Practical Privacy Enhancement of Web Services

by

Sumit Paul

A thesis
submitted to the University of Ottawa
in partial fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

© Sumit Paul, Ottawa, Canada, 2026

Abstract

The rise of web services has made our lives more convenient, yet it has also raised significant privacy concerns. Users grant web service providers access to their personal data, as part of obtaining service. Service providers may track access patterns for web services (e.g. for frequently-browsed items). As a result, protecting users' personal information and their access patterns has become a critical privacy issue, which this thesis aims to address.

We acknowledge that enhancing user privacy has a cost for service providers, whether financial or in the form of reduced features, which could impede the adoption of Privacy-Enhancing Technologies. Our research seeks to reduce these costs while offering additional benefits to service providers. We believe that by providing such incentives, we can encourage service providers to adopt these solutions in practice.

We provide access pattern protection for both private and public database accesses. There is a well-established lower limit on the amount of work needed for protecting access patterns in private database contexts, which significantly slows down the access process. We demonstrate a way to reduce the latency while accessing private databases. When it comes to public databases, however, protecting client access patterns can increase server overhead significantly, but we also show how to reduce server overhead to unprecedented levels.

We also outline different ways of preserving the privacy of personal client data. Instead of relying solely on traditional cryptographic methods, we leverage the growing prevalence of trusted hardware in server environments. By utilizing this technology, we have developed effective solutions for protecting personal data, both in a single location and during the necessary sharing of the data.

Acknowledgements

I would like to take a moment to express my sincere gratitude to my supervisor, Dr. David Knox. His steadfast support and encouragement have served as a guiding light during challenging times, and I truly believe that this thesis would not have reached its completion without his invaluable assistance.

I want to express my heartfelt gratitude to my family for their unwavering love and support. Their sacrifices and steadfast belief in me have been my anchors throughout this journey. A massive thank you to my friends who encouraged me to take this path in the first place. I would like to give special recognition to Ritin Kumar Das, not only for his constant motivation but also for developing a Java package, which significantly assisted me in evaluating the performance of a comparison candidate. I want to thank Sathvika Balumuri for carrying out experiments with comparative PIR schemes.

Finally, I extend my gratitude to all the teachers in my life, including my parents, who have shaped me in profound ways since my childhood to date. Each lesson learned has been a stepping stone toward my growth, and I am forever grateful.

Dedication

This thesis is dedicated to the millions of underprivileged children worldwide whose potential remains untapped due to inadequate access to essential nutrition and education.

Table of Contents

List of Tables xii

List of Figures xiii

Terms and Abbreviations xvi

1 Introduction 1

1.1 A Web-service Scenario Example 1

1.2 Targeted Concerns for Web Services' Privacy 3

1.2.1 Leakage of Access Patterns 3

1.2.2 Leakage of Personal Data 4

1.2.3 Deterrence for Privacy Enhancement in Web Service 4

1.3 Our contributions 5

1.3.1 Trust Model: Neither Service Providers nor Users Can Be Trusted . 7

1.3.2 Publications and Software 8

2 A Low-latency ORAM to Protect Access Patterns in a Private Database **10**

2.1 Motivation: Protection of Access Patterns inside Private Databases with Low Latency 10

2.2 Related Work 12

2.2.1 Early Research 12

2.2.2 Towards Reducing Overhead to the Theoretical Limit 12
2.2.3 Situation-Specific Overhead Reduction 13
2.2.4 Utilizing Server's Computation Capability to Reduce Latency . . . 14
2.3 *RouterORAM*: An ORAM scheme with $O(1)$-latency 15
2.3.1 Targeted Problem Statement 16
2.3.2 Background: PathORAM 16
2.3.3 Overview of *RouterORAM* 17
2.4 *RouterORAM* Details 19
2.4.1 Client Storage 19
2.4.2 Outsourced Data 20
2.4.3 Server Storage 20
2.4.4 *RouterORAM* Initialization 20
2.4.5 Accessing a Block in *RouterORAM* 21
2.4.6 Higher Layer Operations of *RouterORAM* and their Latencies . . . 27
2.4.7 Background Routing in *RouterORAM* 27
2.5 Privacy Analysis 32
2.6 Analysis of Block Access Failures 36
2.6.1 Effect of Access Frequency on Failure 36
2.6.2 Effect of Failure on Latency 37
2.7 Discussion 39
2.7.1 Summary of Theoretical Contribution 39
2.7.2 Practical Considerations 40
2.7.3 Other Applications of *RouterORAM* 42

3 Efficient PIR schemes to Protect Access Patterns in a Public Database 43
3.1 Motivation: Efficient Protection of Access Patterns in Public Databases . . 43
3.2 *Basic Scheme:* Reducing Server Overhead to $O(\sqrt{\mathsf{N}})$ 44
3.2.1 Related Work for *Basic Scheme*, for reducing server overhead 44

3.2.2 Overview: A Three Server Architecture with Shuffling and Homomorphic Evaluations . . . 47
3.2.3 Storage Details . . . 50
3.2.4 One-time Initialization . . . 51
3.2.5 (Re-)encrypt and (re-)shuffle entire $\widetilde{\mathbb{D}}$ . . . 51
3.2.6 Fetch Block Ciphertext from the Server Storage . . . 54
3.2.7 Return Block Plaintext to the PIR-client . . . 57
3.2.8 Asymptotic Overhead Analysis . . . 58
3.3 *DP-Variant Scheme*: Extending to a Differentially Private Scheme with Even Lower Overhead . . . 60
3.3.1 Related work Regarding Differentially Private Schemes . . . 61
3.3.2 Overview: Reducing Overhead by Allowing Limited re-touching . . . 62
3.3.3 Differences in the Storage Details . . . 63
3.3.4 Differences in the Shuffling Process . . . 64
3.3.5 Differences in the Fetching Process . . . 66
3.3.6 Effect on Overheads . . . 68
3.3.7 Privacy-Overhead Tradeoff Analysis . . . 69
3.4 *DPF-Variant Scheme*: Enhancing the Practicality by Utilizing DPF . . . 77
3.4.1 Related Work Regarding utilization of DPF in PIR and DORAM . . . 77
3.4.2 Overview: Performing DPF-based Shelter Searching . . . 79
3.4.3 One-time Initialization . . . 82
3.4.4 Shuffled Database, Mask Database, Shelter and Tag Details . . . 82
3.4.5 Per Epoch Operations . . . 85
3.4.6 Determination of the shelter-tag to be searched (Phase 1 of Figure 3.8): 87
3.4.7 Oblivious Shelter Search (Phase 2): . . . 89
3.4.8 Oblivious tag selection for the shuffled databases (Phase 3): . . . 91
3.4.9 Fetch-combine shares and then select response (Phase 4-6): . . . 92
3.4.10 Update shelter tags (Phase 7): . . . 93

3.4.11 Delivering Response and Append to the Shelter(Phase 8): 94
3.4.12 Privacy Analysis 95
3.4.13 Overhead Analysis 105
3.5 Benefits of Our PIR Schemes 107
3.5.1 Practicality: Considering both Server and Client Overhead 108
3.5.2 Maintaining Aggregated Statistics 108
3.5.3 Detection of Malicious Behavior 110

4 Protecting Personal Data in a single location **111**
4.1 Motivation: Enforcing Privacy Policies and a Data Expiry Mechanism . . . 111
4.1.1 Trusted Execution Environment: A Practical Alternative 112
4.2 Related Work 113
4.2.1 Privacy Policy Enforcement 113
4.2.2 Data Expiry 115
4.2.3 Protection against Rollback attack on TEEs 116
4.3 *ROT*: A Retention and Operation Limitation Framework utilizing TEE . . 117
4.3.1 Background: Trusted Execution Environment 117
4.3.2 Overview: Process data within TEE and Leverage Blockchain Against Rollback Attack 118
4.4 *ROT*: Details 120
4.4.1 Threat Model and Assumptions 120
4.4.2 Utilization of Blockchain Properties 121
4.4.3 Structure of Data Owner's Personal Data 122
4.4.4 Structure of the Computing Enclaves 123
4.4.5 Details of Protocol Stages 124
4.5 Privacy Analysis 131
4.5.1 Formal Representation of *ROT* 131
4.5.2 Privacy and Security Goals 134

4.5.3 Ideal Functionality . . . 134
4.5.4 UC-proof . . . 137
4.6 Implementation and Performance Analysis . . . 137

5 Bidirectional Privacy Preservation During Data-Sharing 141
5.1 Motivation: Bidirectional Privacy and Practicality of Privacy Policy Enforcement . . . 141
5.2 Related Work . . . 143
5.2.1 Ensuring Data Owner's Privacy During Data Sharing . . . 143
5.2.2 Reducing Auditing Burden . . . 144
5.3 *BPPM:* A framework for Bidirectional Privacy preservation with Practicality during Multi-layer data sharing . . . 145
5.3.1 Overview: Aggregated Policy, Limited Sharing and Reusing of Audited Code . . . 146
5.3.2 Data Sharing Stakeholders and Flow of Personal Data . . . 147
5.4 *BPPM*:Details . . . 148
5.4.1 Threat model and Assumption . . . 148
5.4.2 Guarantee of Privacy Preservation . . . 149
5.4.3 Piracy-free Source-code Sharing . . . 149
5.4.4 Structures of Personal Data and Processing Consent . . . 151
5.4.5 Details of Protocol Stages . . . 154
5.5 Privacy Analysis . . . 160
5.5.1 Formal Representation of *BPPM* . . . 160
5.5.2 Privacy and Security Goals . . . 164
5.5.3 Ideal Functionality . . . 165
5.5.4 UC-proof . . . 167
5.6 Implementation and Performance Analysis . . . 167
5.6.1 Implementation Details . . . 167
5.6.2 Performance Analysis . . . 168

