Lambda := (\kappa_1, \ldots, \kappa_n)$, and has the following syntax:

> $\mathsf{AuthVerify}(\Lambda, i, \kappa) \to \mathsf{yes}$ or no. Takes as input the access policy Λ, index i, and an access key κ. Outputs yes if and only if κ is a valid access key with respect to Λ and i.

We use the $\mathsf{AuthVerify}$ predicate to define Pirac in Definition 5.1.

Example: Equality Authorization Predicate. In our constructions, we define Λ to be an ordered list of access keys $(\kappa_1, \ldots, \kappa_n)$ and $\mathsf{AuthVerify}$ to be the "equality predicate" that outputs yes if and only if κ is the i-th access key in Λ. We note that this model of access control is fully compatible with generic access policies [13, 19, 31].

Definition 5.1 (Private Information Retrieval with Access Control)**.** Fix security parameter $\lambda \in \mathbb{N}$ and database parameters $n, \ell \in \mathbb{N}$. Let $\mathcal{DB}$ be an n-record database

with ℓ-bit records. Pirac consists of four efficient (possibly randomized) algorithms (KeyGen, Query, Answer, $\mathsf{Recover}$) with the following syntax:

- $\mathsf{KeyGen}(1^\lambda, i) \to \kappa_i$. Takes as input a security parameter λ and index $i \in \{1, \ldots, n\}$. Outputs an access key κ_i for the i-th record.
- $\mathsf{Query}(\kappa, i) \to \mathsf{q}$. Takes as input the access key κ and index $i \in \{1, \ldots, n\}$. Outputs a query q.
- $\mathsf{Answer}(\mathcal{DB}, \Lambda, \mathsf{q}) \to \mathsf{c}$. Takes as input a database $\mathcal{DB}$, access policy Λ, and query q. Outputs answer c.
- $\mathsf{Recover}(\kappa, \mathsf{c}) \to (\mathsf{x} \text{ or } \perp)$. Takes as input access key κ and a query answer c. Outputs database record x or $\perp$ (fail).

We let $\Lambda := (\kappa_1, \ldots, \kappa_n)$, where κ_i are generated according to KeyGen. The above functionality must satisfy: *correctness*, *privacy*, *authorization*, and *efficiency*. We define authorization in Section 5.1.

- **Correctness.** For all databases $\mathcal{DB}$, indices $i \in \{1, \ldots, n\}$, access policies Λ, and access keys κ, if $\mathsf{AuthVerify}(\Lambda, i, \kappa) = \mathsf{yes}$, then

$$\Pr\left[\begin{array}{l} \mathsf{q} \leftarrow \mathsf{Query}(\kappa, i) \\ \mathsf{c} \leftarrow \mathsf{Answer}(\mathcal{DB}, \Lambda, \mathsf{q}) \\ \mathsf{x} \leftarrow \mathsf{Recover}(\kappa, \mathsf{c}) \end{array} : \mathsf{x} = \mathcal{DB}[i]\right] = 1 - \mathsf{negl}(\lambda).$$

- **Privacy.** For all indices $i \in \{1, \ldots, n\}$ and access keys κ, there exists an efficient simulator $\mathcal{S}$ such that $\mathsf{Query}(\kappa, i) \approx_c \mathcal{S}(1^\lambda)$. That is, the query reveals no information on the access key κ and index i to computationally-bounded adversaries.
- **Efficiency.** Fix a PIR scheme. Pirac is said to be efficient (with respect to PIR) if the server and user work and total communication is at most a factor of $\mathsf{poly}(\lambda)$ greater compared to PIR.

5.1 Defining Authorization

We now turn to define the access authorization property for Pirac. As already mentioned in Section 1.1, we consider three "tiers" of access control: *static-database*, *dynamic-database*, and *forward-secret* authorization. We specify each authorization property using the standard "real vs. ideal" simulation paradigm [50, 68]. For each authorization tier, we specify an *ideal* functionality modeling a world where access control is enforced by a trusted party. Proving that a Pirac protocol meets the specified authorization functionality then requires constructing an efficient simulator that generates a computationally-indistinguishable view to that of the adversary interacting with the database server through the real protocol.

We start by describing *static-database authorization*, which captures a setting with a fixed database and access policy. We then proceed to define dynamic-database and forward-secret authorization. We highlight the differences between the ideal functionalities.

Definition 5.2 (Static-database Authorization)**.** Fix security parameter $\lambda \in \mathbb{N}$, and database parameters $n, \ell \in \mathbb{N}$. We say that Pirac satisfies *static-database authorization* if, for all $p \leq \mathsf{poly}(\lambda)$, it instantiates the ideal functionality described in Functionality 1.

Functionality 1: Static-database Authorization

Public parameters: $\lambda, n, \ell \in \mathbb{N}$ and $\mathsf{AuthVerify}$.

Server input: database $\mathcal{DB}$ and policy Λ.

User input: query indices $i_1, \ldots, i_p$ and access keys $\kappa_1, \ldots, \kappa_p$.

Procedure:

1: **foreach** $j \in \{1, \ldots, p\}$:

- **if** $\mathsf{AuthVerify}(\Lambda, i_j, \kappa_j) = \mathsf{yes}$ **then** set $\tilde{\mathsf{x}}_j := \mathcal{DB}[i_j]$.
- **if** $\mathsf{AuthVerify}(\Lambda, i_j, \kappa_j) = \mathsf{no}$ **then** set $\tilde{\mathsf{x}}_j := \perp$.

2: Output $(\tilde{\mathsf{x}}_1, \ldots, \tilde{\mathsf{x}}_p)$ to the user and $\perp$ to the server.

Definition 5.3 (Dynamic-database Authorization)**.** Let λ, n, ℓ, p be as in Definition 5.2. Pirac satisfies *dynamic-database authorization* if it instantiates the ideal functionality described in Functionality 2.

Functionality 2: Dynamic-database Authorization

Public parameters: $\lambda, n, \ell \in \mathbb{N}$ and $\mathsf{AuthVerify}$.

Server input: databases $\mathcal{DB}_1, \ldots, \mathcal{DB}_p$ and policy Λ.

User input: query indices $i_1, \ldots, i_p$, and access keys $\kappa_1, \ldots, \kappa_p$.

Procedure:

1: **foreach** $j \in \{1, \ldots, p\}$:

- **if** $\mathsf{AuthVerify}(\Lambda, i_j, \kappa_j) = \mathsf{yes}$ **then** set $\tilde{\mathsf{x}}_j := \mathcal{DB}_j[i_j]$.
- **if** $\mathsf{AuthVerify}(\Lambda, i_j, \kappa_j) = \mathsf{no}$ **then** set $\tilde{\mathsf{x}}_j := \perp$.

2: Output $(\tilde{\mathsf{x}}_1, \ldots, \tilde{\mathsf{x}}_p)$ to the user and $\perp$ to the server.

In words, dynamic-database authorization strengthens static-database authorization to capture a setting where queries are computed on potentially different databases. It requires that no information on database $\mathcal{DB}_j$ is leaked when $\mathsf{AuthVerify}$ outputs no on index i_j. For example, in a setting where the database records are dynamic, users should only learn that the i-th record was updated if they also have permission to access it.

Claim 1. Dynamic-database authorization (Definition 5.3) implies static-database authorization (Definition 5.2).

Proof. Consider an efficient simulator $\mathcal{S}'$ that can simulate Functionality 2 of Definition 5.3. We can immediately construct an efficient simulator $\mathcal{S}$ for Functionality 1 as follows: given the database $\mathcal{DB}$ along with other input parameters specified in the ideal functionality, $\mathcal{S}$ runs $\mathcal{S}'$ on p copies of $\mathcal{DB}$. It then follows that if $\mathcal{S}'$ correctly simulates the p copies of $\mathcal{DB}$, $\mathcal{S}$ correctly simulates the Functionality 1. □

Definition 5.4 (Forward-secret Authorization)**.** Let λ, n, ℓ, p be as in Definition 5.2. Fix an efficient and deterministic Update function that takes in an access policy Λ

and auxiliary information $z \in \{0,1\}^*$ and outputs an updated access policy Λ'. Pirac satisfies *forward-secret authorization* if it instantiates the ideal functionality described in Functionality 3 parameterized by Update.

Functionality 3: Forward-secret Authorization

Public parameters: $\lambda, n, \ell \in \mathbb{N}$, $(z_1, \ldots, z_p \mid z_i \in \{0,1\}^*)$, Update and AuthVerify.

Server input: databases $\mathcal{DB}_1, \ldots, \mathcal{DB}_p$ and initial access policy Λ.

User input: query indices $i_1, \ldots, i_p$, and access keys $\kappa_1, \ldots, \kappa_p$.

Procedure:

1: Initialize $\Lambda_0 := \Lambda$.

2: **foreach** $j \in \{1, \ldots, p\}$:

- $\Lambda_j \leftarrow \mathsf{Update}(\Lambda_{j-1}, z_j)$.
- **if** $\mathsf{AuthVerify}(\Lambda_j, i_j, \kappa_j) = \mathsf{yes}$ **then** set $\tilde{\mathsf{x}}_j := \mathcal{DB}_j[i_j]$.
- **if** $\mathsf{AuthVerify}(\Lambda_j, i_j, \kappa_j) = \mathsf{no}$ **then** set $\tilde{\mathsf{x}}_j := \perp$.

3: Output $(\tilde{\mathsf{x}}_1, \ldots, \tilde{\mathsf{x}}_p)$ to the user and $\perp$ to the server.

In words, forward-secret authorization strengthens the dynamic-database authorization definition to capture a setting where queries are computed on potentially different databases using potentially different access policies. Additionally, the Update function represents the procession of time, where subsequent access policies are derived from previous ones. For instance, if the Update function is an identity function, the access policy remains unchanged. However, if Update continuously modifies the access keys, Functionality 3 requires that users should not gain knowledge about past records after being granted permission to access an updated record at a future point in time.

Claim 2. Forward-secret authorization (Definition 5.4) implies dynamic-database authorization (Definition 5.3).

Proof. Consider an efficient simulator $\mathcal{S}'$ that can simulate Functionality 3 of Definition 5.4. We can immediately construct an efficient simulator $\mathcal{S}$ for Functionality 2 as follows: given the input parameters, $\mathcal{S}$ runs $\mathcal{S}'$ with an identity Update function

$(\Lambda \leftarrow \mathsf{Update}(\Lambda, \_))$. Then it is easy to see that if $\mathcal{S}'$ correctly simulates Functionality 3, $\mathcal{S}$ correctly simulates Functionality 2. □

5.1.1 Breach Resilient Authorization

Breach resilience requires authorization (static-database, dynamic-database, or forward-secret) to hold in Pirac even when a copy of the entire server state is revealed to the adversary. We note that breach resilience only requires providing security against a snapshot adversary that obtains a copy of the server state [9]—it does not assume that the adversary has a persistent view of the server state, as that would trivially break authorization.

Definition 5.5 (Breach Resilience [9])**.** We say that Pirac is *breach resilient* if there exists an efficient simulator $\mathcal{S}$ such that $\{\mathcal{DB}, \Lambda\} \approx_c \mathcal{S}(1^\lambda, z)$, where $\mathcal{DB}$ and Λ are as defined in Definition 5.1, and $z \in \{0,1\}^*$ denotes arbitrary auxiliary information.

Chapter 6

Pirac Construction

In this chapter, we realize Pirac in two ways: (1) through keyword-private PIR-by-keywords and (2) using any semantically-secure symmetric encryption. In Section 6.1, we describe our construction of Pirac from keyword-private PIR-by-keywords. In Section 6.2, we describe our construction of Pirac from symmetric-key encryption. Finally, in Section 6.3, we provide some practical optimizations related to our constructions.

6.1 Pirac from PIR-by-keywords

Before diving into our construction, we first formalize the keyword-privacy property we require of the PIR-by-keywords scheme.

6.1.1 Keyword-private PIR-by-keywords

While it is known how to turn *any* PIR scheme into a PIR-by-keywords scheme [29, 78], doing so leaks to the user the keywords that exist in the database. In our construction of Pirac (Construction 2), we use the keywords as the access control keys. Therefore, revealing the keywords that exist in the database to the user would immediately subvert access control. Constructing PIR-by-keywords, *without* leaking the keywords to users presents several challenges. However, in Appendix A, we identify

two existing PIR-by-keywords schemes for both single-server [71] and multi-server settings [16] that satisfy this property.