5.7 Cost-Benefit Analysis . . . 172
5.7.1 Costs and Benefits for Service Users . . . 172
5.7.2 Costs and Benefits for Service providers . . . 173

6 Conclusions and Future Work **175**
6.1 Regarding *RouterORAM* . . . 175
6.2 Regarding PIR-schemes . . . 176
6.3 Regarding *ROT* . . . 178
6.4 Regarding *BPPM* . . . 179
6.5 Overall Future work and Conclusion . . . 179

References **181**

A Notation **205**

B Distributed Point Function **210**

C Homomorphic Properties of El-Gamal Encryption **211**

D Cuckoo Hash with Stash **213**

E Trusted Execution Environment **214**
E.1 Enclave . . . 214
E.2 Attestation . . . 217
E.2.1 Attested-TLS . . . 218
E.3 Sealing . . . 219
E.4 Formal modeling . . . 220

F Universal Composability **222**
F.1 Background concept: Simulation-based Security 222
F.2 Additional concepts in UC-framework: Environment and Ideal Functionality 223
F.3 Real-world protocol execution 223
F.4 Ideal-world protocol execution 225

G UC-proof of *ROT* **226**
G.1 Sim Design - UsageApproval Stage: 226
G.2 Sim Design - DataProcurement Stage: 229
G.3 Sim Design - DataUsage Stage: 232

H UC-proof of *BPPM* **235**
H.1 Sim Design - Setup Stage: 235
H.2 Sim Design - SendOrigData Stage: 236
H.3 Sim Design - ForwardData Stage: 239
H.4 Sim Design - DataUsage Stage: 241

List of Tables

1.1 Adversarial Summary Table 7

2.1 Comparisons 40

3.1 Secret Allocations to the Three Servers 49

3.2 Summary of the overheads 58

3.3 Summary of the overheads of our *DP-Variant Scheme* 68

3.4 δ-values of our scheme ($\mathsf{N} = \mathsf{1.68}$ billion) 76

3.5 The inputs and outputs of all parties while processing the ith PIR-request . 96

3.6 Empirical analysis ($\mathsf{N} = \mathsf{1677721600}$) 107

3.7 Asymptotic Overhead Comparisons 109

5.1 Compliance Cost Comparison Across Regulations (assuming 1 CAD=0.73 USD) 174

A.1 Notation regarding *RouterORAM* 206

A.2 Notation regarding the PIR schemes 206

A.3 Notation regarding *ROT* 208

A.4 Notation regarding *BPPM* 209

List of Figures

1.1 Summary of our contributions 5

2.1 Example execution scenario of PathORAM protocol 17
2.2 Server Tree and Routing 28
2.3 Effect of Access Frequency on Failures 37
2.4 Number of effective bucket touches 38

3.1 Overview of *Basic Scheme* 48
3.2 Shuffling Process Sequence Diagram 53
3.3 Fetch Block Ciphertext Sequence Diagram 55
3.4 Return Block Plaintext Sequence Diagram 57
3.5 Shuffling Process of Differentially Private Scheme 64
3.6 Fetching Process of Differentially Private Scheme 66
3.7 Privacy-Overhead Tradeoff Analysis 75
3.8 High-Level Flow of Request Processing 81
3.9 Per Epoch Operations 85
3.10 Shelter-tag determination 87
3.11 Obliviously search shelter 89
3.12 Select shuffled database fetch tag 91
3.13 Fetch-combine and select 92
3.14 Shelter update 93

3.15 Returning Plaintext Response 95

4.1 Top-Level Overview of *ROT* 118
4.2 Structure of the enclave 123
4.3 Sequence of Usage Approval Stage 126
4.4 Sequence of Data Procurement Stage 128
4.5 Sequence of Data Usage Stage 130
4.6 Formal representation of the *ROT* protocol 132
4.7 Enclave program of *ROT* 133
4.8 $\mathcal{F}_{ROT}$: Ideal Functionality of *ROT* 135
4.9 Number of required bytes for sending data 138
4.10 Required time to perform computation 139
4.11 Number of bytes transferred while performing computation 140

5.1 Top-level overview of *BPPM* 147
5.2 Code reusage scenario 150
5.3 Data-structures PC and D and their relationship 152
5.4 Detailed sequence of "Setup" stage 154
5.5 Detailed sequence of "Send Original Data" stage 156
5.6 Detailed sequence of "Process" stage 157
5.7 Detailed sequence of "Forward" stage 159
5.8 Formal representation of the *BPPM* protocol 162
5.9 Base enclave program of *BPPM* 163
5.10 $\mathcal{F}_{BPPM}$: Ideal Functionality of *BPPM* 165
5.11 Communication cost comparison 169
5.12 Processing time comparison 170
5.13 Redactable Signature Performance Comparsion 171

E.1 Memory layout of *enclave* 215

E.2 Ideal functionality for G_{att} [1] 220

F.1 Real-world protocol execution in UC-Framework [2, Chapter 23.5.1] 224
F.2 Ideal-world protocol execution in UC-Framework [2, Chapter 23.5.2] . . . 225

Terms and Abbreviations

epoch The time period after which the contents of the PIR databases are shuffled 50

shelter A cache area in the PIR scheme to store past responses 49

BPPM Bidirectional Privacy Preservation in Web Services - Scheme 145

COED computation on encrypted data 137

DoS Denial of Service 8

DP Differential Privacy 5, 60

DPF Distributed Point Function 5, 77

FE Functional Encryption 137

FHE Fully Homomorphic Encryption 15

MPC Multi Party Computation 137

ORAM Oblivious Random Access Machine 5

PET Privacy Enhancing Technology 3

PIR Private Information Retrieval 5

ROT Retention and Operation Limitation using TEE - Scheme 117

TEE Trusted Execution Environment 112

UC Universal Composability 131

VM Virtual Machine 137

Chapter 1

Introduction

When accessing web services - whether it is an e-commerce website, a patient portal for booking appointments with your doctor, or a media streaming platform - users are often required to provide various pieces of personal information to the service provider. Some of this data is essential; for instance, it is necessary to supply payment information in order to complete a purchase on an e-commerce site. The concern is, when a user shares their data with another party, they irrevocably relinquish control over that data.

Another subtle yet significant privacy concern associated with the use of web services is that servers can observe user activities within their systems. For instance, the servers can track which doctor they frequently schedule appointments with or the types of movies a user typically watches. These access patterns may inadvertently reveal undesirable private aspects of users' lives to the servers.

1.1 A Web-service Scenario Example

Bob runs a website or an app[1] that makes it easy for customers to schedule medical appointments. Alice, who is feeling under the weather with a fever, wants to book an appointment with a doctor. To start the process, she enters her postal code on Bob's website, which helps her find available options nearby, and she decides to book an appointment with Chuck's clinic. Then she checks her personal calendar (Outlook, Google Calendar, etc.) to identify a convenient time for her visit.

[1] Telus Health is a real example that University of Ottawa students use to book medical appointments.

Finally, she completes the appointment booking on the selected date by providing her personal details on Bob's website, such as her name, date of birth and phone number. Alice doesn't leave Bob's website immediately. With some concerns about her heart health, she spends some time browsing through the available specialist doctors. However, she hasn't made up her mind yet, so she eventually closes the website browser.

The first privacy concern here is the fact that Bob now possesses a significant amount of personal information about Alice, including her date of birth, the date of her appointment, the clinic she visits and her postal code. Bob might have a privacy policy and may already have undergone the privacy audit process to comply with privacy regulations. Nevertheless, this does not guarantee that Bob is actually behaving honestly all the time. Bob can still misuse Alice's personal data covertly for ulterior motives. For instance, he might choose to retain this sensitive information indefinitely or secretly sell details to an advertising firm, which was not disclosed in his privacy policy.

The second problem is that Bob may be honest but curious. Bob may observe what Alice is doing, after landing on his website. He notices that, even if Alice did not book any appointment with a heart specialist, she was still curious about the details of them. Bob may infer that Alice might be suffering from heart-related problems as well. Bob can use the inferred idea for some benign purposes, like sending promotional offers regarding a blood cholesterol test. Alternatively, he can sell that information about Alice's browsing behavior on the black market. Social engineering attackers (e.g. spearphishers) might have a value for this kind of information.

Even if we assume that Bob is completely honest, some of Alice's personal data is shared with Chuck's clinic. There's a risk that this clinic might not be as trustworthy as Bob and could misuse the information that they receive. Additionally, there's another, more subtle issue at play, even assuming all of these parties are honest.

If Bob has hosted his application and its accompanying database on a cloud server, the cloud service provider, Dave, has the capability to monitor the database's access patterns, which enables them to glean insights into how the app is utilizing data, based on its access location, rather than its specific contents. For example, whenever Alice books an appointment, Dave can deduce that someone has made a reservation at Chuck's clinic. This deduction can be made by observing that a series of accesses have been made to identical database locations (over time by the same application or even by using "fake" client experiments executed by Dave). This issue also arises when Alice accesses her personal calendar, as the calendar service provider can see the time slots when she is busy.

Alarmingly, this concern remains valid even if all the actual data stored in the database is encrypted.

1.2 Targeted Concerns for Web Services' Privacy

In our research, we are focusing on two essential privacy concerns associated with web services: the privacy of personal data and the privacy of access patterns. We recognize that, in addition to these fundamental issues, there are further obstacles that may impede the widespread adoption of Privacy Enhancing Technologies (PETs). Our goal is to address some of these challenges.

1.2.1 Leakage of Access Patterns

The access pattern can reveal significant personal information about a user, depending on the context [3, 4]. In our earlier example, we observed that Alice's access pattern indicated her worries about her heart health. Alarmingly, no data encryption method can address this issue. The real challenge lies in finding ways to "encrypt" our access patterns themselves.

When accessing web services, two primary types of access situations can arise. The first occurs when a single client retrieves data stored in a private database maintained by an untrusted server. For instance, consider our experiences with Google Drive, OneDrive, or Google Calendar, as discussed in our previous example. This private database scenario can also manifest for the service providers themselves when the services are hosted in the cloud. In the earlier example, the cloud storage provider, Dave, could potentially gather insights into the usage patterns of Bob's app, thereby compromising Bob's privacy, as well as the privacy of his users.

The second type of access pertains to public databases, where multiple users retrieve data from a shared source. Notably, the server has knowledge of the database's content. This scenario is prevalent in various contexts, such as when users stream videos from media platforms or browse the catalog of an e-commerce website. In these cases, it is straightforward for the server to monitor which users are accessing specific content within the database.

Our goal is to address the issue of access pattern leakage in both of these scenarios. It is worth noting that there is another related privacy concern in this context: identifying who accesses the database or web service. However, our research does not concentrate on that particular problem.

1.2.2 Leakage of Personal Data

The privacy of personal data shared by users is equally important - if not more so. While data encryption is essential for protecting information, it does not resolve every issue. Specifically, in order to process personal data, service providers need access to an unencrypted version of it. If these providers have malicious intentions, they could secretly copy this unencrypted data. Once this happens, the malicious service provider could endlessly process the data for nefarious purposes, and restoring privacy becomes almost impossible. Although there are mechanisms that enable computation on ciphertexts without requiring decryption, finding effective ways to mitigate the risk of malicious processing remains a significant challenge.

The situation becomes increasingly concerning when a service provider requires users to share their personal data with multiple third parties. These third parties may, in turn, share that information even further. For service users, keeping track of where their data ends up can be very challenging. Even those who can follow their data trail may find it nearly impossible to evaluate the trustworthiness of the parties handling their information. In this complex web of data sharing, if any involved party has malicious intent, the privacy of personal data could be irreparably compromised. Furthermore, these data users may operate in different jurisdictions, which complicates matters even more since the same privacy clause can have entirely different meanings depending on the governing laws.

1.2.3 Deterrence for Privacy Enhancement in Web Service

Producing solutions to the aforementioned problems involves not only technical obstacles but also practical challenges that can complicate the adoption of these solutions in real-world scenarios. Generally speaking, PETs aim to safeguard user privacy; however, they often put service providers at a disadvantage. Consider the issue of access patterns: when these patterns are kept confidential, service providers lose the ability to analyze service usage trends and enhance their services.

Not only do PETs pose challenges for service providers, but privacy regulations themselves also create considerable difficulties. For example, privacy regulations usually require service providers to disclose third party details in their privacy policies, which can put the providers' valuable trade secrets at risk. Additionally, to comply with these regulations, service providers often bear significant costs related to auditing, training, and other compliance measures. They invest time and resources into these activities without expecting immediate benefits. It is no surprise that many service providers, while not actually opposed to privacy, feel less inclined to adopt PETs and may even resist them in practice.

1.3 Our contributions

Often, the solutions for protecting personal data and access patterns are slow and practically unusable. Previous researchers have made notable strides in addressing these challenges and have laid a solid foundation for further advancements. In our study, we aim to push these boundaries further, with a strong focus on practicality guiding our efforts. Please refer to Figure 1.1, which summarizes the contributions of our research.

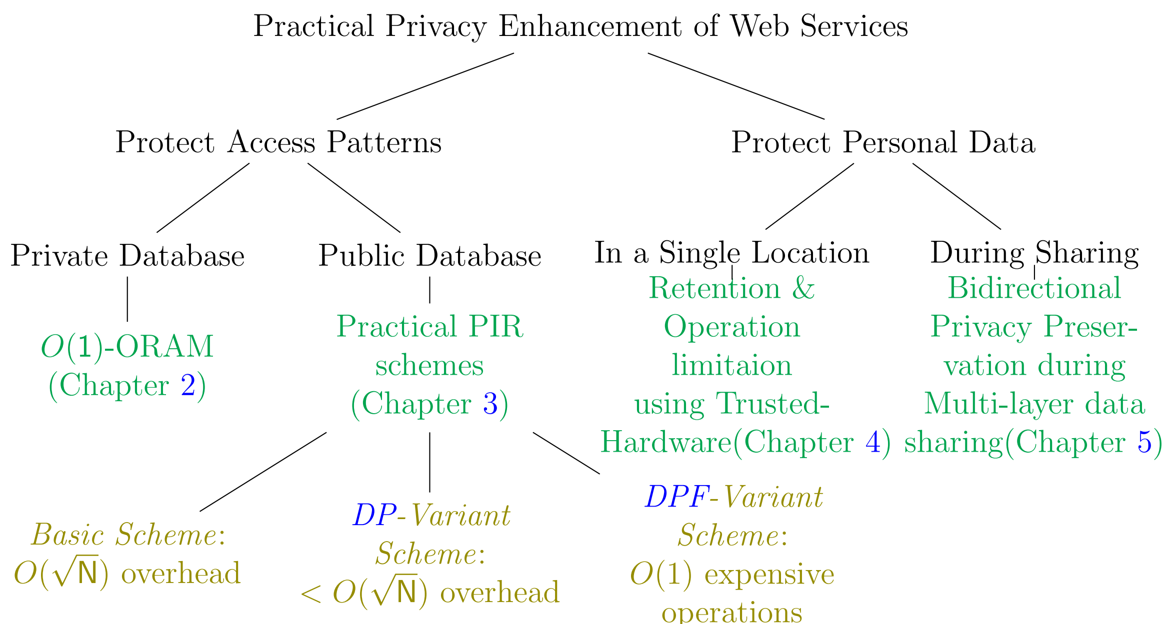


Figure 1.1: Summary of our contributions

We focused on protecting access patterns in both private and public database environments. In the domain of private databases, there exists a compelling area of research known as Oblivious RAM (ORAM). This method is notable for its inherent overhead of $O(\log N)$ per access operation, where N denotes the total number of items in the database. This overhead implies that, as databases continue to grow, an increasingly common trend - users may experience longer waits to access the desired item. In our research, we have designed a solution that significantly reduces this latency to $O(1)$. Details of our solution can be found in Chapter 2.

In the realm of public databases, Private Information Retrieval (PIR) is a significant area of research. However, the main concern regarding PIR is its $O(N)$ overhead. In

Chapter 3 of this thesis, we design three different schemes to tackle this issue. Our first scheme reduced the overhead to $O(\sqrt{\mathsf{N}})$. Building on this foundation, our second solution introduces a differentially private scheme that further reduces overhead.

Despite the asymptotic efficiency of our first scheme, we encountered a challenge: while it reduces the overhead to $O(\sqrt{\mathsf{N}})$, the individual operations are quite demanding. This means that the advantages of our approach might only become apparent when dealing with extremely large databases. To address this, we uncovered yet another scheme that maintains the $O(\sqrt{\mathsf{N}})$ overhead while drastically reducing the number of expensive operations to $O(1)$.

Next, we turn our attention to the privacy of personal data. Instead of relying solely on traditional cryptographic techniques, we leverage the growing adoption of trusted hardware in server environments. By utilizing this technology, we have developed an efficient solution in Chapter 4 that enforces data expiry and guarantees that personal data is processed only for approved purposes, all while never disclosing the plaintext of the personal data.

In Chapter 5, we build upon our previous solution by exploring the complexities of sharing personal data among multiple parties. This improved approach not only guarantees robust privacy protections for service users but also demonstrates how service providers can reduce privacy-related costs while potentially reaping tangible benefits. This dual advantage may serve as a crucial incentive for service providers to adopt this PET in real-world applications.

In addition to lowering overhead expenses, a key motivation behind all of our suggested solutions is to alleviate other practical challenges (such as financial costs and usability) and make our solutions more appealing for regular use. In this context, besides safeguarding users' privacy, we also seek to offer advantages to service providers, who usually bear a considerable share of the expenses related to privacy measures.

In some situations, the service providers have to lose several key features due to the adoption of PETs. In our research, we also kept that aspect in mind and designed the PETs to continue providing those features to service providers while enhancing privacy for service users. We anticipate that these improvements will motivate both service users and providers to incorporate these PETs into practical applications.

1.3.1 Trust Model: Neither Service Providers nor Users Can Be Trusted

Whenever a service provider holds a database, the database either contains private but encrypted content or publicly known content. Consequently, the service provider can either learn nothing about the plaintext content, or already knows all of it before any interaction with service users. Hence, becoming malicious in this context does not make much sense. The only thing they can do is observe or remain curious about how service users interact. In fact, in a public database scenario, they can also launch fake clients to learn more.

However, whenever service providers process users' personal data and provide services based on it, they have real motives to act maliciously and deviate from the established protocol. Even when a service provider outlines its practices in privacy policies or asserts that it passes privacy audits, we cannot be certain of their long-term adherence to those guidelines. A successful audit does not ensure that they will consistently uphold their policies over time.