Definition 6.1 (Keyword Privacy)**.** Fix security parameter $\lambda \in \mathbb{N}$ and database parameters $n, \ell \in \mathbb{N}$. A PIR-by-keywords scheme defined by $(\mathsf{Query}, \mathsf{Answer}, \mathsf{Recover})$ is said to be *keyword private* if for all databases $\mathcal{DB}$, sets of keywords $\mathsf{W} := (w_1, \ldots, w_n)$, and queries q for a keyword w, there exists an efficient simulator $\mathcal{S}$ such that:

$$\{\mathsf{c} : \mathsf{c} \leftarrow \mathsf{Answer}(\mathcal{DB}_{\mathsf{W}}, \mathsf{q})\} \approx_c \mathcal{S}(1^\lambda, w, \tilde{\mathsf{x}}, z),$$

where $\tilde{\mathsf{x}} := \mathcal{DB}_{\mathsf{W}}[w]$ if $w \in \mathsf{W}$, $\tilde{\mathsf{x}} := \perp$ otherwise, and $z \in \{0,1\}^*$ denotes arbitrary auxiliary information. In words, the query answer c does not reveal anything beyond $(w, \mathcal{DB}_{\mathsf{W}}[w])$.

6.1.2 Constructing Pirac from Keyword-private PIR-by-keywords

Pirac from PIR-by-keywords
with forward-secret authorization

Database $\mathcal{DB} := (\mathsf{x}_1, \ldots, \mathsf{x}_n)$ and access policy $\Lambda := (\kappa_1, \ldots, \kappa_n)$.
A PIR-by-keywords scheme $\mathsf{PIR} = (\mathsf{Query}, \mathsf{Answer}, \mathsf{Recover})$.

- $\mathsf{KeyGen}(1^\lambda, i)$:
  1: $k := \lambda + \log n$.
  2: $\kappa_i \leftarrow_R \{0,1\}^k$.
  3: Output κ_i.

- $\mathsf{Answer}(\mathcal{DB}, \Lambda, \mathsf{q})$:
  1: $\mathsf{c} \leftarrow \mathsf{PIR.Answer}(\mathcal{DB}_\Lambda, \mathsf{q})$.
  2: Output c.

- $\mathsf{Query}(\kappa, \_)$:
  1: $\mathsf{q} \leftarrow \mathsf{PIR.Query}(1^\lambda, \kappa)$.
  2: Output q.

- $\mathsf{Recover}(\kappa, \mathsf{c})$:
  1: $\mathsf{x} \leftarrow \mathsf{PIR.Recover}(\mathsf{c})$.
  2: Output x.

Construction 2: Forward-secret Pirac from PIR-by-keywords.

In Construction 2, we realize Pirac with forward-secret authorization (Definition 5.4) using any keyword-private PIR-by-keywords scheme (Definition 6.1). We let the access policy Λ be the ordered list of n access keys, where κ_i is the access key for record $\mathcal{DB}[i]$. This corresponds to an $\mathsf{AuthVerify}$ that outputs yes if and only if q

is accessing the i-th record and the access key κ is equal to $\kappa_i \in \Lambda$. We then let PIR be a PIR-by-keywords scheme and $\mathcal{DB}_\Lambda$ denote the database where κ_i is the *keyword* for the i-th record. We formally prove correctness and security in Chapter 7.

We note that by Claims 1 and 2, Construction 2 also achieves dynamic-database and static-database authorization.

6.2 Pirac from Encryption

In Section 6.2.1, we start by constructing Pirac from symmetric-key encryption which achieves dynamic-database authorization. In Section 6.2.2, we explain how to upgrade the construction to provide forward-secret authorization. Finally, in Section 6.2.3, we show how to make Pirac breach resilient using public-key encryption.

6.2.1 Pirac with Dynamic-database Authorization

In Construction 3, we realize Pirac from symmetric-key encryption (Definition 4.2) to satisfy dynamic-database authorization. At a high level, our approach augments the strawman Pirac construction from Section 2.1 to satisfy the dynamic-database authorization property (Definition 5.3). The main idea to prevent metadata leakage is to *re-randomize* the database encryption by re-encrypting the database *prior* to computing the PIR response. In this way, each response appears pseudorandom to users who do not have permission to access the record. See Chapter 7 for security analysis.

6.2.2 Upgrading to Forward Secrecy

To upgrade Construction 3 to achieve forward-secret authorization (Definition 5.4), we present a function $\mathsf{KeyRefresh}$ that instantiates Update (defined in Definition 5.4) and "refresh" the access keys between queries (in addition to re-encrypting the database records). Let t denote the current query epoch. Then, we define $\mathsf{KeyRefresh}$ such that it outputs a fresh access key $\kappa_i^{(t+1)}$ for epoch $t+1$ when given the key $\kappa_i^{(t)}$ for epoch

Pirac from encryption (dynamic-database authorization)

Database $\mathcal{DB} := (\mathsf{x}_1, \ldots, \mathsf{x}_n)$ and access policy $\Lambda := (\kappa_1, \ldots, \kappa_n)$.
$\mathsf{PIR} = (\mathsf{Query}, \mathsf{Answer}, \mathsf{Recover})$ is any PIR scheme.
$\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is any symmetric encryption scheme.

- $\mathsf{KeyGen}(1^\lambda, i)$:
  1: $\mathsf{sk}_i \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^\lambda)$.
  2: Output $\kappa_i := \mathsf{sk}_i$.
- $\mathsf{Query}(\kappa, i)$:
  1: $\mathsf{q} \leftarrow \mathsf{PIR.Query}(1^\lambda, i)$.
  2: Output q.
- $\mathsf{Answer}(\mathcal{DB}, \Lambda, \mathsf{q})$:
  1: $\widetilde{\mathcal{DB}} := (\bot, \ldots, \bot)$.
  2: $\widetilde{\mathcal{DB}}[i] \leftarrow \mathcal{E}.\mathsf{Enc}(\kappa_i, \mathcal{DB}[i])$, $\forall i \in \{1, \ldots, n\}$.
  3: $\mathsf{c} \leftarrow \mathsf{PIR.Answer}(\widetilde{\mathcal{DB}}, \mathsf{q})$.
  4: Output c.
- $\mathsf{Recover}(\kappa, \mathsf{c})$:
  1: $\tilde{\mathsf{c}} \leftarrow \mathsf{PIR.Recover}(\mathsf{c})$.
  2: $\mathsf{x} \leftarrow \mathcal{E}.\mathsf{Dec}(\kappa, \tilde{\mathsf{c}})$.
  3: Output x.

Construction 3: Dynamic-database Pirac from encryption.

t. Crucially, KeyRefresh must be (1) a deterministic and (2) one-way function so as to (1) ensure that users can refresh keys themselves and (2) provide forward secrecy from epoch-to-epoch.

We realize KeyRefresh in Construction 4 using a pseudorandom function (PRF) F [51]. The idea is to start with a random seed s_1. This seed is used as the randomness in PIRAC.KeyGen to generate κ_1 for epoch $t = 1$. Then, for each subsequent epoch, we apply F (keyed using the seed s_t) to generate a fresh psuedorandom seed for epoch $t + 1$. Without loss of generality, we let $\kappa_i^{(t)} = (\kappa_{t,i}, \mathsf{s}_{t,i})$ where $\mathsf{s}_{t,i}$ is the seed used as the randomness to generate $\kappa_{t,i}$.

Observe that, using KeyRefresh, a user with an access key at epoch t can compute the access key for all subsequent epochs. However, an access key for epoch t does not allow computing the access key for epoch $t - 1$. In Chapter 7, we prove that instantiating Update by applying KeyRefresh on *every access key* of the access policy,

PRF $F : \{0,1\}^\lambda \times \{0,1\}^p \to \{0,1\}^\lambda$ for $p \leq \mathsf{poly}(\lambda)$.

$\mathsf{KeyRefresh}(\kappa_i^{(t)}, t)$:

1: Parse $\kappa_i^{(t)} = (\kappa_{t,i}, \mathsf{s}_{t,i})$.
2: $\mathsf{s}_{t+1,i} \leftarrow F(\mathsf{s}_{t,i}, t)$.
3: $\kappa_{t+1,i} \leftarrow \mathsf{PIRAC.KeyGen}(1^\lambda, i; \mathsf{s}_{t+1,i})$.
▷ PIRAC.KeyGen uses $\mathsf{s}_{t+1,i}$ as the randomness source.
4: Output $\kappa_i^{(t+1)} := (\kappa_{t+1,i}, \mathsf{s}_{t+1,i})$.

Construction 4: Forward-secure key refresh procedure.

together with Construction 3, results in a Pirac scheme that achieves forward-secret authorization as described in Functionality 3.

6.2.3 Making Pirac Breach Resilient

Here, we describe how to make Pirac from encryption breach resilient.

Breach resilience and dynamic-database authorization. We modify Construction 3 to provide breach resilience by using re-randomizable public-key encryption [27, 45, 53], as opposed to symmetric-key encryption. A re-randomizable public key encryption scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ReRand}, \mathsf{Dec})$ is equipped with an efficient algorithm ReRand that takes as input a public key pk and ciphertext c and outputs a *randomized* ciphertext $\tilde{c}$. We can then define $\Lambda := (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ to consist of the *public* keys and let κ_i be the *secret* key corresponding to pk_i. Similarly, we modify $\mathsf{AuthVerify}$ to output yes if and only if κ is the secret key corresponding to pk_i (e.g., if it holds that $\mathsf{Dec}(\kappa, \mathsf{Enc}(\mathsf{pk}_i, 0^\ell)) = 0^\ell$). In Construction 5, we describe how to modify $\mathsf{PIRAC.Answer}$ from Construction 3 to use public-key re-randomization to achieve breach resilience.

Breach resilience and forward-secret authorization. While public-key re-randomization makes Construction 3 resilient to snapshot adversaries and provides dynamic-database authorization, it is important to consider whether the upgrade to forward-secret authorization, described in Section 6.2.2, remains compatible. Observe that the user can locally update their secret key by applying $\mathsf{KeyRefresh}$, even when using public

PIRAC.Answer($\mathcal{DB}, \Lambda, \mathsf{q}$):

1: $\widetilde{\mathcal{DB}} := (\bot, \ldots, \bot)$.
2: $\widetilde{\mathcal{DB}}[i] \leftarrow \mathcal{E}.\mathsf{ReRand}(\mathsf{pk}_i, \mathcal{DB}[i])$, $\forall i \in \{1, \ldots, n\}$.
3: $\mathcal{DB} \leftarrow \widetilde{\mathcal{DB}}$ and store the updated database $\mathcal{DB}$.
4: $\mathsf{c} \leftarrow \mathsf{PIR.Answer}(\mathcal{DB}, \mathsf{q})$.
5: Output c.

Construction 5: Breach resilience modification for Construction 3.

key encryption. However, the server cannot do so because the randomness used to generate the key pair must remain secret. We can get around this by having the access authority provide the server with a *list* of public keys, computed by repeatedly applying KeyRefresh, such that the t-th public key is used for epoch t. A similar solution is described by Canetti et al. [26] in the context of forward-secret public-key encryption. We leave open the problem of finding a more practical solution for breach resilience with forward-secret authorization.

6.3 Practical Considerations

We discuss four practical optimizations we can apply to Pirac.

Query-independent pre-processing. We observe that the access control components (key refreshing and database re-encryption) can be computed independently of the user's query. This allows for server-side pre-processing (at the cost of extra storage) when using Pirac from encryption (Section 6.2).

Periodic database updates. We observe that it is only necessary to have dynamic-database (or forward-secret) authorization when the database (or access policy) undergoes updates. Therefore, we can avoid the re-randomization of the ciphertext for every query if we assume that the database server only updates the database at fixed time intervals (e.g., once an hour). This has the potential to significantly increase performance of practical deployments of Construction 3, since re-randomization will only be required *periodically* (e.g., once an hour) rather than per query. Similarly, when considering forward-secret authorization, we can periodically update the access

policy to avoid re-keying per query. In this way, re-keying for forward-secret authorization (using KeyRefresh) is only necessary between updates, rather than per query. We showcase the efficiency improvement of doing periodic updates in Chapter 8.