Table 1.1: Adversarial Summary Table

Chapter	**Service Provider**	**Service User**	**Comments**
Chapter 2	Honest-but-curious	Honest	Service user fetches its own outsourced data. Since all the operations are performed by the service user and the data remains encrypted, the service provider can only observe. On the other hand, the service user uses the server for their own purposes only and has nothing to learn about it.
Chapter 3	Honest-but-curious	Honest-but-curious	In a multi-client scenario, some of the clients can be fake and launched by the service provider itself.
Chapter 4	Malicious	Malicious	Both might have motives to take advantage by deviating from the established protocol.
Chapter 5	Malicious	Malicious	Similar motivation factors for both the parties.

On the other hand, we do not completely trust the service users either. Alarmingly, in the realm of web services, some individuals may resort to deceptive practices to gain unfair

advantages. For example, consider a situation where someone uses a fraudulent credit card to make a purchase. Moreover, in environments with multiple service users, the likelihood of encountering impersonators rises significantly. In fact, a third party may control these fake users to extract private information from legitimate users. Table: 1.1 summarizes the adversarial capabilities of the involved parties in each chapter.

In our research, we set aside the potential motivations of service users who might disrupt a service provider's functionality. For instance, we don't assume that users would engage in launching denial of service (DoS) attacks. Overall our focus is on safeguarding against honest-but-curious servers that might seek to pry into the personal data or actions of service users. At the same time, we remain vigilant to ensure that protective measures must not inadvertently offer any advantages to malicious users who may seek to exploit our solutions for unauthorized access to services.

1.3.2 Publications and Software

- All of the work presented in Chapter 2 is published by Springer in the conference proceedings for the Symposium on Foundations & Practice of Security, 2025:
  - Sumit Kumar Paul and D. A. Knox. 2025. *RouterORAM*: An $O(1)$-Latency and Client-Work ORAM. *Foundations and Practice of Security.* Springer, 2025 [5].
  - We have developed a simulator to figure out the long-term behavior of this ORAM scheme, which can be found at: `https://github.com/sumitkumarpaul/oram`.
- In Chapter 3, we present three distinct PIR schemes. We have prepared a manuscript focusing solely on the *DPF-Variant Scheme*, which has been submitted to IEEE Transactions on Privacy. Additionally, we plan to combine *Basic Scheme* and the *DP-Variant Scheme* for submission to the Wiley Security and Privacy Journal.
  - We have also developed a functional PIR scheme based on *DPF-Variant Scheme*, available at: `https://github.com/sumitkumarpaul/pir`.
- The solution presented in Chapter 4 is published in the proceedings of the IEEE International Conference on Cyber Security and Resilience (CSR), 2024:
  - Sumit Kumar Paul and D. A. Knox. 2024. ROT: Retention and Operation Limitation Using TEE *IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE, 2024 [6].

  - We have implemented a usable solution in C/C++, which is available at: https://github.com/sumitkumarpaul/data_rotting.

- The solution presented in Chapter 5 is published in the Special Issue Emerging Trends in Network Security and Applied Cryptography of the MDPI Computers Journal, 2025:

  - Sumit Kumar Paul and D. A. Knox. 2025. Bidirectional Privacy Preservation in Web Services. *Computers*. MDPI, 2025 [7].

  - We have also implemented this solution in C/C++, and it is available at: https://github.com/sumitkumarpaul/bidirectional_privacy.

Chapter 2

A Low-latency ORAM to Protect Access Patterns in a Private Database

First, we concentrate on protecting the client's access patterns when they fetch their personal data from a remote database. Our objective is to obscure this access pattern from an honest-but-curious server that controls the database. We propose an Oblivious RAM (ORAM) solution that achieves $O(1)$ latency and client work. In this chapter, we will explore this approach, evaluate its associated overhead, and analyze its usefulness in everyday scenarios.

2.1 Motivation: Protection of Access Patterns inside Private Databases with Low Latency

The utilization of personal cloud storage services, such as Google Drive, iCloud, and OneDrive, has become an increasingly important aspect of our daily routines, catering to both professional and personal needs [8]. The cloud provider may have access to the personal data stored within their systems. If so, then a resulting question is whether the service offers encrypted storage and, if it does, how the decryption key is managed.

In most cases of encrypted storage, the service provider retains control of the decryption key. However, some providers offer advanced features [9,10] that ensure the key is accessible

solely to the client. When the decryption key is exclusively held by the client, it becomes possible to fully conceal the content from the storing server.

Even when a decryption key is solely retained by the client, there are still privacy concerns about access patterns. For instance, when a user/client requests the server to send the encrypted content of a file (or a portion of it), the server gains insight into which file, or which part of it, is being accessed. It can also discern whether the operation is a write or read action. In the case of a write operation, the ciphertext located at the accessed area is modified. By observing these generated patterns over time, the server can ascertain how frequently a file is accessed, how often it is read compared to being written, and any potential correlations between these two activities.

Research [3, 4] shows that a server can actually learn a surprising amount of private information, simply by monitoring access patterns. In fact, researchers have demonstrated that it is even possible to predict the underlying plaintexts just by observing the access patterns on encrypted storage systems [11].

This concern extends beyond just personal cloud storage accounts; it applies to any scenario where a client stores personal data on a server and accesses it regularly. For example, the Government of Canada retains a substantial amount of sensitive information in the cloud [12], typically in an encrypted format. However, it is still possible for the cloud server to accumulate a significant amount of information, merely by observing the access patterns, regardless of the strength of the data encryption scheme. According to a current study, 60% of the organizations' data is already outsourced to the cloud server [13]. This trend is expected to continue growing [14]. Consequently, there is a pressing need to protect access patterns in these contexts.

To obscure access patterns, a client can take a trivial approach: divide the outsourced data into equal-sized blocks, encrypt each block individually, and store them on the server in a shuffled order. This strategy prevents the server from correlating accesses, based on content or location. While this method is effective if the client accesses each block only once, any repeated access to a specific server location signals that the same block has been accessed multiple times. To mitigate this issue, the client must re-encrypt and re-shuffle each block after every access. However, this solution can be quite resource-intensive, resulting in a complexity of $O(\mathsf{N})$ per access, where N represents the total number of blocks stored on the remote server.

Thus, the question remains: is there a more efficient way to achieve access pattern privacy?

2.2 Related Work

Over the past thirty years, numerous solutions have been proposed to reduce the overhead related to protecting access patterns. This overhead encompasses both bandwidth and computational burdens. While some progress has been made in reducing bandwidth usage, it has been proven that the computation overhead has a lower bound and cannot fall below $O(\log \mathsf{N})$[1] per access [15].

2.2.1 Early Research

In this context of access pattern privacy, Oded Goldreich was the first to introduce the term Oblivious RAM (ORAM) [16]. Together with Rafail Ostrovsky, they proposed the first non-trivial solution with a complexity of $O(\sqrt{\mathsf{N}})$, known as square-root ORAM [15]. Their approach involved storing data in an encrypted and shuffled database, while also utilizing a server-side encrypted cache, referred to as the shelter. During each request, the client performs a linear scan of the shelter.

If the desired block is not located in the shelter, the client then accesses the block from the actual shuffled database. Conversely, if the block is already found in the shelter, the client *touches*[2] a dummy location in the shuffled database to mislead the server. This approach ensures that, regardless of the actual access pattern, the server always observes a linear scan of the shelter, followed by access to an untouched location within the shuffled database. After accessing $\sqrt{\mathsf{N}}$ blocks, it becomes necessary to re-encrypt and re-shuffle the entire server storage, resulting in an overhead of $O(\sqrt{\mathsf{N}})$.

In that seminal paper, the authors refined the initial concept by implementing a hierarchical arrangement of shelters of varying sizes. This hierarchical approach successfully reduced the overhead to $O((\log \mathsf{N})^3)$. Additionally, they proved that an overhead of $\Omega(\log \mathsf{N})$ is necessary to attain perfect privacy in an ORAM framework.

2.2.2 Towards Reducing Overhead to the Theoretical Limit

As a natural consequence, the research community started finding an ORAM scheme that could meet the theoretical limit. PathORAM [17], having $O(\log \mathsf{N})$ overhead, is one of the most popular breakthroughs in this direction [18]. Instead of a shelter-based approach,

[1] Unless otherwise stated, in this thesis $\log(\cdot)$ denotes $\log_2(\cdot)$.

[2] In ORAM context, *touching* means replacing the existing ciphertext with a new ciphertext.

in PathORAM, the server stores the outsourced N encrypted data blocks in a binary tree having $\log \mathsf{N}$ levels. The client maps each block to a random path, which means the block resides *somewhere* on that mapped path.

To access a block, the client looks at its local *position map* to find the mapped path. Then reads all the blocks on that path of the server tree. The requested block is then remapped to a new random path and placed either along that path or in the client's local storage, called the *stash*. Additionally, whenever possible, the client attempts to push existing blocks along a path towards the leaf, thereby creating space for new blocks.

In PathORAM, the client needs to maintain a local position map that tracks the indices of N blocks. While this position map is theoretically of size $O(\mathsf{N})$, it is usually not a significant concern. Each block index can generally be represented by a small number, and storing all N block indices typically requires minimal client space. If the client prefers not to retain even this small amount of data locally, it can recursively outsource the position map to the server. Although this method adds complexity, raising the overhead to $O(\log \mathsf{N}^2)$.