Keychain checkpoints. The key-refresh procedure (used to achieve forward secrecy) requires users to compute a PRF proportionally to the number of queries made through Pirac (in the worst case). Therefore, refreshing keys can become a computational bottleneck for users. Because the server has to evaluate the PRF per query regardless, it can store "checkpoints" along the way. These checkpoints consist simply of the keys at time $t + i$ where the access key at time $t + i - 1$ is required to access the corresponding access key at time $t + i$. Then, a user can query for an access key at the latest checkpoint if the total query latency is *less* than the time required to evaluate the PRF up to the current time. Let T_{q} be the query latency of Pirac when computed over a database consisting of *access keys* and let T_F be the time it takes to evaluate the PRF for the slowest user. Then, the server should keep one checkpoint every $O(T_{\mathsf{q}}/T_F)$ queries to ensure straggling users can refresh their keys without needing to recompute the PRF.

Efficient access revocation. We describe an efficient way to keep separate access policies for different users so as to make revocation more efficient. Let B be a bound on the number of users who have access to any given record in the database. We then set up B access policies $(\Lambda_1, \ldots, \Lambda_B)$. To make a PIR query, the user sends the policy index $t \in \{1, \ldots, B\}$ (in the clear) along with the PIR query. The server then runs the PIRAC.Answer protocol, as before, but now uses Λ_t in PIRAC.Answer. All the constructions and optimizations work as before, with the only difference being that the server stores B access policies instead of one. To revoke the access of a user holding the access key of the i-th entry from Λ_t, the access authority can simply replace $\Lambda_t[i]$ with a new access key. The above construction adds no processing overhead to the server or user computation, but does require $O(\lambda n B)$ additional storage on the server.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 7

Security Analysis

In Section 7.1, we prove security of our Pirac construction from PIR-by-keywords. In Section 7.2, we prove security of our Pirac construction from symmetric encryption and authorization upgrades.

7.1 Security of Pirac from PIR-by-keywords

Here, we analyze the security of Construction 2 when using a keyword-private PIR-by-keywords scheme.

Claim 3. Let $\mathsf{PIR} = (\mathsf{Query}, \mathsf{Answer}, \mathsf{Recover})$ be a PIR-by-keywords scheme satisfying Definition 6.1 (keyword privacy). Construction 2 satisfies the Pirac properties defined in Definition 5.1 and achieves forward-secret authorization, as defined in Definition 5.4.

Proof. We prove the four properties required: *correctness*, *privacy*, *efficiency*, and *forward-secret authorization*.

Correctness. Follows directly from the correctness property of PIR-by-keywords (see Definition 4.1 and Section 4.2.1). Specifically, if κ_i is in the set of keywords Λ then, by correctness of PIR-by-keywords, the user obtains the record $\mathcal{DB}[\kappa_i] = \mathcal{DB}[i]$.

Privacy. Follows directly from the privacy property of PIR-by-keywords (see Definition 4.1 and Section 4.2.1) given that the query consists solely of the output of

$\mathsf{PIR.Query}$.

Efficiency. Let $C(n, \ell)$ and $W(n, \ell)$ be the communication and server-side work of PIR, respectively. Construction 2 incurs at most $O(\lambda \cdot W(n, \ell))$ server-side work and at most $O(\lambda \cdot C(n, \ell))$ communication. To see why, note that $C(n, \ell)$ and $W(n, \ell)$ are the baseline communication and work for $\lceil \log n \rceil$-sized keywords. In contrast, the keywords in Construction 2 are each $(\lambda + \lceil \log n \rceil)$ bits in total. Therefore, the PIR-by-keywords problem with λ-bit keywords would require at most $O(\lambda)$ times the work of one invocation, resulting in the above bounds.

Forward-secret authorization. We prove that forward-secret authorization is met by reducing to the keyword-privacy property of the PIR scheme. Let $\mathcal{S}'$ be the keyword privacy simulator from Definition 6.1. We prove that Construction 2 instantiates the forward-secret functionality specified in Functionality 3. To do so, we construct an efficient simulator $\mathcal{S}$ for the view of the adversary interacting with the server via Construction 2. $\mathcal{S}$ proceeds as follows:

1: Receive as input the set $\{i_j, \kappa_j, \tilde{\mathsf{x}}_j \mid j \in \{1, \ldots, p\}\}$, where $i_j, \kappa_j, \tilde{\mathsf{x}}_j$ are the user's inputs and output, as specified in Functionality 3.

2: Output $\bigcup_{j=1}^{p} \mathcal{S}'(1^\lambda, \kappa_j, \tilde{\mathsf{x}}_j)$.

We now argue that the output of $\mathcal{S}$ is computationally indistinguishable to the real view of the adversary interacting with the server. Notice that the real view of the adversary consists of the set of answers computed by the server in response to p queries:

$$\mathsf{Real} := \{\mathsf{c}_1, \ldots, \mathsf{c}_p : \mathsf{c}_j \leftarrow \mathsf{Answer}(\mathcal{DB}_j, \Lambda_j, \mathsf{q}_j) \mid j \in \{1, \ldots, p\}\},$$

where $\mathsf{q}_1, \ldots, \mathsf{q}_p$ are chosen arbitrarily by the adversary.

By the specification of Functionality 3, we have that $\tilde{\mathsf{x}}_j = \mathcal{DB}_j[i_j]$ if and only if $\mathsf{AuthVerify}(\Lambda_j, i_j, \kappa_j) = \mathsf{yes}$.

By our definition of $\mathsf{AuthVerify}$, we have that $\mathsf{AuthVerify}(\Lambda_j, i_j, \kappa_j) = \mathsf{yes}$ if and only if Λ_j is the i_j-th key in Λ_j. Furthermore, because of the one-to-one correspondence between κ_j and i_j, we additionally have that $\tilde{\mathsf{x}}_j = \mathcal{DB}_{j_{\Lambda_j}}[\kappa_j]$ if and only if Λ_j is the

i_j-th key in Λ_j.

Note that this matches the definition of PIR-by-keywords, where $\tilde{\mathsf{x}}_j = \mathcal{DB}_{j\Lambda_j}[\kappa_j]$ if and only if Λ_j is the i_j-th key in Λ_j. Therefore, κ_j and $\tilde{\mathsf{x}}$ are distributed identically to the user's input and output in the ideal functionality implicit in the keyword-privacy definition (Definition 6.1). From this, we conclude that $\mathcal{S}'$ can simulate the view of the adversary when interacting with the server on one query and one database. We then have that, for any $j \in \{1, \ldots, p\}$, $\mathcal{S}'$ generates a computationally indistinguishable view for the query answer c_j computed on database $\mathcal{DB}_j$. Then, by sequential composition [25, 50, 68], we get that the view output by $\mathcal{S}$ is computationally indistinguishable to Real, which proves that Construction 2 instantiates Functionality 3.

Lastly, we note that because each access key (keyword) is chosen from the space $\{0,1\}^{\lambda+\lceil \log n \rceil}$, guessing the correct keyword of the i-th record has negligible (in λ) probability of success. □

7.2 Security of Pirac from Encryption

Here, we analyze the security of the constructions from Section 6.2.

Claim 4. Let $\mathsf{PIR} = (\mathsf{Query}, \mathsf{Answer}, \mathsf{Recover})$ be any PIR scheme. Construction 3 satisfies the Pirac properties defined in Definition 5.1.

Proof. We prove the three properties of Construction 3 as defined in Definition 5.1.

- *Correctness:* Follows immediately from the correctness property of PIR (see Definition 4.1) and the correctness of the symmetric-key encryption scheme $\mathcal{E}$ (see e.g., [14, 51]).
- *Privacy:* Follows immediately from the privacy property of PIR (see Definition 4.1). Specifically, in Construction 3, the server only obtains q, which is output by $\mathsf{PIR.Query}$ and is therefore efficiently simulatable by Definition 4.1.
- *Efficiency:* Let $C(n, \ell)$ and $W(n, \ell)$ be the communication and server-side work of PIR. By inspection, it is clear that Construction 3 has at most $O(\lambda n \ell) + W(n, \ell)$ server-side work and $C(n, \ell)$ communication, when $\mathcal{E}$ is a rate-1 encryption scheme.

□

In Section 7.2.1, we prove that Construction 3 satisfies dynamic-database authorization. Then, in Sections 7.2.2 and 7.2.3 we prove security of the forward-secret authorization upgrade and breach resilience transformation.

7.2.1 Proof of Dynamic-database Authorization

We first prove some useful lemmas.

Lemma 1. *Let* PIR *be any PIR scheme satisfying Definition 4.1. The trivial PIR scheme [30], where the user sends* $\mathsf{q} := \perp$ *to the server and the server sends the entire database* $\mathcal{DB}$ *to the user reveals strictly more information to the user than* PIR*.*

Proof. The proof follows trivially from the fact that Definition 4.1 (1) makes no database privacy guarantees and (2) has a communication efficiency guarantee that (information theoretically) prevents revealing the full database to the user in one query. □

Lemma 2. *Let* $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be any semantically-secure encryption scheme. We can assume, without loss of generality, that* $\mathcal{E}.\mathsf{Dec}$ *outputs* $\perp$ *when provided with the wrong decryption key.*

Proof. It suffices to use authenticated encryption [14], which fails to decrypt (outputs $\perp$) when given the wrong key. □

Claim 5. Construction 3 instantiates the dynamic-database authorization functionality specified in Functionality 2, assuming the existence of semantically-secure symmetric-key encryption.

Proof. We prove that dynamic-database authorization is met by reducing to the semantic security of the encryption scheme $\mathcal{E}$.

First, by Lemma 1, we can assume, without loss of generality, that $\mathsf{c} = \widetilde{\mathcal{DB}}$ (the encrypted database, as defined in Construction 3) and $\mathsf{q} = \perp$. Then, to prove that Construction 3 satisfies the dynamic-database authorization functionality, we

construct an efficient simulator $\mathcal{S}$ for the view of the adversary interacting with the server via Construction 3. Let $\mathcal{S}'$ be the simulator from Definition 4.3 and let $p \leq \frac{\mathsf{poly}(\lambda)}{n}$ be an integer. Define $\mathcal{S}$ as follows:

1: Receive as input the set $\{i_j, \kappa_j, \tilde{\mathsf{x}}_j \mid j \in \{1, \ldots, p\}\}$, where $i_j, \kappa_j, \tilde{\mathsf{x}}_j$ are the user's inputs and output, as specified in Functionality 2.

2: $(\mathsf{c}_{1,1}, \ldots, \mathsf{c}_{1,n}, \mathsf{c}_{p,1}, \ldots, \mathsf{c}_{p,n}) \leftarrow \mathcal{S}'(1^\lambda, \ell, np, z)$.

3: Set $\mathsf{c}_j := (\mathsf{c}_{j,1}, \ldots, \mathsf{c}_{j,n})$, $\forall j \in \{1, \ldots, p\}$.

4: $\forall j \in \{1, \ldots, p\}$, if $\tilde{\mathsf{x}}_j \neq \perp$, set $\mathsf{c}_{j,i_j} \leftarrow \mathcal{E}.\mathsf{Enc}(\kappa_j, \tilde{\mathsf{x}}_j)$.

We now argue that the output of $\mathcal{S}$ is computationally indistinguishable to the real view of the adversary interacting with the server. Notice that the real view consists of the set of answers computed on the p queries:

$$\mathsf{Real} := \{(\mathsf{c}_1, \ldots, \mathsf{c}_p) : \mathsf{c}_i \leftarrow \mathsf{Answer}(\mathcal{DB}_j, \mathsf{q}_j) \mid j \in \{1, \ldots, p\}\},$$

where each q_j is arbitrarily chosen by the adversary. By Lemma 1 and the description of Construction 3, we can further interpret Real as a $p \times n$ matrix of ciphertexts. Specifically, we have $\mathsf{c}_j = (\mathsf{c}_{j,1}, \ldots, \mathsf{c}_{j,n})$, where $\mathsf{c}_{j,k} \leftarrow \mathcal{E}.\mathsf{Enc}(\kappa_k, \mathcal{DB}_j[k])$. Therefore, we get that Real is distributed as:

$$\left\{ \begin{bmatrix} \mathsf{c}_{1,1}, \ldots, \mathsf{c}_{1,n} \\ \vdots \\ \mathsf{c}_{p,1}, \ldots, \mathsf{c}_{p,n} \end{bmatrix} : \mathsf{c}_{j,k} \leftarrow \mathcal{E}.\mathsf{Enc}(\kappa_k, \mathcal{DB}_j[k]) \;\middle|\; \begin{array}{l} j \in \{1, \ldots, p\}, \\ k \in \{1, \ldots, n\} \end{array} \right\},$$

For each $\tilde{\mathsf{x}}_j \neq \perp$, $\mathcal{S}$ sets $\mathsf{c}_{j,i_j} \leftarrow \mathcal{E}.\mathsf{Enc}(\kappa_j, \mathcal{DB}_j[i_j])$ because the ideal functionality outputs $\tilde{\mathsf{x}}_j = \mathcal{DB}_j[i_j]$ when κ_{i_j} is the i_j-th key in Λ. This perfectly matches the distribution of c_{j,i_j} in Real. On the other hand, for all j where $\tilde{\mathsf{x}}_j = \perp$, c_{j,i_j} (output by $\mathcal{S}'$) is computationally indistinguishable from a real encryption of $\mathcal{DB}[i_j]$, by the semantic-security of $\mathcal{E}$. With the help of a standard hybrid argument, it then follows that Real is computationally indistinguishable from the output of $\mathcal{S}$, assuming $\mathcal{E}$ satisfies the semantic security definition (Definition 4.3). □

7.2.2 Security of the Forward Secrecy Upgrade

Claim 6. Construction 3 in conjunction with KeyRefresh (Construction 4) instantiates the forward-secret authorization functionality specified in Functionality 3, assuming the existence of a PRF.