Asharov et al. constructed the optimal $O(\log \mathsf{N})$-ORAM, OptRAMa [19], even with a constant amount of client storage. Even if an ORAM with theoretical minimal overhead is achieved, it still does not result in a practical scheme. This is because $O(\log \mathsf{N})$ overhead does not only mean the client has to incur $O(\log \mathsf{N})\times$ computation; it also means the client has to incur $O(\log \mathsf{N})\times$ bandwidth and $O(\log \mathsf{N})\times$ latency while accessing each block.

Server storage costs are gradually decreasing [20]. Considering this fact, Ren et al. proposed Ring ORAM [21], in which they showed that it is possible to reduce the bandwidth blowup from $O(\log \mathsf{N})\times$ to $O(1)\times$ by keeping some metadata (e.g., encryption seed, bitmask, etc.) in the server along with the outsourced data. Also, they used a technique where the server *XOR*s multiple encrypted blocks together and sends a single compressed response to the client. While the bandwidth is reduced, it is not possible to reduce the computation overhead of an ORAM scheme below $O(\log \mathsf{N})$ due to its theoretical bound, *when both read and write operations are supported*.

2.2.3 Situation-Specific Overhead Reduction

Surprisingly, although the lower limit of $O(\log \mathsf{N})$ applies to both read/write and read-only settings, concealing access patterns in a write-only setting is considerably easier [22]. Roche et al. proposed a write-only ORAM, WoORAM [23], with only $O(1)$ overhead. However, having no read capability at all makes it less useful as, mostly, in the majority of the situations, the client outsources some data to remote storage with the intention of accessing (reading) it later.

In a read-only setting, the server's content does not change. It gives a possibility of pre-performing the next shuffle in a square-root ORAM. Based on this idea, Tople et al. proposed a read-only ORAM, PRO-ORAM [24]. Moreover, they observed that Melbourne shuffle [25], an oblivious shuffling algorithm, can shuffle N-items in $O(\sqrt{\mathsf{N}})$ independent steps. So, although the total amount of shuffling work is $O(\sqrt{\mathsf{N}})$, it can be performed in $O(1)$ time by launching $O(\sqrt{\mathsf{N}})$ parallel threads. As a result, PRO-ORAM achieves $O(1)$ latency. However, the main concern is that if N is not relatively small, launching $\sqrt{\mathsf{N}}$ parallel treads is not practical, even for a powerful server.

On the other hand, reducing latency might be possible if the shuffling can be broken down into *online* and *offline* phases. The online phase occurs when the client accesses the remote storage for read/write. Some bare minimum tasks must be done during this phase; it determines the client-observed access latency. The offline phase can be postponed to a less busy period when the client and server jointly perform the remaining shuffling tasks.

Stefanov and Shi proposed ObliviStore [26], which requires only a *constant* amount of shuffling work in its online phase, resulting in an $O(1)$-latency ORAM. Dautrich et al. extended ObliviStore to cope with the client's bursty access patterns and proposed Burst ORAM [27]. Burst ORAM does not perform any shuffling work at all during the online phase, so it becomes even faster. However, here, ultimately, in *some idle* period, the *client* has to perform all the accumulated pending tasks to keep the ORAM in a usable state.

2.2.4 Utilizing Server's Computation Capability to Reduce Latency

In many situations, a remote server is treated only as a *storage device*, with no processing capability. It is assumed that, on client's instruction, the server can only read and write from certain locations. Even though the remote server is capable of performing some useful work, independently. If a client could outsource some part of the shuffling task to the server, the challenge would then be how to do so, without leaking any private information to the server.

Homomorphic encryption [28] enables computations to be conducted on encrypted data. If a client stores the homomorphically-encrypted data on the remote server, any computation could be performed by the server. This might help the client to outsource some part of their shuffling task. Apon et al. analyzed this idea and formalized the notion of *Verifiable Oblivious Storage* [29].

Devadas et al. applied the same idea and proposed a PathORAM-based construction, Onion ORAM [30]. In Onion ORAM, the required network bandwidth is reduced, by

treating each path of the server tree as a remote database and fetching the required block from that path using PIR techniques. After accessing each block, it is remapped to a new random path but placed in the root. When the root is about to become full, the client and server collaborate and evict the old blocks from it to make space for newer ones. During the eviction phase, instead of downloading the entire encrypted content and locally performing shuffling, the client sends the permutation instruction to the server in an encrypted format. The server performs the actual permutation via homomorphic evaluation.

Subsequently, Chen et al. improved Onion ORAM and proposed Onion Ring ORAM [31]. They showed how the homomorphic permutation can be done more efficiently and how by performing homomorphic expansion of the client's *instruction* on the server side, the bandwidth requirement can be reduced even more. Not only that, they showed that Fully Homomorphic Encryption (FHE)-based ORAM schemes are no longer theoretical possibilities by implementing their construction with the currently available FHE building blocks like cMUX-gate. However, the computation done by the client, especially during the offline eviction phase, is still quite heavy.

Recently Cong et al. proposed Panacea [32], another FHE-based ORAM scheme. It does not require any involvement of the client during the offline phase at all and achieves $O(1)$-client work per access. However, the access latency is high in this case. Naively, in Panacea, the server is required to perform $O(\mathsf{N})$ amount of computation per query. However, if only $1.5\times$ bandwidth and $3\times$ expansion in server storage are sacrificed, then the concept of probabilistic batch coding can be applied. In that case, instead of responding to queries individually, the server can process queries in a batch of size k, bringing the amortized computation cost per query down to $O(\mathsf{N}/k)$.

2.3 *RouterORAM*: An ORAM scheme with $O(1)$-latency

Due to the established theoretical limit of $O(\log \mathsf{N})$ overhead [15], as remote storage continues to grow in size, ORAM is likely to become increasingly impractical. However, advancements in server hardware are significant, and existing research suggests that leveraging the computational power of servers could reduce the burden on the client. Furthermore, we notice that it may even be possible to surpass the theoretical limit when targeting specific access patterns.

2.3.1 Targeted Problem Statement

The proof of the ORAM lower limit [33] examines a generic scenario in which the client can read or write any item, at any time, and in any order. For example, the client may remain active for 24 hours of a day and continuously access the remote storage. Another extreme example is when, out of a total of N outsourced blocks, the client persistently accesses only one specific block and nothing else.

Fortunately, the client's access patterns are not that extreme for the majority of the real-life use cases. For the majority of use cases, the client does not engage with remote storage continuously over a 24-hour period; instead, they access remote storage in short bursts [34]. Additionally, read operations significantly outnumber write operations [35, 36].

Therefore, rather than attempting to protect all possible access patterns, we restrict the problem space to protecting these *common* access patterns. Specifically, our goal is to minimize both latency and the client's overhead in an ORAM system, where it is anticipated that the client will access remote storage in bursts, and the volume of read operations will exceed that of write operations.

2.3.2 Background: PathORAM

Our solution is based on the PathORAM [17] construction. First, we will provide a succinct overview of PathORAM. In this approach, the server-side storage is structured as a binary tree, with each node representing a bucket that can accommodate a fixed number of encrypted blocks. The binary tree consists of N leaf buckets and $\log \mathsf{N}$ levels.

In PathORAM, each encrypted block is mapped to a uniformly random path within the server tree. This implies that the corresponding block resides *somewhere* along that designated path. Due to the randomness of this mapping, there is a small probability that certain paths may receive more mapped blocks than they can accommodate. To address this issue, the client maintains a small local storage area, known as a *stash*, to accommodate the unplaced blocks.

Whenever the client needs to access a block, they consult their secret position map to determine the mapped path. The client then fetches the entire path from the server and decrypts each block residing in that path to locate the requested block. Once accessed, the block is removed from its current path and reassigned to a new random path.

The path that was just read is then written back to the server after re-encrypting each individual block. Consequently, the server is unable to determine which specific block the

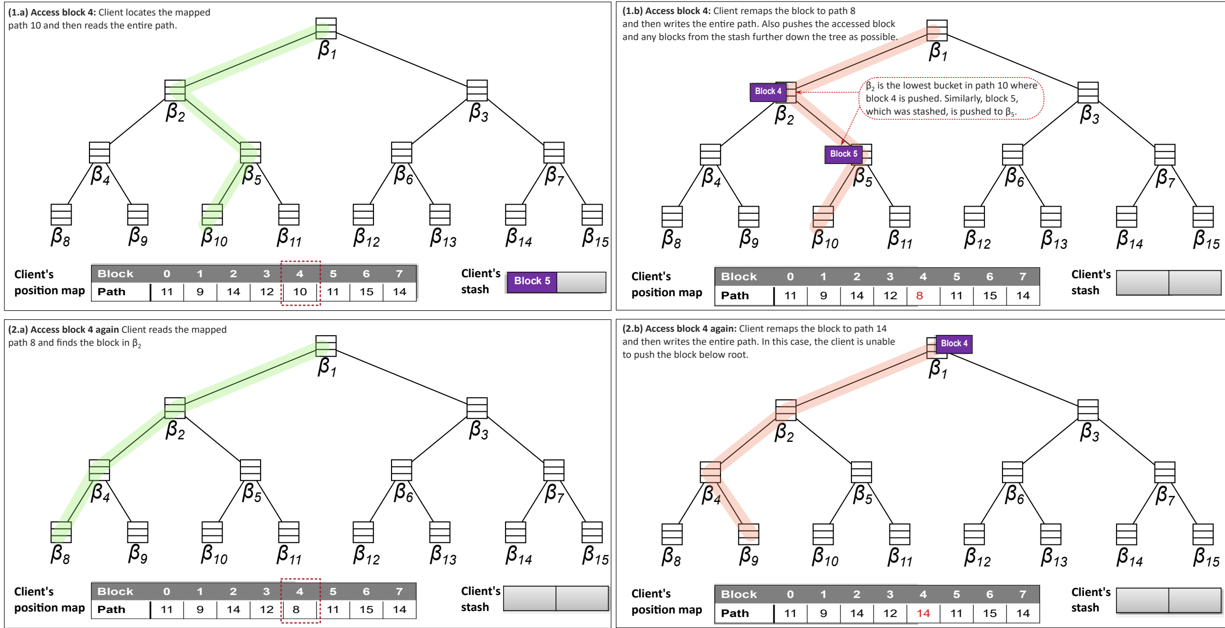


Figure 2.1: Example execution scenario of PathORAM protocol

client has just accessed. As the path is written back, additional blocks from the stash may be evicted into the path, provided there is sufficient space available. The client aims to push the blocks as far down their mapped path as possible during this write-back process. Figure 2.1 depicts the working of PathORAM.