Proof. Let Update_R be an instance of the Update function (Definition 5.4) such that, given input Λ (and auxiliary information z), it generates Λ' completely independently of Λ. It disregards Λ and employs fresh randomness along with PIRAC.KeyGen to generate Λ'. Moreover, for the sake of simplicity, we will often refer to instantiating Update with KeyRefresh, which means defining Update by applying KeyRefresh to *every access key* of the access policy.

We first consider the case where $(\Lambda_1, \ldots, \Lambda_p)$ are generated independently of one another, that is, Update is instantiated using Update_R. Later, we will replace this assumption with $(\Lambda_1, \ldots, \Lambda_p)$ generated using KeyRefresh as the Update function.

In this case, we can compute sets $I_1, \ldots, I_q$, where $I_i \subseteq \{1, \ldots, p\}$ such that $\forall j, k \in I_i \ \Lambda_j = \Lambda_k$. We can then define the set $\mathcal{DB}_{I_i}$ and Λ_{I_i} to be subsets of $\{\mathcal{DB}_1, \ldots, \mathcal{DB}_p\}$ and $\{\Lambda_1, \ldots, \Lambda_p\}$, induced by indices in I_i.

By Claim 5, we have that there exists an efficient simulator $\mathcal{S}'$ when the server inputs are $(\mathcal{DB}_{I_i}, \Lambda_{I_i})$, because it holds that for all $j, k \in I_i$, $\Lambda_j = \Lambda_k$. (Notice that Functionality 3 only differs from Functionality 2 in that the authentication is performed using p, possibly unique, access policies $\Lambda_1, \ldots, \Lambda_p$.)

By the existence of $\mathcal{S}'$ for each subset of indices $I_1, \ldots, I_q$, and sequential composition [25, 50, 68], we get that there exists an efficient simulator $\mathcal{S}$ that is computationally indistinguishable to Real, which proves that Construction 3 instantiates Functionality 3 when access policies are generated independently of each other.

We now replace our starting assumption (each access policy is independent) to instead have the access policies generated according to KeyRefresh (Construction 4).

Hybrid 0. In this hybrid, we have $(\Lambda_1, \ldots, \Lambda_p)$ generated independently according to Update_R. Recall, Update_R uses PIRAC.KeyGen with fresh randomness to use independent access policies.

Hybrid 1. In this hybrid, we generate $(\Lambda_1, \ldots, \Lambda_p)$ using Update_F. Update_F works similar to Update_R but determines randomness of $\mathsf{PIRAC.KeyGen}$ using a PRF F. Specifically, $\Lambda_i[j] = \kappa_{i,j}$ for $i \in \{1, \ldots, p\}$, $j \in \{1, \ldots, n\}$ is generated by first computing $r_{i,j} \leftarrow F(k, i\|j)$, where $k \in \{0,1\}^\lambda$ is a random PRF key (coded in Update_F), and then running $\kappa_{i,j} \leftarrow \mathsf{PIRAC.KeyGen}(1^\lambda; r_{i,j}) = \mathcal{E}.\mathsf{KeyGen}(1^\lambda; r_{i,j})$. We now argue that Hybrid 0 is computationally indistinguishable from Hybrid 1.

Lemma 3. *Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be any semantically secure encryption scheme. The output of KeyGen computed with pseudorandom coins (determined by a PRF F) is computationally indistinguishable from the output of KeyGen on truly random coins.*

Lemma 3 follows from F being a PRF. Suppose, towards contradiction, that the output of KeyGen on pseudorandom coins was computationally distinguishable from the output of KeyGen on truly random coins, then there would also exist an efficient distinguisher for F, which contradicts the pseudorandomness of F.

Hybrid 2. In this hybrid, we generate $(\Lambda_1, \ldots, \Lambda_p)$ using $\mathsf{KeyRefresh}$ as the Update function. We prove that Hybrid 1 is computationally indistinguishable from Hybrid 2 in Lemma 4 by showing that F keyed with a pseudorandom key (i.e., the pseudorandom key output by F at epoch $t-1$) is still a PRF.

Putting things together. In sum, we have that Construction 3 when the access keys are generated according to $\mathsf{KeyRefresh}$ is equivalent to Construction 3 instantiated with p independent access policies. Then, because $\mathsf{KeyRefresh}$ is deterministic and efficiently computable "in the forward direction," we get the desired correctness and forward secrecy requirements. □

Lemma 4. *Fix security parameter $\lambda \in \mathbb{N}$ and let $k, p \leq \mathsf{poly}(\lambda)$. Let $F : \{0,1\}^\lambda \times \{0,1\}^k \to \{0,1\}^\lambda$ be a PRF. Then, it holds that the function $F' : \{0,1\}^\lambda \times \{0,1\}^{k \cdot p} \to \{0,1\}^\lambda$ defined as:*

$$F'(y_0, x_1, \ldots, x_p) = y_p,$$

where $y_0 \leftarrow_R \{0,1\}^\lambda$ and $y_{i+1} \leftarrow F(y_i, x_{i+1})$, $\forall i \in \{0, \ldots, p-1\}$, is also a PRF.

Proof. Suppose, towards contradiction, that there exists an efficient distinguisher $\mathcal{D}$ and non-negligible function ν such that:

$$\left|\Pr[\mathcal{D}^{F'(y_0,\cdot)}(1^\lambda)] - \Pr[\mathcal{D}^{R(\cdot)}(1^\lambda)]\right| \geq \nu(\lambda),$$

where R is a random function. Consider the hybrid distribution:

$$\mathcal{H}_i := \left\{ y_p : \begin{array}{l} y_i \leftarrow_R \{0,1\}^\lambda \\ y_{j+1} \leftarrow F(y_j, x_j) \end{array} \mid j \in \{i, \ldots, p-1\} \right\}.$$

Because $\mathcal{H}_0$ is identical to the distribution of $F'(y_0, \cdot)$ and $\mathcal{H}_p$ is identical to the distribution of a random function, it must be that $\mathcal{D}$ is also a distinguisher between $\mathcal{H}_i$ and $\mathcal{H}_{i+1}$, for some $i \in \{1, \ldots, p\}$.

We construct an efficient $\mathcal{A}$ such that:

$$\left|\Pr[\mathcal{A}^{F(y_0,\cdot)}(1^\lambda)] - \Pr[\mathcal{A}^{R(\cdot)}(1^\lambda)]\right| \geq \nu'(\lambda),$$

for some non-negligible function ν', contradicting the pseudorandomness of F. $\mathcal{A}$ proceeds as follows.

1: Receive as input 1^λ and run $\mathcal{D}(1^\lambda)$.

2: For each query $(x_1, \ldots, x_p)$ from $\mathcal{D}$,

- Query the oracle on x_i to get response y_{i+1}.
- Compute $y_{j+1} \leftarrow F(y_j, x_j)$ for all $j \in \{i+1, \ldots, p-1\}$.
- Respond with y_p.

3: Output as $\mathcal{D}$ does.

On the one hand, if $\mathcal{A}$ is given oracle access to F', then the response given to $\mathcal{D}$'s queries is identical to the distribution $\mathcal{H}_i$ (y_{i+1} is pseudorandom). On the other hand, if $\mathcal{A}$ is given oracle access to R, then the response given to $\mathcal{D}$'s queries is identical to the distribution $\mathcal{H}_{i+1}$ (y_{i+1} is truly random and y_{i+2} is pseudorandom). Therefore, $\mathcal{A}$ succeeds with the same probability as $\mathcal{D}$, contradicting the pseudorandomness of F. As such, F' must be a PRF if F is a PRF. □

7.2.3 Security of Breach Resilience Upgrade

Here, we analyze the security of our breach resilience transformation (Section 6.2.3).

Claim 7. The breach resilience modification described in Section 6.2.3 satisfies Definition 5.5, assuming the existence of semantically-secure re-randomizable public-key encryption.

Proof. First, we prove that our upgrade to breach resilience from Section 6.2.3 preserves the dynamic-database authorization property of Construction 3. To see this, observe that compared to Construction 3, we make two changes in Section 6.2.3: we (1) replace the symmetric-key encryption with public-key encryption and (2) replace re-encryption with re-randomization. We define the following hybrid distributions:

Hybrid 0. Identical to the real view of the user in Construction 3.

Hybrid 1. Replace the symmetric-key encryption scheme in Construction 3 with a re-randomizable public-key encryption scheme.

Hybrid 2. Replace Answer (as defined in Construction 3) with the re-randomizing variant described in Construction 5.

We now argue that Claim 7 follows from the proof of Claim 5. First, the proof of Claim 5 applies to Hybrid 1 because $\mathcal{S}'$ (the simulator from the semantic security definition) can be replaced with the simulator of the public-key encryption scheme. Second, we can safely replace the re-encryption in Hybrid 1 with re-randomization of ciphertexts because re-randomization is equivalent to a fresh encryption [27, 45, 53]. This proves that Hybrid 2 instantiates the dynamic-database authorization functionality.

Now, we show that our upgrade meets the breach resilience property specified in Definition 5.5. To do so, we must prove the existence of an efficient simulator $\mathcal{S}$ for the server state consisting of the database $\mathcal{DB}$ and the access policy Λ. This follows immediately from (1) $\mathcal{DB}$ being encrypted (and therefore efficiently simulatable) and (2) Λ consisting only of *public keys*, which can be efficiently generated using $\mathcal{E}.\mathsf{KeyGen}$ to match the real distribution. □

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 8

Evaluation

In this chapter, we evaluate Pirac from encryption (Section 6.2), when applied to single-server and multi-server PIR. We provide an evaluation of Pirac from PIR-by-keywords (Construction 2) in Appendix B.1, as existing keyword-private PIR-by-keywords schemes have limited practical efficiency or are restricted to two-server PIR.

Goals. The goals of our evaluation are to:

1. evaluate Pirac when instantiated using symmetric-key encryption and applied to single-server and multi-server PIR,
2. compare the performance of Pirac to existing approaches for access control in PIR, and
3. build the case for practical applications of Pirac.

Implementation and environment. We implement[1] the encryption and key-refresh in C, as these are performance-critical operations. We use Go and Python to run and analyze experiments. All experiments are run on an AWS `c5a.8xlarge` EC2 instance, with each experiment evaluated five times (unless otherwise stated) and the standard deviation across experiments is reported in-line. All evaluations are performed on a *single core*.

Organization. To gain a better understanding of the overhead that Pirac intro-

[1] Our implementation is open-source [4].

duces, in Section 8.1 we evaluate the cost of re-encryption, as required to satisfy dynamic-database authorization in Construction 3, and key-refresh, as required for the forward-secret authorization upgrade in Construction 4. Then, in Sections 8.2 and 8.3, we evaluate Pirac on state-of-the-art PIR schemes. In Section 8.4, we evaluate the overhead of introducing breach resilience into Pirac. In Section 8.5, we evaluate the periodic-update optimization from Section 6.3. Finally, in Section 8.6, we compare our performance to prior approaches for access control in PIR.

Table color coding. One of the key goals of our evaluation is to assess the overhead incurred by Pirac in server-side throughput compared to the baseline. Therefore, to help readers better interpret our results, we use color coding in our tables to represent the throughput overhead over the baseline. We use the following colors to represent different ranges of overhead: $> 10\times$, 3-10×, 2-3×, and $< 2\times$.