In PathORAM, the client faces a network bandwidth and computational cost of $O(\log \mathsf{N})$. This arises because the client needs to retrieve the selected path, which consists of $O(\log \mathsf{N})$ blocks, and then decrypt each block individually. After this, the client updates the path and re-encrypts each block before writing the entire path back to the server storage.

2.3.3 Overview of *RouterORAM*

In a typical ORAM scheme, it is generally assumed that the server acts as a storage device and reads from and writes to its storage according to the client's instructions during each access. In the case of PathORAM, the server responds to each request by returning the requested path and then re-writes that same path using ciphertexts generated by the client. Meanwhile, the client takes on the responsibility of reorganizing the entire path, leaving the server idle.

Even though the server has the capability to handle various tasks, this potential often remains untapped. The client cannot simply shift the responsibility of reorganizing the path to the server; doing so could compromise the confidentiality of the sensitive information (i.e., the position map). In *RouterORAM*, we utilize homomorphic encryption, which enables computations on encrypted data, allowing us to delegate the reorganization task to the server, securely.

In contrast to PathORAM, where a block can be located *anywhere* along the mapped path, *RouterORAM* restricts each block to reside solely at the leaf bucket of the mapped path. This means that when a client needs to retrieve a block, it can do so by directly accessing only one specific leaf bucket. After accessing a block, the client remaps that block to a new randomly-chosen leaf. However, to confuse the server, the client next deliberately "misplaces" the block on the server tree (i.e. places the block in a different and unrelated random bucket from the newly remapped leaf).

In *RouterORAM*, once the block has been misplaced within the server tree, the client's involvement ends, and it becomes the server's job to rearrange the tree. The server continuously runs a background routing process that ensures all misplaced blocks are directed to their designated remapped leaf buckets (hence the name *RouterORAM*). To facilitate routing, the client specifies the destination location (i.e., the remapped leaf) as encrypted metadata. The server utilizes that information homomorphically to route the block to its proper destination.

In *RouterORAM*, the client touches only two buckets during each access. Beyond this $O(1)$-work, there is no additional effort required while accessing a block. As a result, *RouterORAM* achieves $O(1)$ latency as well as $O(1)$ client work. However, all of the server's routing tasks remain. *RouterORAM* simply helps the server handle these tasks at more suitable times, leading to a better distribution of workload over time. This is particularly beneficial in scenarios where servers experience a combination of busy and idle periods.

Although applying this strategy naively offers $O(1)$ latency, there is still an outstanding issue. The client should not be able to access the same block again, until the server has completed the routing of it to its re-mapped leaf. In any ORAM scheme, in addition to the real data blocks, the client must store a significant number of encrypted dummy blocks on the server to obfuscate the access pattern. However, in an IND-CPA secure encryption scheme [37], the ciphertexts of multiple replicas of the same block remain indistinguishable. Therefore, within the server tree, if we substitute some of the encrypted dummy blocks with encrypted replicas, it becomes possible to re-access the original block even before completing the routing of the previous replica.

In *RouterORAM*, the client can store multiple replicas of each outsourced block. In this way, some replicas of a block may be present in their mapped locations while others are in transit. As long as at least one replica is present at its mapped leaf bucket, the content of the block can be accessed. Therefore, *RouterORAM* not only lets the client outsource data to the remote storage, but also allows the client to manage the number of replicas of each block. By dynamically adjusting the number of replicas of each individual block according to their anticipated access frequencies, the client can effectively mask the time that the server requires to complete the routing.

2.4 *RouterORAM* Details

In this section, we explore the details of our ORAM construction. We outline the specifics of our protocol, as well as the storage structures utilized by both the client and the server. The *RouterORAM*'s notation is summarised in Appendix Table A.1.

2.4.1 Client Storage

The client is only required to store and maintain the position map, $\mathsf{pos}[]$, which is an array indexed by the block identifier $\mathsf{a} \in [1, \mathsf{N}]$. Each element of this array is a list. The client maps multiple replicas of each block to different leaf buckets of the server tree. The information about the mapped leaf buckets, corresponding to all the replicas of block a[3], is kept in the list stored in $\mathsf{pos}[\mathsf{a}]$. It is to be noted that, because of the deliberate misplacement, all the replicas may not be available immediately (the background routing might take some time to place them) at their mapped leaf bucket.

Hence, the availability information of all of the replicas also needs to be stored. Thus $\forall_{\mathsf{a} \in [\mathsf{N}]}, \mathsf{pos}[\mathsf{a}]$ is actually a list having the format: $\langle (\mathsf{b}_1, \tau_1), .., (\mathsf{b}_{\#(\mathsf{a})}, \tau_{\#(\mathsf{a})}) \rangle$, where each element is a tuple consisting of the label of the mapped leaf bucket (b_i) and the expected timestamp (τ_i), when the replica will be available at the bucket with label b_i. The client ensures that $\mathsf{pos}[\mathsf{a}]$ always remains sorted, in ascending order, based on the availability timestamps.

[3]In our discussion, the term "block a" refers to the block with identifier a.

2.4.2 Outsourced Data

Outsourced data is stored and accessed in fixed-size blocks. Each block blk, has a data part ($\mathsf{blk.d}$) having size B-bits and a fixed size metadata part ($\mathsf{blk.m}$). The client wants to outsource N different blocks to the server. The block identifier, a, is stored as a part of block metadata ($\mathsf{blk.m.a}$). The other part of the block metadata ($\mathsf{blk.m.x}$) stores the block's destination (i.e., the currently mapped leaf). In *RouterORAM*, the client may store multiple replicas of each block on the server. $\#(\mathsf{a})$ denotes the number of replicas of block a. The client is allowed to control this number from time to time. Along with *real* data blocks (and their replicas), some *dummy* blocks are also outsourced. The plain text content of a dummy block is all zeros (i.e., $\mathsf{blk.m.a} = \mathsf{blk.m.x} = 0$ and $\mathsf{blk.d} = \{0\}^{\mathsf{B}}$).

2.4.3 Server Storage

The server arranges its storage as a full binary tree. Each node of the tree is a bucket having a fixed number (Z) of slots. $\ell \in [1, \mathsf{L}]$ denotes the levels of the tree, where the root has $\ell = 1$ and all the leaves have $\ell = \mathsf{L}$. In each level ℓ, the tree has $2^{\ell-1}$ nodes or buckets. In *RouterORAM*, the number of leaf buckets is set to N, which means, $\mathsf{L} = \lfloor \log \mathsf{N} \rfloor + 1$. Each individual bucket is labeled with $\mathsf{b} \in [1, 2^{\mathsf{L}} - 1]$. The bucket having label b is denoted by β_{b}. The labeling is done in such a way that for bucket β_{b}, $\beta_{2\mathsf{b}}$ is its left child, and $\beta_{2\mathsf{b}+1}$ is its right child. The root bucket is labeled with $\mathsf{b} = 1$.

Each bucket slot stores an encrypted real or dummy block, along with its metadata part. Since everything is encrypted under an IND-CPA secure fully homomorphic encryption scheme (FHE), real blocks, their replicas, and dummy blocks are indistinguishable from each other. $\overline{\beta_{\mathsf{b}}[\mathsf{i}]}$ represents the stored ciphertext in i^{th} slot of β_{b} and $\beta_{\mathsf{b}}[\mathsf{i}]$ represents the corresponding plaintext. Leaf buckets are special; they additionally store a list having Z-elements, called the invalidation list ($\beta_{\mathsf{b}}.\mathsf{il}$). Elements of $\beta_{\mathsf{b}}.\mathsf{il}$ are either a valid block identifier $\in [1, \mathsf{N}]$ or 0. $\beta_{\mathsf{b}}.\mathsf{il}$ also remain encrypted with FHE.

2.4.4 *RouterORAM* Initialization

In this one-time activity, the client securely outsources real data blocks (along with some replicas) and some dummy blocks to the remote server. Algorithm 2.1, shows the details, where both the client and server participate jointly during the steps enclosed in the dashed boxes, and the rest of the steps are performed only by the client.