8.1 Re-encryption and Key Refreshing

We start by benchmarking re-encryption and key-refresh, as these inform the primary overheads in Pirac when using dynamic-database and forward-secret authorization, respectively.

Record Size	**Re-encryption (MB/s)**	**Key-refresh+Re-encryption (MB/s)**
16 B	809	11
64 B	1880	43
256 B	2636	164
1 KiB	2791	553
4 KiB	2170	1219
8 KiB	2200	1566
16 KiB	2200	1831

Table 8.1: Throughput of re-encryption and key-refresh in our Pirac implementation from symmetric encryption. The throughput for re-encryption plateaus with sufficiently large records due to the linear complexity of AES. When key-refresh and re-encryption are applied sequentially, throughput decreases initially but eventually amortizes with larger records because key-refresh is a fixed cost.

8.1.1 Re-encryption

To benchmark re-encryption, we measure the throughput for a range of record sizes; see Table 8.1. Re-encryption with AES delivers a throughput of approximately 2100-2700 MB/s when the record size becomes large enough. The throughput, in MB/s, does not change significantly when the record size increases, since the time to encrypt each additional byte remains the same.

8.1.2 Key-refresh

We set $\lambda = 128$ (i.e., 128-bit AES encryption keys). We instantiate the PRF F (for KeyRefresh in Construction 4) using AES to take advantage of `AES-NI` hardware acceleration. The average throughput, measured over 20 iterations, is 670.0 ± 2.4 *thousand key-refresh operations per second*. Since key-refresh throughput is independent of record size: key-refresh adds a *fixed* overhead per record to the processing of the PIR response in Pirac. Furthermore, we report the throughput achieved when key-refresh and re-encryption are executed sequentially, as required by the forward-secret authorization (Section 6.2.2). The results are reported in the third column of Table 8.1, with a maximum standard deviation of 0.6%. For smaller record sizes (i.e., $\leq$ 1 KiB), key-refresh is the throughput bottleneck, thereby determining the overall throughput. However, as the record size increases, the time required for key-refresh remains constant, while the time required for re-encryption increases linearly. Consequently, for larger record sizes, the throughput is predominantly determined by re-encryption.

8.1.3 Key-refresh (user)

We also benchmark key-refreshing for users who might run the Pirac protocol on devices with limited computation resources (e.g., phones). We use an AWS `t2.small` EC2 instance with 2 GiB RAM and 1vCPU. The average throughput (over 20 trials), is 469.3 ± 1.6 thousand key-refreshes per second.

8.2 Single-server PIR

Here, we evaluate Pirac on four modern single-server PIR schemes: SealPIR [11], FastPIR [7], Spiral [75], and SimplePIR [58]. We evaluate Pirac on databases of size $n = 2^{20}$ with record size (ℓ) varying from 16 B to 16 KiB.

Throughput (MB/s)		Record Size:	16 B	64 B	256 B	1 KiB	4 KiB	8 KiB	16 KiB
	Simple PIR	Baseline	10447	11024	1150	11545	11003	-	-
		w/ static-database auth.	10447	11024	1150	11545	11003	-	-
		w/ dynamic-database auth.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
		w/ forward-secret auth.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	Spiral PIR	Baseline	37	84	145	262	315	328	344
		w/ static-database auth.	37	84	145	262	315	328	344
		w/ dynamic-database auth.	36	81	137	239	275	285	298
		w/ forward-secret auth.	8	28	76	177	249	271	290
	Spiral Stream	Baseline	75	213	283	364	557	565	729
		w/ static-database auth.	75	213	283	364	557	565	729
		w/ dynamic-database auth.	69	191	256	321	444	449	548
		w/ forward-secret auth.	9	35	102	218	379	415	522
	Spiral Pack	Baseline	35	85	193	263	444	505	545
		w/ static-database auth.	35	85	193	263	444	505	545
		w/ dynamic-database auth.	34	82	180	240	368	410	437
		w/ forward-secret auth.	8	28	87	177	323	381	420
	Spiral Stream Pack	Baseline	346	539	871	1676	1957	1997	2160
		w/ static-database auth.	346	539	871	1676	1957	1997	2160
		w/ dynamic-database auth.	242	420	655	1042	1025	1043	1089
		w/ forward-secret auth.	10	39	135	410	740	878	992
	Seal PIR	Baseline	54	75	88	91	65	87	87
		w/ static-database auth.	54	75	88	91	65	87	87
		w/ dynamic-database auth.	50	72	85	88	63	83	84
		w/ forward-secret auth.	9	27	57	78	62	82	83
	Fast PIR	Baseline	104	178	205	216	217	213	213
		w/ static-database auth.	104	178	205	216	217	213	213
		w/ dynamic-database auth.	92	163	190	200	198	194	194
		w/ forward-secret auth.	10	34	91	155	184	188	190

Table 8.2: Throughput of Pirac from symmetric encryption when used with single-server PIR schemes on a database of size $n = 2^{20}$.

8.2.1 Static database and Static Access Policy

We first consider the static setting, where static-database authorization (Definition 5.2) suffices. In this scenario, the database is encrypted once and does not change, following strawman Pirac (Construction 1). This authorization model is ideally suited

to PIR schemes that assume static databases, such as SimplePIR [58] and other PIR schemes that have offline pre-processing requirements [32, 34, 84, 92]. Since the database needs to be encrypted just once (e.g., in the setup phase), Pirac *does not* introduce any processing overheads on the server, making the throughput equivalent to the underlying PIR scheme. We evaluate different state-of-art single-server PIR schemes with and without Pirac under static-database authorization. We report the mean throughput in Table 8.2 (the standard deviation was below 3.8%).

8.2.2 Dynamic Database and Static Access Policy

We now turn our attention to settings in which the database undergoes regular updates, which requires Pirac to satisfy dynamic-database authorization (Definition 5.3). PIR schemes in the pre-processing model designed for static databases (e.g., SimplePIR [58]) cannot handle dynamic databases by default [62, 70]. Spiral [75] is the state-of-the-art PIR scheme that does not require pre-processing and is specifically optimized for PIR in dynamic database settings. The throughput of Pirac when applied to the Spiral family, together with other common PIR schemes, is presented in Table 8.2. The standard deviation was below 3.5% across all throughputs.

We compare the throughput for SealPIR [11], FastPIR [7], and the Spiral family [75], under the dynamic-database authorization tier for varying record sizes in the first plot of Figure 8-1. For SealPIR [11], FastPIR [7], and most variants of the Spiral family, the re-encryption required in Pirac (which has a throughput of around 2200 MB/s) is significantly faster than computing the PIR response. As a result, computing the PIR answer is the primary bottleneck and determines the overall throughput of Pirac. This is evident in Figure 8-1 where dynamic-database authorization (dashed line) is very close to the baseline. We note that Pirac has an overhead of *at most* 1.3× in throughput compared to the baseline PIR schemes, with the exception of SpiralStreamPack where Pirac incurs an overhead of *at most* 2.0×. SpiralStreamPack is a variant of Spiral where the query is reused repeatedly and has a baseline throughput that is comparable to AES re-encryption, resulting in the 2.0× performance hit observed above. Furthermore, in Figure 8-1, we note that even though

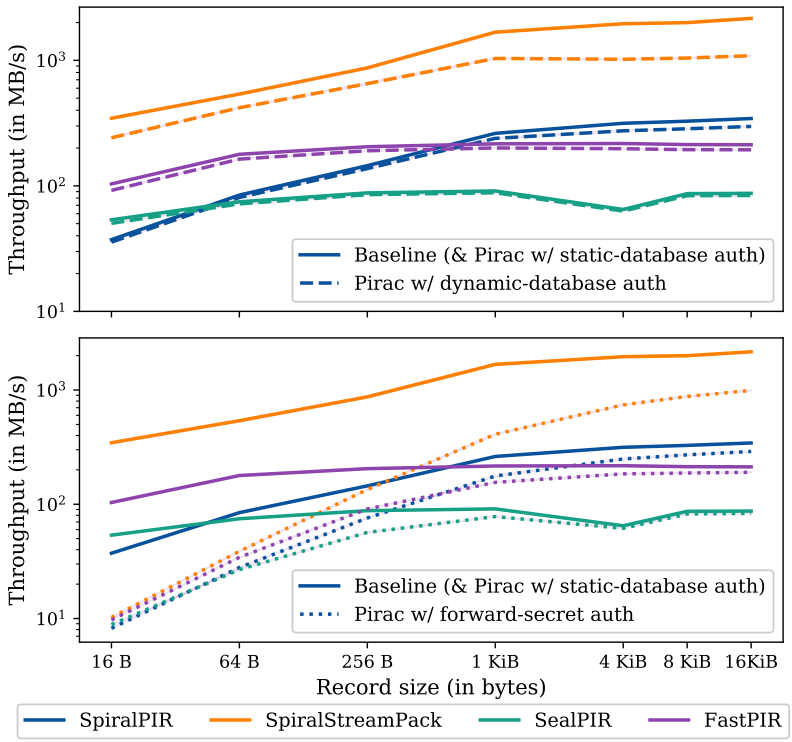

Figure 8-1: Throughput of Pirac when using single-server PIR schemes for different record sizes and considering the three authorization models: Static-database authorization (Construction 1; same throughput as baseline), with dynamic-database authorization (Construction 3), and with forward-secret authorization (Section 6.2.2).

dynamic-database authorization is very close to the baseline for small record sizes, it starts to diverge for larger record sizes. This is because re-encryption has a constant throughput, independent of record size ℓ, while the throughput of single-server PIR schemes (generally) increases with ℓ on our evaluation parameters.

8.2.3 Dynamic Database and Dynamic Access Policy

Finally, we evaluate Pirac in a setting where both the database and access policy change over time, which requires Pirac to satisfy forward-secret authorization (Definition 5.4). This requires the server to apply key-refresh (Construction 4) to each access key, prior to re-encrypting the database records. We report the mean throughput ($\pm 0.8\%$) of Pirac under this setting in Table 8.2.

As before, we plot the throughput for SealPIR [11], FastPIR [7], and the Spiral family [75], under forward-secret authorization, for varying record sizes in the bot-

tom plot of Figure 8-1. Pirac with forward-secret authorization results in *at most* 10.7× overhead for smaller records ($\leq$ 1 KiB) and 1.5× for larger records ($\geq$ 4 KiB) compared to baseline, when used with SealPIR [11], FastPIR [7], and most variants of Spiral [75]. For SpiralStreamPack, however, Pirac with forward-secret authorization results in an overhead of *at most* 33.9× for smaller records and 2.6× for larger records. Overhead drops significantly for larger record sizes as key-refresh adds a fixed cost per record (see Section 8.1.2) that amortizes as the records become larger. This is evident in Figure 8-1, where forward-secret authorization (dotted line) has a high overhead for small records but converges to the baseline throughput for large records.

8.3 Multi-server PIR

Here, we evaluate Pirac when applied to multi-server PIR schemes. For multi-server PIR, the XOR-based scheme of Chor et al. [30] is the fastest in terms of server-side throughput, at 4196 MB/s, using the open-source Percy++ [49] implementation. However, in practice, two-server PIR based on FSS [16, 17] or multi-server PIR extending Chor et al. [30] are used, as they provide better communication overheads and other features [37, 39, 42, 55, 56, 64, 80, 88, 89].

As mentioned in Chapter 3, Henry et al. [57] develop access control for Goldberg [48]'s multi-server PIR scheme. For FSS-based multi-server PIR, Servan-Schreiber et al. [82] develop access control using private access control lists (PACLs). Both PACLs and the technique of Henry et al. [57] achieve our notion of *forward-secret* authorization. However, these two approaches for access control do not apply to other multi-server PIR schemes (e.g., [16, 38]).

We evaluate these PIR schemes in the *two-server* setting, as this setting is often more concretely efficient in terms of throughput [56, 58]. The results for databases of size $n = 2^{20}$ and varying record sizes are presented in Table 8.3.

All the reported throughputs have a standard deviation of at most 3.3%. Using Percy++ [49], Pirac with forward-secret authorization results in a high overhead for

	Percy++ [49]		FSS-based PIR [16]		
Record Size	**Baseline (MB/s)**	**Pirac (MB/s)**	**Baseline (MB/s)**	**PACLs (MB/s)**	**Pirac (MB/s)**
16 B	2017	10	240	201	10
64 B	2404	41	716	614	40
256 B	2488	149	1303	1201	145
1 KiB	2537	446	1779	1739	422
4 KiB	2551	810	1996	1989	757
8 KiB	2557	957	2054	2054	889
16 KiB	2543	1048	2085	2085	975

Table 8.3: Pirac with forward-secret authorization (Section 6.2.2) applied to multi-server PIR schemes.

small records (because of the high fixed cost of key-refresh) but drops to 3.1× for records of size $\ell \geq 4$ KiB. Using FSS-based PIR, Pirac (also with forward-secret authorization) results in at most 2.6× overhead for the large record sizes ($\ell \geq 4$ KiB) over the baseline. However, for FSS-based PIR, PACLs prove to be a more performant choice for access control.

Unfortunately, we are unable to provide a complete head-to-head comparison between Pirac and Henry et al. [57] due to missing functionality in their open-source implementation [49] (see Appendix B). However, because they also use Percy++ as a baseline, we re-run Percy++ *in their evaluation setting* and extrapolate based on the relative overhead reported in their paper [57, Figure 1] (our methodology is described further in Appendix B). We find that, under their parameter settings, Pirac incurs an overhead of 1.9-2.2× whereas the access control scheme of Henry et al. [57] incurs an overhead of 3.5-10× over the Percy++ baseline.

8.4 Pirac with Breach Resilience

We benchmark Pirac with dynamic-database authorization and the breach resilience modification described in Section 6.2.3. Because the modification involves swapping out symmetric-key encryption with a re-randomizable public-key encryption scheme (Construction 5), the throughput of Pirac is bottlenecked by the re-randomization re-

quired per record. Re-randomization in Paillier [77] and ElGamal [44] has a throughput of 0.05 MB/s and 0.29 MB/s, respectively. In contrast, without breach resilience, the throughput of randomization using AES is approximately 2200 MB/s (Table 8.1). This illustrates the current impracticality of having breach resilience in Pirac with dynamic-database or forward-secret authorization.

8.5 Optimization: Periodic Updates

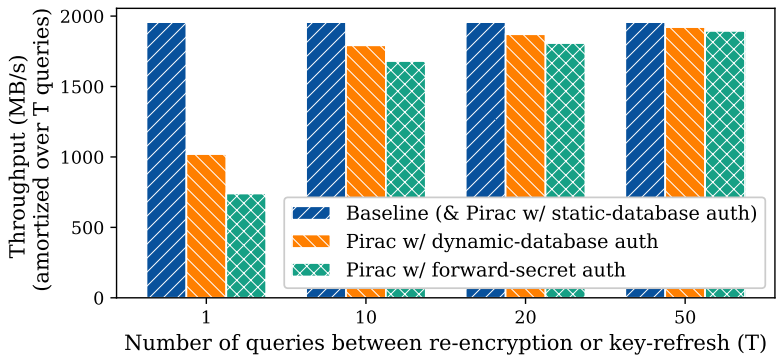

Figure 8-2: Overhead of Pirac (with dynamic-database and forward-secret authorization) applied to SpiralStreamPack using the optimization from Section 6.3 for different number of queries (T) between updates to the database and/or access policy. The baseline throughput is not impacted by periodic updates. Increasing T causes Pirac's performance to approach the baseline.

The results reported in Tables 8.2 and 8.3 and Figure 8-1 assume a *worst-case* scenario where the database and access policy change with *every* query. In real-world databases, the database and/or access policy are only going to change periodically, making the optimization described in Section 6.3 applicable to Pirac. If we assume that the server updates the database and access policy after every T queries, then re-encryption and/or key-refresh only needs to be applied between updates. We evaluate Pirac with different values of T to calculate the *amortized* server-side overhead of Pirac with dynamic-database and forward-secret authorization. We evaluate Pirac with SpiralStreamPack using $n = 2^{20}$ and 4 KiB record size. This combination of parameters results in the largest server-side overheads for Pirac among all large record ($\ell \geq 4$ KiB) sizes, see Table 8.2. We report the results of our "periodic update"

optimization in Figure 8-2. As before, Pirac with dynamic-database and forward-secret authorization has an overhead of 1.9× and 2.6×, respectively, with $T = 1$. However, the overhead of authorization drops significantly as T increases. These experiments highlight the performance benefit of batching database and access policy updates when using Pirac. We note that while Figure 8-2 reports the performance improvement of periodic updates in the single-server setting, similar improvements emerge in the multi-server setting.

8.6 Comparison to Prior Work

To the best of our knowledge, in the single-server setting, only Layouni [65] and Layouni et al. [66], consider access control for PIR. We estimate their concrete throughput as they do not evaluate their approaches. Specifically, given that they require a linear (in n) public key operations in addition to computing the SPIR query, the throughput is likely to be on-par with the throughput of Paillier-based PIR (0.05 MB/s). Consequently, these approaches to access control for single-server PIR are highly impractical.

PACLs [82] currently provide the best throughput for FSS-based PIR, but do not extend to single-server or other multi-server PIR schemes. Similarly, the access control technique of Henry et al. [57] only applies to Goldberg [48]'s multi-server PIR scheme. Additionally, their construction incurs high computational overheads (for both the user and the servers) and requires linear communication in the number of records. Compared to Henry et al. [57], Pirac reduces overhead on the servers by 1.6-4.5× and is applicable to other multi-server PIR schemes.

Chapter 9

Applications of Pirac

In many real-world settings, databases cater to many users and contain content that is exclusive to a subset of those users. Such settings demand both privacy and authorization: users want to privately retrieve records from this database, while the database wants to ensure that no malicious user can retrieve content they do not have permission to access. Applications with this twofold problem include: password breach lookups [62, 85], (priced) digital media downloads [56, 57], and friend discovery services [61, 73]. We briefly survey how Pirac can be used in these applications.

9.1 Private Password Breach Lookups

Account breach alerting services [2, 59] allow users to check their usernames against a database of exposed credentials. Most breach alerting services only allow users to check for username matches to avoid revealing sensitive information. However, this approach can result in false positive breach alerts for passwords users have already changed and, moreover, it does not provide actionable information to the user about which password was leaked. Thomas et al. [86] provide a comprehensive discussion on the challenges related to this problem.

A better approach is to query using both the username and password, but this reveals the sensitive (and possibly uncompromised) passwords to the checkup server. Because breached credentials may remain valid for multiple years [85] after being

compromised, revealing the passwords to the checkup server is a security risk.

PIR can be applied to provide user privacy while Pirac can provide access control to prevent malicious actors from stealing credentials. Using Pirac, the breach alerting server sets up a database where each record entry corresponds to a unique username (u) and password (p) pair and includes details about the breaches this credential is involved in. The *access key* then consists of a hash of the credential $H(\mathsf{u}\|\mathsf{p})$ using a suitable randomness extractor H [87]. Because the database is dynamic, Pirac with *dynamic-database authorization* should be used. However, because access to credentials does not change, the access policy is static and therefore forward-secret authorization is not required.

For instance, some databases of exposed credentials consists of 4 billion unique usernames and passwords [86]. In prior approaches, the server partitions the database randomly (with the help of a hash function), assigning a subset of credentials to each partition. Thomas et al. [86] make the partitions consist of roughly 500,000 credentials, which translates to a 500 MB partition database, assuming 1 KiB records. Pirac (Construction 3) using SpiralPIR [75] would require 2.1 seconds of server-side processing (Table 8.2 in Appendix B). The processing time can be further reduced by using the optimization from Section 6.3. This makes the response time comparable to the work of Thomas et al. [86], where they achieve a median end-to-end (including network latency) response time of 8.5 seconds. (Thomas et al. [86] do **not** use PIR citing the lack of database access control, even though using PIR would improve user privacy guarantees in their system [86, Section 2.5].)

9.2 Private Purchased Content Retrieval

Many online services have databases that contain exclusive or premium content accessible only to a subset of users. For example, Bloomberg Terminal [1] provides proprietary market data to financial analysts. As the service provider, Bloomberg needs to ensure that users can only access the content they have paid for. On the flip side, analysts are inherently secretive and may want to keep their retrieval pat-

terns private from Bloomberg, as their accesses could reveal information about their investment strategies.

With Pirac's *forward-secret authorization*, Bloomberg can provide query privacy to their customers while ensuring that *only* paying users can access the premium content. In this example, the access policy might change monthly when users renew or upgrade their subscriptions, making the periodic update optimization from Section 6.3 applicable. Additionally, the server can determine the frequency of database updates and apply periodic updates to records as well.

For instance, consider a database of stock prices for 500 companies tracked by the S&P 500 [5]. Each record may store the historical stock prices over the past three decades and would be roughly 100KiB in size [72]. Using Pirac with SpiralPIR would require 0.18 seconds of server time, suggesting the practicality of our scheme.

Similarly, in many other contexts, such as private media retrieval [56], purchases can be privately accessed from a remote database using PIR while Pirac can guarantee that only authorized (paying) users can download the digital content. Because Pirac applies to all PIR schemes, it can be integrated directly with Percy++ [48], as done in Popcorn [56] for movie streaming, or Spiral, which was used to privately query Wikipedia [74].

9.3 Private Friend Discovery

Social media applications often use our contact books to recommend us other friends who are using the platform [61, 73]. Friend discovery can easily be made private using PIR, however, this runs the risk of malicious users scraping the discovery service for user contact information, which leads to a severe privacy concern. With Pirac, we can generate an access key for each user based on information that is only available to a user's "real" friends, for example, a phone number and personal email address. Only people that have those two pieces of contact information in their address book would be able to discover the corresponding user on the platform. For instance, consider a database of 40 million users (approximately the number of monthly active

users on Signal as of January 2022 [36]) with 16-byte records. We can also assume that the database is static (e.g., the database is replaced once a week), allowing us to use a PIR-with-preprocessing scheme like SimplePIR. Then Pirac, combined with SimplePIR, enables efficient authenticated friend discovery, requiring only 60 *milliseconds* of server-side processing per query.

Chapter 10

Conclusion

We present Pirac, a framework for adding access control to PIR in a black-box way using lightweight cryptographic tools. We evaluate our open-source implementation on a variety of PIR schemes and demonstrate the concrete efficiency of Pirac in different settings. Finally, we show that Pirac enables new applications of PIR and has exciting potential use-cases in real-world systems.

Future Work. Currently, the performance of Pirac using constant-weight PIR is poor compared to our evaluation of Pirac using symmetric-key encryption (Chapter 8 and Appendix B.1), where the throughput achieved using state-of-the-art PIR schemes is roughly two orders of magnitude higher. To address this disparity, we hope to see new keyword-private PIR schemes that can bridge this performance gap.

In addition, our current construction of breach resilience is slow and primarily of theoretical interest. We acknowledge the need for improved constructions that can be applied in real-life scenarios. Therefore, we leave it as future work to devise better breach-resilient constructions that are both efficient and practical.

Furthermore, we aspire to deploy Pirac in various applications mentioned in Chapter 9 to bring us closer to achieving privacy and anonymity. By exploring different use cases, we can gain valuable insights and refine the implementation of Pirac for real-world scenarios.

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] All products | bloomberg professional services. URL https://www.bloomberg.com/professional/all-products/. Accessed May 2023.

[2] Enzoic. URL https://www.enzoic.com/. Accessed May 2023.

[3] Mirrored Percy++ source code. https://github.com/pawangoyal137/Percyxx. Accessed May 2023.

[4] Pirac source code. https://github.com/pawangoyal137/PIRAC.

[5] S&P500, 2023. URL https://en.wikipedia.org/wiki/S%26P_500. Accessed May 2023.

[6] Masayuki Abe, Jan Camenisch, Maria Dubovitskaya, and Ryo Nishimaki. Universally composable adaptive oblivious transfer (with access control) from standard assumptions. In *Proceedings of the 2013 ACM workshop on Digital identity management*, pages 1–12, 2013.

[7] Ishtiyaque Ahmad, Yuntian Yang, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. Addra: Metadata-private voice communication over fully untrusted infrastructure. In *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI2021, July 14-16, 2021*, 2021.

[8] Bill Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology—EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques*

Innsbruck, Austria, May 6–10, 2001 Proceedings 20, pages 119–135. Springer, 2001.

[9] Ghous Amjad, Seny Kamara, and Tarik Moataz. Breach-Resistant Structured Encryption. *Proceedings on Privacy Enhancing Technologies*, 1:245–265, 2019.

[10] Sebastian Angel and Srinath Setty. Unobservable Communication over Fully Untrusted Infrastructure. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, page 551–569, USA, 2016. USENIX Association. ISBN 9781931971331.