Algorithm 2.1 ORAMInit()

1: $\mathsf{ToBePlaced} \leftarrow [1, \mathsf{N}]$ ▷ Initialize the set with indices of all the blocks
2: **while** $\mathsf{ToBePlaced} \neq \emptyset$ **do**
3: $\quad \mathsf{a} \xleftarrow{\$} \mathsf{ToBePlaced}$
4: $\quad \mathsf{b} \xleftarrow{\$} [2^{\mathsf{L}-1}, 2^{\mathsf{L}} - 1]$ ▷ Randomly choose a leaf bucket label, b
5: $\quad \mathsf{pos}[\mathsf{a}] \cup (\mathsf{b}, \tau_{\mathsf{cur}})$
6: $\quad$ Store $\overline{\{\mathsf{a}, \mathsf{b}, \mathbb{D}(\mathsf{a})\}}$ in an empty slot of β_{b}
7: $\quad \#(\mathsf{a}) = \#(\mathsf{a}) - 1$
8: $\quad$ **if** $\#(\mathsf{a}) = 0$ **then** ▷ All replicas of this block are already placed
9: $\quad\quad \mathsf{ToBePlaced} = \mathsf{ToBePlaced} \setminus \{\mathsf{a}\}$
10: $\quad$ **end if**
11: **end while**
12: **for** Each remaining empty slots of the server tree **do**
13: $\quad$ Store $\overline{\{0, 0, \{0\}^{\mathsf{B}}\}}$ ▷ Store encrypted dummy block, along with dummy metadata
14: **end for**
15: **for** Each leaf bucket, β_{b}, of the server tree **do**
16: $\quad$ Initialize $\beta_{\mathsf{b}}.\mathsf{il} = \overline{\{0\}^{\mathsf{Z}}}$ ▷ Initialize the invalidation list with encryption of zeros
17: **end for**

The client first randomly chooses a block, a, and stores its data and metadata in a uniform randomly chosen leaf bucket, β_b, after encrypting with FHE. Accordingly, the client updates its local position map. Since, during this initialization, each replica is placed at the mapped leaf, they are available immediately. Hence, the client specifies the current timestamp, τ_{cur}, as the expected availability for each replica (Algorithm 2.1, Line: 5).

In this manner, the client first completes the placement of all replicas of the real blocks in a random order. Next, the remaining empty slots are filled with the ciphertext of dummy blocks along with their corresponding metadata. Finally, the client initializes the invalidation list for all the leaf buckets with encrypted zero values.

2.4.5 Accessing a Block in *RouterORAM*

In *RouterORAM*, the client can interact with the server through only one basic building block, an $\mathsf{Access}(\mathsf{a}^{\mathsf{R}}, \mathsf{a}^{\mathsf{I}}, \mathbb{D}(\mathsf{a}^{\mathsf{I}}))$-call (Algorithm 2.2). During each $\mathsf{Access}()$-call, one block, a^{R} is removed from the server tree, and another block, a^{I} having the data content $\mathbb{D}(\mathsf{a}^{\mathsf{I}})$ is inserted. a^{R} is removed from a leaf-bucket, while a^{I} is inserted into a non-leaf bucket.

By attentively choosing the parameter set of the $\mathsf{Access}()$-calls, the client can achieve the following high-layer operations: reading a block, writing a block, and managing the number of replicas for each block. *RouterORAM* ensures that individual $\mathsf{Access}()$-call preserve privacy. Thus, any higher-layer operations built upon it will also remain privacy-preserving.

Algorithm 2.2 $\mathsf{Access}(\mathsf{a^R}, \mathsf{a^I}, \mathbb{D}(\mathsf{a^I}))$

```
1: data ← Remove(a^R)
2: if a^R ≠ 0 and removal returns τ_1 then          ▷ No replica of a^R is available yet
3:     Wait until τ_1
4:     Access(a^R, a^I, D(a^I))                         ▷ Try again
5: else if a^R ≠ 0 and removal fails then               ▷ Removal failure
6:     Insert(0, ∗)                  ▷ Dummy insertion to maintain same touch pattern
7:     Wait for a random amount of time   ▷ So that timing does not reveal anything
8:     Access(a^R, a^I, D(a^I))                         ▷ Try again
9: else                                                  ▷ Successfully removed
10:    Insert({a^I, D(a^I)})
11:    if Insertion fails then
12:        Wait for a random amount of time
13:        Access(0, a^I, D(a^I))          ▷ Partial retry with dummy removal
14:    end if
15: end if
16: return data
```

RouterORAM keeps the parameter sets of the $\mathsf{Access}()$-calls (i.e., $\mathsf{a^R}, \mathsf{a^I}, \& \ \mathbb{D}(\mathsf{a^I})$) hidden from the server; moreover, the server learns nothing from the observable trace during the protocol execution. On rare occasions, some steps of $\mathsf{Access}()$ may fail. However, *RouterORAM* ensures that observed bucket touch patterns always remain the same: touching one leaf bucket, x, followed by touching another non-leaf bucket, w.

There may be instances when no replica of the intended block $\mathsf{a^R}$ is yet available. The client can ascertain this locally by examining its position map, which does not necessitate any interaction with the server. In such cases, the client must wait until the earliest available replica is ready (Algorithm 2.2, line 2).

A $\mathsf{Remove}()$ failure occurs when the client expects to find a block replica at a specific leaf node but does not locate it in the bucket returned by the server. As a result, the client is unable to proceed with the subsequent step of inserting block $\mathsf{a^I}$. However, since the

server has already recognized a $\mathsf{Remove}()$ call, the $\mathsf{Access}()$ function performs a dummy insertion to maintain an indistinguishable access pattern.

Subsequently, the client attempts again with a recursive $\mathsf{Access}()$ call (line 8). Although this constitutes a retrial, *RouterORAM* guarantees that the server does not learn any information from that. On the other hand, if the removal is successful but the insertion fails, $\mathsf{Access}()$ is invoked again, this time with a dummy removal. The objective remains that, regardless of the circumstances, the adversary (in this case, the server) must not gain any knowledge while the client performs an $\mathsf{Access}()$ call. Moreover, during the failure, the client waits for a random amount of time before invoking the second $\mathsf{Access}()$. This ensures that the time gap between two $\mathsf{Access}()$ calls does not reveal whether a failure occurred.

Specifically, the *RouterORAM* must adhere to the established privacy definition of an ORAM [17] scheme, which says: no information should be leaked about: 1) which block is being accessed; 2) how old it is (when it was last accessed); 3) whether the same block is being accessed (linkability); 4) access pattern (sequential, random, etc); or 5) whether the access is a read or a write.

In other words, *RouterORAM* meets the following privacy definition:

Definition 2.1 (ORAM Privacy)**.** *Let* $\tilde{\mathsf{y}} = ((\mathsf{a}^{\mathsf{R}}_{\mathsf{A}}, \mathsf{a}^{\mathsf{I}}_{\mathsf{A}}, \mathbb{D}(\mathsf{a}^{\mathsf{I}}_{\mathsf{A}})), .., (\mathsf{a}^{\mathsf{R}}_{1}, \mathsf{a}^{\mathsf{I}}_{1}, \mathbb{D}(\mathsf{a}^{\mathsf{I}}_{1})))$ *denote an access sequence of length* A*, where* $(\mathsf{a}^{\mathsf{R}}_{\mathsf{i}}, \mathsf{a}^{\mathsf{I}}_{\mathsf{i}}, \mathbb{D}(\mathsf{a}^{\mathsf{I}}_{\mathsf{i}}))$ *denotes the parameter set of the* i^{th} $\mathsf{Access}()$*-call. Let* $\mathsf{ORAM}(\tilde{\mathsf{y}})$ *be the observable trace on the server storage due to* $\tilde{\mathsf{y}}$*. Then RouterORAM guarantees that for any two access sequences* $\tilde{\mathsf{y}}_1$ *and* $\tilde{\mathsf{y}}_2$ *where* $|\tilde{\mathsf{y}}_1| = |\tilde{\mathsf{y}}_2|$*, an adversary* $\mathcal{A}$*, cannot distinguish between* $\mathsf{ORAM}(\tilde{\mathsf{y}}_1)$ *and* $\mathsf{ORAM}(\tilde{\mathsf{y}}_2)$*.*

Block Removal

At the first step of the $\mathsf{Access}()$-call, the client invokes $\mathsf{Remove}(\mathsf{a}^{\mathsf{R}})$(Algorithm 2.3) to remove a replica of the block a^{R}, from the server tree. $\mathsf{pos}[\mathsf{a}^{\mathsf{R}}]$ holds the list of labels corresponding to the leaf buckets that contain replicas of a^{R}. This list is maintained in sorted order, based on the availability of the replicas. Consequently, the client selects the leaf-bucket label x from the front of the list, where the earliest available replica of a^{R} is located. Since β_{x} may consist of up to Z different blocks, the client retrieves the entire β_{x} from the server and decrypts all the slots of it, to determine which one corresponds to a^{R}.