[11] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE symposium on security and privacy (SP)*, pages 962–979. IEEE, 2018.

[12] Michael Backes, Aniket Kate, Matteo Maffei, and Kim Pecina. Obliviad: Provably secure and practical online behavioral advertising. In *2012 IEEE Symposium on Security and Privacy*, pages 257–271. IEEE, 2012.

[13] D Elliott Bell and Leonard J La Padula. Secure computer system: Unified exposition and multics interpretation. Technical report, MITRE CORP BEDFORD MA, 1976.

[14] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Draft 0.5*, 2020.

[15] Nikita Borisov, George Danezis, and Ian Goldberg. DP5: A private presence service. *Proceedings on Privacy Enhancing Technologies*, 2015(2):4–24, 2015.

[16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 337–367. Springer, 2015.

[17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1292–1303, 2016.

[18] Zvika Brakerski. Fundamentals of Fully Homomorphic Encryption-A Survey. In *Electron. Colloquium Comput. Complex.*, volume 25, page 125, 2018.

[19] David FC Brewer and Michael J Nash. The Chinese Wall Security Policy. In *IEEE symposium on security and privacy*, volume 1989, page 206. Oakland, 1989.

[20] Paul Bunn, Eyal Kushilevitz, and Rafail Ostrovsky. CNF-FSS and its applications. In *Public-Key Cryptography–PKC 2022: 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8–11, 2022, Proceedings, Part I*, pages 283–314. Springer, 2022.

[21] Jan Camenisch, Maria Dubovitskaya, and Gregory Neven. Oblivious transfer with access control. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 131–140, 2009.

[22] Jan Camenisch, Maria Dubovitskaya, and Gregory Neven. Unlinkable priced oblivious transfer with rechargeable wallets. In *Financial Cryptography and Data Security: 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers 14*, pages 66–81. Springer, 2010.

[23] Jan Camenisch, Maria Dubovitskaya, Gregory Neven, and Gregory M Zaverucha. Oblivious transfer with hidden access control policies. In *Public Key Cryptography–PKC 2011: 14th International Conference on Practice and Theory in Public Key Cryptography, Taormina, Italy, March 6-9, 2011. Proceedings 14*, pages 192–209. Springer, 2011.

[24] Jan Camenisch, Maria Dubovitskaya, Robert R Enderlein, and Gregory Neven. Oblivious Transfer with Hidden Access Control from Attribute-Based Encryption. In *SCN*, volume 7485, pages 559–579. Springer, 2012.

[25] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13:143–202, 2000.

[26] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22*, pages 255–271. Springer, 2003.

[27] Ran Canetti, Hugo Krawczyk, and Jesper B Nielsen. Relaxing chosen-ciphertext security. In *Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23*, pages 565–582. Springer, 2003.

[28] Benny Chor and Niv Gilboa. Computationally Private Information Retrieval (Extended Abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 304–313, New York, NY, USA, 1997. Association for Computing Machinery. ISBN 0897918886.

[29] Benny Chor, Niv Gilboa, and Moni Naor. *Private information retrieval by keywords*. Citeseer, 1997.

[30] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private Information Retrieval. *J. ACM*, 45(6):965–981, nov 1998. ISSN 0004-5411.

[31] David D Clark and David R Wilson. A comparison of commercial and military computer security policies. In *1987 IEEE Symposium on Security and Privacy*, pages 184–184. IEEE, 1987.

[32] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 44–75. Springer, 2020.

[33] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE, 2015.

[34] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. Single-server private information retrieval with sublinear amortized time. In *Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part II*, pages 3–33. Springer, 2022.

[35] Scott Coull, Matthew Green, and Susan Hohenberger. Controlling access to an oblivious database using stateful anonymous credentials. In *Public Key Cryptography–PKC 2009: 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings 12*, pages 501–520. Springer, 2009.

[36] David Curry. Signal Revenue & Usage Statistics, 2023. URL https://www.businessofapps.com/data/signal-statistics/. Accessed May 2023.

[37] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. DORY: An encrypted search system with distributed trust. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pages 1101–1119, 2020.

[38] Daniel Demmler, Amir Herzberg, and Thomas Schneider. RAID-PIR: practical multi-server PIR. In *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*, pages 45–56, 2014.

[39] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: Scaling Private Contact Discovery. *Proceedings on Privacy Enhancing Technologies*, 4: 159–178, 2018.

[40] Giovanni Di Crescenzo, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan.

Conditional oblivious transfer and timed-release encryption. In *EuroCrypt*, volume 99, pages 74–89. Springer, 1999.

[41] Whitfield Diffie, Paul C Van Oorschot, and Michael J Wiener. Authentication and authenticated key exchanges. *Designs, Codes and cryptography*, 2(2):107–125, 1992.

[42] Jack Doerner and abhi shelat. Scaling ORAM for secure computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 523–535, 2017.

[43] Nico Döttling and Jesko Dujmovic. Maliciously Circuit-Private FHE from Information-Theoretic Principles. In *3rd Conference on Information-Theoretic Cryptography*, 2022.

[44] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[45] Antonio Faonio, Dario Fiore, Javier Herranz, and Carla Ràfols. Structure-preserving and re-randomizable RCCA-secure public key encryption and its applications. In *Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part III*, pages 159–190. Springer, 2019.

[46] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[47] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting Data Privacy in Private Information Retrieval Schemes. *Journal of Computer and System Sciences*, 60(3):592–629, Jun 2000. ISSN 0022-0000.

[48] Ian Goldberg. Improving the robustness of private information retrieval. In *2007 IEEE Symposium on Security and Privacy (SP'07)*, pages 131–148. IEEE, 2007.

[49] Goldberg, Ian and Devet, Casey and Lueks, Wouter and Yang, Anna and Hendry, Paul and Henry, Ryan. Percy++ source code. https://percy.sourceforge.net, 2014. Accessed May 2023.

[50] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge University Press, 2009.

[51] Oded Goldreich et al. Foundations of cryptography–a primer. *Foundations and Trends® in Theoretical Computer Science*, 1(1):1–116, 2005.

[52] Matthew Green, Watson Ladd, and Ian Miers. A protocol for privately reporting ad impressions at scale. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1591–1601, 2016.

[53] Jens Groth. Rerandomizable and replayable adaptive chosen ciphertext attack secure cryptosystems. In *TCC*, volume 2951, pages 152–170. Springer, 2004.

[54] Christoph G Günther. An identity-based key-exchange protocol. In *Advances in Cryptology—EUROCRYPT'89: Workshop on the Theory and Application of Cryptographic Techniques Houthalen, Belgium, April 10–13, 1989 Proceedings 8*, pages 29–37. Springer, 1990.

[55] Daniel Günther, Maurice Heymann, Benny Pinkas, and Thomas Schneider. GPU-accelerated PIR with Client-Independent Preprocessing for Large-Scale Applications. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1759–1776, 2022.

[56] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with Popcorn. In *13th USENIX Symposium on Networked Systems Design and Implementation NSDI 16)*, pages 91–107, 2016.

[57] Ryan Henry, Femi Olumofin, and Ian Goldberg. Practical PIR for electronic commerce. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 677–690, 2011.

[58] Alexandra Henzinger, Matthew M Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. *Cryptology ePrint Archive*, 2022.

[59] Troy Hunt. Have I been pwned: Check if your email has been compromised in a data breach. URL https://haveibeenpwned.com/. Accessed May 2023.

[60] Ari Juels. Targeted advertising... and privacy too. In *Cryptographers' Track at the RSA Conference*, pages 408–424. Springer, 2001.

[61] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1447–1464, 2019.

[62] Dmitry Kogan and Henry Corrigan-Gibbs. Private Blocklist Lookups with Checklist. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 875–892. USENIX Association, August 2021. ISBN 978-1-939133-24-3.

[63] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings 38th annual symposium on foundations of computer science*, pages 364–373. IEEE, 1997.

[64] Maximilian Lam, Jeff Johnson, Wenjie Xiong, Kiwan Maeng, Udit Gupta, Minsoo Rhu, Hsien-Hsin S Lee, Vijay Janapa Reddi, Gu-Yeon Wei, David Brooks, et al. GPU-based Private Information Retrieval for On-Device Machine Learning Inference. *arXiv preprint arXiv:2301.10904*, 2023.

[65] Mohamed Layouni. Accredited symmetrically private information retrieval. In *Advances in Information and Computer Security: Second International Work-*

shop on Security, IWSEC 2007, Nara, Japan, October 29-31, 2007. Proceedings 2, pages 262–277. Springer, 2007.

[66] Mohamed Layouni, Maki Yoshida, and Shingo Okamura. Efficient multi-authorizer accredited symmetrically private information retrieval. In *Information and Communications Security: 10th International Conference, ICICS 2008 Birmingham, UK, October 20-22, 2008 Proceedings 10*, pages 387–402. Springer, 2008.

[67] Benoît Libert, San Ling, Fabrice Mouhartem, Khoa Nguyen, and Huaxiong Wang. Adaptive oblivious transfer with access control from lattice assumptions. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 533–563. Springer, 2017.

[68] Yehuda Lindell. How to simulate it–a tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, pages 277–346, 2017.

[69] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In *Information Security: 8th International Conference, ISC 2005, Singapore, September 20-23, 2005. Proceedings 8*, pages 314–328. Springer, 2005.

[70] Yiping Ma, Ke Zhong, Tal Rabin, and Sebastian Angel. Incremental Offline/Online PIR. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1741–1758, 2022.

[71] Rasoul Akhavan Mahdavi and Florian Kerschbaum. Constant-weight PIR: Single-round Keyword PIR via Constant-weight Equality Operators. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1723–1740, 2022.

[72] Boris Marjanovic. Huge Stock Market Dataset, 2017. URL https:

//www.kaggle.com/datasets/borismarjanovic/price-volume-data-for-all-us-stocks-etfs. Accessed May 2023.

[73] Moxie Marlinspike. Technology preview: Private contact discovery for Signal. *Signal Messenger*, 2017.

[74] Samir Menon. Spiral demo. URL https://spiralwiki.com/. Accessed May 2023.

[75] Samir Jordan Menon and David J Wu. Spiral: Fast, high-rate single-server PIR via FHE composition. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 930–947. IEEE, 2022.

[76] Rafail Ostrovsky, Anat Paskin-Cherniavsky, and Beni Paskin-Cherniavsky. Maliciously circuit-private FHE. In *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34*, pages 536–553. Springer, 2014.

[77] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*, pages 223–238. Springer, 1999.

[78] Sarvar Patel, Joon Young Seo, and Kevin Yeo. Don't be Dense: Efficient Keyword PIR for Sparse Databases. In *32th USENIX Security Symposium (USENIX Security 23)*, 2023.

[79] Michael O Rabin. How to exchange secrets with oblivious transfer. *Cryptology ePrint Archive*, 2005.

[80] Len Sassaman, Bram Cohen, and Nick Mathewson. The Pynchon gate: A secure method of pseudonymous mail retrieval. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 1–9, 2005.

[81] Sacha Servan-Schreiber, Kyle Hogan, and Srinivas Devadas. AdVeil: A Private Targeted Advertising Ecosystem. *Cryptology ePrint Archive*, 2021.

[82] Sacha Servan-Schreiber, Simon Beyzerov, Eli Yablon, and Hyojae Park. Private Access Control for Function Secret Sharing. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1257–1276. IEEE Computer Society, 2022.

[83] Sacha Servan-Schreiber, Simon Langowski, and Srinivas Devadas. Private approximate nearest neighbor search with sublinear communication. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 911–929. IEEE, 2022.

[84] Elaine Shi, Waqar Aqeel, Balakrishnan Chandrasekaran, and Bruce Maggs. Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part IV 41*, pages 641–669. Springer, 2021.

[85] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, Angelika Moscicki, et al. Data breaches, phishing, or malware? Understanding the risks of stolen credentials. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 1421–1434, 2017.

[86] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, et al. Protecting accounts from credential stuffing with password breach alerting. In *USENIX Security Symposium*, pages 1556–1571, 2019.

[87] L. Trevisan and S. Vadhan. Extracting Randomness from Samplable Distributions. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS '00, page 32, USA, 2000. IEEE Computer Society. ISBN 0769508502.

[88] Adithya Vadapalli, Fattaneh Bayatbabolghani, and Ryan Henry. You May Also Like... Privacy: Recommendation Systems Meet PIR. *Proceedings on Privacy Enhancing Technologies*, 2021(4):30–53, 2021.

[89] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical Private Queries on Public Data. In *NSDI*, pages 299–313, 2017.

[90] Ye Zhang, Man Ho Au, Duncan S Wong, Qiong Huang, Nikos Mamoulis, David W Cheung, and Siu-Ming Yiu. Oblivious Transfer with Access Control: Realizing Disjunction without Duplication. *Pairing*, 6487:96–115, 2010.

[91] Ke Zhong, Yiping Ma, and Sebastian Angel. Ibex: Privacy-preserving ad conversion tracking and bidding. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 3223–3237, 2022.

[92] Mingxun Zhou, Andrew Park, Elaine Shi, and Wenting Zheng. Piano: Extremely Simple, Single-Server PIR with Sublinear Server Computation. *Cryptology ePrint Archive*, 2023.

Appendix A

Keyword-private PIR-by-keywords

Here, we describe two PIR-by-keyword schemes and argue why they satisfy keyword privacy (Definition 6.1). For convenience, we define the function f_{eq}:

$$f_{eq}(\mathcal{DB}_{\mathsf{W}}, w) = \begin{cases} \mathcal{DB}_{\mathsf{W}}[w] & \text{if } w \in \mathsf{W}, \\ \perp & \text{otherwise.} \end{cases}$$

Single-server PIR-by-keywords. Mahdavi and Kerschbaum [71] construct a *single-server* PIR-by-keywords scheme that matches the syntax of Definition 4.1. Their construction is equivalent to evaluating f_{eq} over an encryption of the keyword w using FHE (Definition A.1). The server outputs $\mathsf{c} \leftarrow \mathsf{FHE.Eval}(f_{eq}, \mathcal{DB}_{\mathsf{W}}, \mathsf{q})$, where f_{eq} is represented as a circuit. We now argue that when the FHE scheme used is *circuit-private* [43, 46, 76] (see also Definition A.2), the scheme of Mahdavi and Kerschbaum [71] satisfies the keyword privacy definition Definition 6.1.

Definition A.1 (Fully Homomorphic Encryption [18, 46])**.** Fix security parameter $\lambda \in \mathbb{N}$. An FHE scheme consists of four efficient (possibly randomized) algorithms $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$. Takes as input a security parameter λ. Outputs a new pair of public and secret keys.
- $\mathsf{Enc}(\mathsf{pk}, m) \to c$. Takes as input a public key pk and message m. Outputs a

ciphertext c. For convenience, we will sometimes denote an FHE encryption of m under public key pk as $\langle m \rangle_{\mathsf{pk}}$.

- $\mathsf{Eval}(\mathsf{pk}, f, c_1 \dots c_k) \to c_f$. Takes as input a public key pk, the description of an efficiently computable k-variate function f, and k ciphertexts $c_1 \dots c_k$. Outputs evaluated ciphertext c_f.
- $\mathsf{Dec}(\mathsf{sk}, c) \to m$. Takes as input a secret key sk and ciphertext c. Outputs a plaintext message m.

The above must satisfy *correctness* and *semantic security*. We elaborate here on the correctness property only. Correctness of an FHE scheme holds if for all efficiently computable k-variate functions f, $\mathsf{Dec}(\mathsf{sk}, \mathsf{Eval}(\mathsf{pk}, f, \langle m_1 \rangle_{\mathsf{pk}}, \dots \langle m_k \rangle_{\mathsf{pk}})) = f(m_1, \dots, m_k)$. We point to the survey of Brakerski [18] for more details on FHE.

Definition A.2 (Maliciously-secure Circuit-private FHE [43, 76])**.** Let $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a fully-homomorphic encryption scheme. Fix $k = \mathsf{poly}(\lambda)$, where λ is the security parameter of FHE. For all public keys pk and k-variate, efficiently computable circuits $\mathcal{C}$ computing a function f, and for all input ciphertexts $c_1 \dots c_k$ encrypting circuits inputs $x_1, \dots, x_k$, there exists an efficient simulator $\mathcal{S}$ such that $\mathsf{FHE.Eval}(\mathsf{pk}, \mathcal{C}, c_1 \dots c_k) \approx_c \mathcal{S}(\mathsf{pk}, \mathcal{C}(x_1, \dots, x_k), \mathsf{leak}(\mathcal{C}))$, where leak captures possible auxiliary information about the circuit $\mathcal{C}$ that is revealed in the scheme (e.g., the size or structure of $\mathcal{C}$).

Claim 8. Let $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a circuit private fully-homomorphic encryption scheme (Definition A.2). The PIR scheme of Mahdavi and Kerschbaum [71] satisfies Definition 6.1 when instantiated with FHE.

Proof. We must construct an efficient simulator $\mathcal{S}$ for the view of the adversary consisting of c (recall Definition 6.1). Let $z \in \{0,1\}^*$ consist of the public key pk of the FHE scheme and a description of the circuit computing f_{eq}. $\mathcal{S}$ proceeds as follows.

1: Receive as input $(w, \tilde{\mathsf{x}}, z)$.
2: Parse $z = (\mathsf{pk}, \mathsf{leak}(\mathcal{C}))$.
3: Output as $\mathcal{S}'(\mathsf{pk}, \tilde{\mathsf{x}}, \mathsf{leak}(\mathcal{C}))$.

Notice that $\mathcal{S}$ obtains $\tilde{\mathsf{x}} := f_{eq}(\mathcal{DB}_{\mathsf{W}}, w) = \mathcal{C}(\mathcal{DB}_{\mathsf{W}}, w)$. As such, the input to $\mathcal{S}'$ is correctly distributed. Therefore, the PIR answer c computed according to $\mathsf{FHE.Eval}$, is efficiently simulatable, satisfying Definition 6.1. Finally, provided that $\mathsf{leak}(\mathcal{C})$ consists of the description of the circuit computing f_{eq} and remains independent of W and the database $\mathcal{DB}$, it does not reveal any information (beyond size) about the database or keywords to the user. In conclusion, we have that a circuit-private FHE scheme makes the PIR-by-keywords construction of [71] satisfy keyword privacy. □

Two-server PIR-by-keywords. Boyle et al. [16] construct a *two-server* PIR-by-keywords scheme that matches the syntax of Definition 4.1. Their construction is equivalent to evaluating f_{eq} over secret shares of w. Moreover, their construction satisfies *function privacy*: the output shares of $\mathsf{PIR.Answer}$ can be efficiently simulated given only $f_{eq}(\mathcal{DB}_{\mathsf{W}}, w)$ [16]. This automatically makes their PIR-by-keyword scheme based on FSS satisfy keyword-privacy following the same template in the proof of Claim 8.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix B

Additional Evaluation

B.1 Evaluation of Pirac from PIR-by-keywords

In this section, we evaluate Pirac from PIR-by-keyword (Construction 2) when instantiated with single-server and two-server PIR-by-keyword schemes. We evaluate Pirac using the two keyword-private PIR-by-keywords schemes described in Appendix A on a database of size $n = 2^{10}$ with record size ℓ varying from 20 KiB to 320 KiB. We choose these parameters as the scheme of Mahdavi and Kerschbaum [71] is designed for large records (and is only practical on smaller databases).

B.1.1 Single-server Pirac from keyword-private PIR-by-keywords

We first evaluate the server throughput for Pirac using the PIR-by-keywords scheme of Mahdavi and Kerschbaum [71]. As mentioned in Appendix A, Mahdavi and Kerschbaum [71] use the equality operator to evaluate f_{eq} under FHE. However, instead of naively evaluating f_{eq} (as described in Appendix A), they propose a faster method for evaluating f_{eq} in the PIR-by-keywords setting use constant-weight codes. They map the keywords of the database to a new space of constant Hamming-weight codewords, which allows them to evaluate f_{eq} more efficiently.

We evaluate the PIR-by-keywords scheme of Mahdavi and Kerschbaum [71] under two parameter settings. Our baseline evaluation uses 10-bit keywords, just enough

to assign a unique keyword to every record, with a Hamming weight of 2 (see [71] for details on how Hamming weight affects performance). For Pirac from PIR-by-keywords, we use *48-bit keywords* with a Hamming weight of 5. Recall that the keyword domain dictates the security we obtain in Pirac (i.e., with 48-bit keywords, the probability of guessing an access key for a particular index is 2^{-48}). Because an adversary cannot mount an offline guessing attack, 48-bit keywords is a reasonable security level: guessing one keyword in a database of size $n = 2^{10}$ would require issuing over 250 billion queries (recall that the probability of guessing *any* keyword is $2^{-\lambda+\lceil \log n \rceil}$). Even with the relatively small keyword domain, the throughput of Pirac applied to Mahdavi and Kerschbaum [71] is not practical: ranging from 0.23 to 3.35 MB/s (see Table B.1), with a standard deviation of at most 0.2%. However, we note that Pirac only adds a 3-4× overhead compared to the baseline in this setting.

The low throughput currently makes Pirac instantiated from PIR-by-keywords in the single-server setting too inefficient for many real-world settings. Moreover, we note that this throughput (both for baseline and Pirac) functions as an *upper bound* on actual performance since larger databases would require a bigger keyword domain and ensuring keyword-privacy in Mahdavi and Kerschbaum [71] requires additional computational overheads, as explained in Appendix A.

Record Size	**Baseline (MB/s)**	**Pirac (MB/s)**
20KiB	0.95	0.23
40 KiB	1.85	0.46
80 KiB	3.51	0.91
160 KiB	6.34	1.76
320 KiB	10.66	3.35

Table B.1: Pirac from single-server PIR-by-keywords [71].

Record Size	**Baseline (MB/s)**	**Pirac (MB/s)**
20 KiB	1880	1629
40 KiB	1857	1710
80 KiB	1843	1761
160 KiB	1877	1886
320 KiB	1962	1787

Table B.2: Pirac from two-server PIR-by-keywords [16].

B.1.2 Two-server Pirac from keyword-private PIR-by-keywords

Next, we evaluate the server throughput of Pirac from PIR-by-keywords using the two-server FSS-based PIR-by-keywords scheme [16]. We note that only **two**-server

constructions of FSS-based PIR-by-keywords constructions are known. To match the single-server setting, we evaluate Pirac on a database of size $n = 2^{10}$ with 48-bit keywords and compare with the baseline PIR-by-keywords with 10-bit keywords. The result for varying record sizes is presented in Table B.2, with a standard deviation of at most 11.8%. Pirac from two-server PIR-by-keywords results in a modest overhead of at *at most* 1.2× compared to the baseline, and is concretely practical in this setting.

B.2 Public-key Re-randomization

Linearly-homomorphic encryption schemes achieve the ciphertext re-randomizability property required for our breach resilience upgrade. We benchmark the linearly-homomorphic Paillier [77] and ElGamal [44] schemes, which to the best of our knowledge, are the most efficient re-randomizable public-key encryption schemes.

We evaluate Paillier in Go using a 2048-bit modulus, which results in a 2048-bit message space. We benchmark the throughput of Paillier by extrapolating the time for one exponentiation and one multiplication[1] and measure an average throughput of 0.05 MB/s over 1,000 iterations.

To benchmark ElGamal, we use the P256 elliptic curve group (available in Go `crypto/elliptic` package), which has a 256-bit message space. Re-randomization in ElGamal has an average throughput of 0.29 MB/s. Our code for benchmarking Paillier and ElGamal can be found in our GitHub repository [4].

B.3 Evaluation of Henry et al. [57]

As mentioned in Chapter 8, the Percy++ source code [49] is more than ten years old and does not directly compile with modern machines. We created a mirror of the original source code with some bug fixes [3]. While we were able to reproduce the results from the original Percy++ paper [48], we were not able to reproduce the results of Henry et al. [57] for two reasons. First, the scheme of Henry et al. [57]

[1] Note that re-randomization in Paillier encryption requires one exponentiation and one multiplication per record entry.

needs to compile the Percy++ PIR scheme into a (symmetric) PIR scheme using a library (`PolyCommit`) that to the best of our knowledge is no longer available on the internet. Additionally, Henry et al. [57] mention that they apply optimizations to the Percy++ scheme that improve the baseline performance by a factor of 40-60×, but do not provide additional details or the code for these optimizations. Consequently, we benchmark the original Percy++ code (after making it compatible with the newer machine) and use the Henry et al. [57] reported overhead relative to Percy++ [57, Figure 1] to compare with the overhead of Pirac.