Algorithm 2.3 Remove(a^R)

```
1: data ← ∅
2: if a^R ≠ 0 then                                  ▷ It is not a dummy removal
3:     (b_1, τ_1) ← pos[a^R][1]     ▷ Chooses the first list-element as earliest available replica
4:     if τ_cur ≥ τ_1 then
5:         x ← b_1
6:         [Fetch β̄_x from server]
7:         β_x ← FHE.dec(β̄_x)
8:         for Each i ∈ [1, Z] do
9:             if β_x[i].a = a^R then
10:                data ← β_x[i].d, β̄'_x[i] ← dummȳ
11:            else
12:                β̄'_x[i] ← β̄_x[i]                    ▷ i.e.,re-encrypt
13:            end if
14:        end for
15:        if data = ∅ then           ▷ The block is not found in its expected location
16:            β_x.il ← β_x.il ∪ {a^R}
17:        end if
18:        β̄'_x.il ← β̄_x.il
19:        [Update β̄_x with β̄'_x in server]
20:        Remove (b_1, τ_1) from pos[a^R]     ▷ Now (b_2, τ_2) becomes the first list-element
21:    else                                     ▷ No replica of a^R is available yet
22:        return τ_1     ▷ Make no server interaction and return when the earliest replica
       will be available.
23:    end if
24: else                                             ▷ It's a dummy removal
25:    x ←$ [2^(L−1), 2^L − 1]
26:    [Update β̄_x in the server, with its re-encrypted version β̄'_x]
27: end if
28: if data = ∅ then ▷ The block is either unavailable or not found in the intended bucket
29:    return Failure
30: else
31:    return data
32: end if
```

After finding a^R, the client reads its content and makes the corresponding slot empty by

writing all zeros (i.e., dummy) in it. In some special situations, a^{R} might not be available in β_{x}; in that case, the client only updates the invalidation list metadata ($\beta_{\mathsf{x}}.\mathsf{il}$) of the bucket (line 16). Irrespective of the situation, the client stores the updated bucket, β'_{x}, in the same leaf of the server tree (line 19). To protect privacy, the client re-encrypts each slot and the metadata of the bucket before sending them to the server.

It is possible that the earliest available replica of the intended block is not yet present in its designated leaf bucket. In that case, the Remove() only returns the timestamp, when the earliest available replica will be available. The client adjusts its access accordingly. In this situation, no server interaction is required. Hence, the server learns nothing about this situation. Conversely, there may be instances when routing delays hinder the replicas from reaching their destination in timely manner. As a result, the client is unable to retrieve the block data, leading to a failure in the removal process. The client remains unaware of this issue unless it interacts with the server.

Sometimes (e.g. to deal with failures or other situations), the invoking Access()-algorithm might be required to issue a dummy Remove()-call (e.g., Algorithm 2.2, line 11). In that case, the client just replaces a random leaf with its re-encryption, without altering anything in its local position map. The goal is to make the actual and the dummy removal access pattern indistinguishable for the server.

Block Insertion

Each removal is paired with an insertion (Algorithm 2.4). To insert the block a^{I}, having data content $\mathbb{D}(\mathsf{a}^{\mathsf{I}})$, the client first locally maps the block with a random *leaf* bucket label: x, but deliberately misplaces the block to another random and independent *non-leaf* bucket of the server, β_{w} (line 2).

After selecting the non-leaf bucket, the client retrieves the entire bucket from the server and locally inserts the intended block into an empty slot of that bucket. In some rare circumstances (discussed later in Section 2.6), the chosen non-leaf bucket may be full, preventing the client from inserting the intended block. However, regardless of the situation, the client re-encrypts all the contents of the touched bucket, β_{w}, before sending it back to the server. Consequently, the server is unable to determine which specific slot of β_{w} has been updated, or even if any slots have been updated at all.

Algorithm 2.4 $\mathsf{Insert}(\mathsf{a^l}, \mathbb{D}(\mathsf{a^l}))$

1: $\mathsf{st} \leftarrow \mathsf{failure}$
2: $\mathsf{x} \xleftarrow{\$} [2^{\mathsf{L}-1}, 2^{\mathsf{L}} - 1], \mathsf{w} \xleftarrow{\$} [1, 2^{\mathsf{L}-1} - 1]$
3: **if** $\mathsf{a^l} \neq 0$ **then**
4: Fetch $\overline{\beta_\mathsf{w}}$ from server
5: $\beta_\mathsf{w} \leftarrow \mathsf{FHE.dec}(\overline{\beta_\mathsf{w}})$
6: **for** *Each* $\mathsf{i} \in [1, \mathsf{Z}]$ **do**
7: **if** $(\beta_\mathsf{w}[\mathsf{i}].\mathsf{a} = 0)$ and $\mathsf{st} \neq \mathsf{success}$ **then** ▷ Insert into an empty slot
8: $\overline{\beta'_\mathsf{w}[\mathsf{i}]} \leftarrow \overline{\{\mathsf{a^l}, \mathsf{x}, \mathbb{D}(\mathsf{a^l})\}}$
9: $\mathsf{st} \leftarrow \mathsf{success}$
10: **else**
11: $\overline{\beta'_\mathsf{w}[\mathsf{i}]} \leftarrow \overline{\beta_\mathsf{w}[\mathsf{i}]}$ ▷ Re-encrypt all other slots
12: **end if**
13: **end for**
14: Replace $\overline{\beta_\mathsf{w}}$ with $\overline{\beta'_\mathsf{w}}$ in server
15: **if** $\mathsf{st} = \mathsf{success}$ **then**
16: $\tau_{\mathsf{exp}} \leftarrow$ Determine expected arrival time, based on $\mathsf{x}, \mathsf{w}, \tau_{\mathsf{cur}}$ and the server's processing capability
17: Insert $(\mathsf{x}, \tau_{\mathsf{exp}})$ at the proper location in $\mathsf{pos}[\mathsf{a_l}]$
18: **end if**
19: **else** ▷ It's a dummy insertion
20: Replace $\overline{\beta_\mathsf{w}}$ in the server, with its re-encrypted version $\overline{\beta'_\mathsf{w}}$
21: $\mathsf{st} \leftarrow \mathsf{success}$
22: **end if**
23: **return** st

The background routing process obliviously ensures that the inserted block reaches from β_w to β_x so that the client can access the block in the future, from the newly mapped bucket, β_x. Since the background routing pattern is deterministic, and both x and w are known to the client, the client can locally compute the expected time, τ_{exp}, when the inserted block reaches its destined leaf bucket. To maintain the sorted order of $\mathsf{pos}[\mathsf{a^l}]$, after a successful insertion, the client adds $(\mathsf{x}, \tau_{\mathsf{exp}})$ as the ith element of $\mathsf{pos}[\mathsf{a^l}]$, such that $\tau_{\mathsf{i}-1} \leq \tau_{\mathsf{exp}} \leq \tau_{\mathsf{i}+1}$.

2.4.6 Higher Layer Operations of *RouterORAM* and their Latencies

RouterORAM provides three higher-layer storage operations to the client: (a.) Block read (b.) Block write and (c.) Managing the number of block replicas. During each $\mathsf{Access}()$ call, one block is removed while another is inserted into server storage. Thus, the client can perform high-level operations by adjusting the parameters of the $\mathsf{Access}()$ calls.

Block reading and manipulating the number of replicas are straightforward. The client chooses $\mathsf{a}^\mathsf{R} = \mathsf{a}_1$ and $\mathsf{a}^\mathsf{I} = 0$ (i.e., removes a_1 with a dummy insertion) to read the block a_1 and decrease $\#(\mathsf{a}_1)$ by one. To read a_1, without affecting $\#(\mathsf{a}_1)$, the client invokes $\mathsf{Access}()$ with $\mathsf{a}^\mathsf{R} = \mathsf{a}_1$, $\mathsf{a}^\mathsf{I} = \mathsf{a}_1$ and $\mathbb{D}(\mathsf{a}^\mathsf{I}) = \mathbb{D}(\mathsf{a}_1)$. With $\mathsf{a}^\mathsf{R} = \mathsf{a}_1$, $\mathsf{a}^\mathsf{I} = \mathsf{a}_2$ and $\mathbb{D}(\mathsf{a}^\mathsf{I}) = \mathbb{D}(\mathsf{a}_2)$, client can read a_1 with the effect of decreasing $\#(\mathsf{a}_1)$ and increasing $\#(\mathsf{a}_2)$ at the same time. Similarly, by choosing $\mathsf{a}^\mathsf{R} = 0$ and $\mathsf{a}^\mathsf{I} = \mathsf{a}_2$ ($\neq 0$), only $\#(\mathsf{a}_2)$ can be increased. Since all these operations mentioned above can be performed by invoking only one $\mathsf{Access}()$, the latency and the client work for all these operations are $O(1)$.

Although *RouterORAM* is targeted for read-dominated access patterns, block writing is also supported. However, since the server may store multiple replicas of each block, block writing is slightly more sophisticated than other operations. Suppose during writing, the client wants to update the content of block a_1 from $\mathbb{D}(\mathsf{a}_1)$ to $\mathbb{D}'(\mathsf{a}_1)$. If the block a_1 currently has $\#(\mathsf{a}_1)$-replicas in the server, all of those replicas must be updated. First, all stale replicas of a_1 must be removed from the server, and new $\#(\mathsf{a}_1)$-replicas having the updated content must then be inserted.

This can be achieved by invoking $\mathsf{Access}(\mathsf{a}^\mathsf{R} = \mathsf{a}_1, \mathsf{a}^\mathsf{I} = \mathsf{a}_1, \mathbb{D}'(\mathsf{a}_1))$, $\#(\mathsf{a}_1)$ times. As a result, the latency of the write operation will be $O(\#(\mathsf{a}_1))$. Since $\#(\mathsf{a}_1)$ is the client-controlled constant (independent of N), complexity theory-wise, the write operation will also have $O(1)$ latency. However, practically, the client incurs $\#(\mathsf{a}_1)\times$ work (as well as $\#(\mathsf{a}_1)\times$ latency) during writing block a_1, which is not required when reading the same block.

2.4.7 Background Routing in *RouterORAM*

While interacting with the client, the server runs a routing process in the background, continuously. After the client misplaces block a^I to β_w, the routing process ensures that it reaches its actual destination, β_x. For example, in Figure 2.2 at time instant t_1, the client maps the block a_1 with β_{14}, but misplaces it at a different and random bucket, β_5. The routing process then ensures that a_1 reaches its mapped leaf bucket by going through the

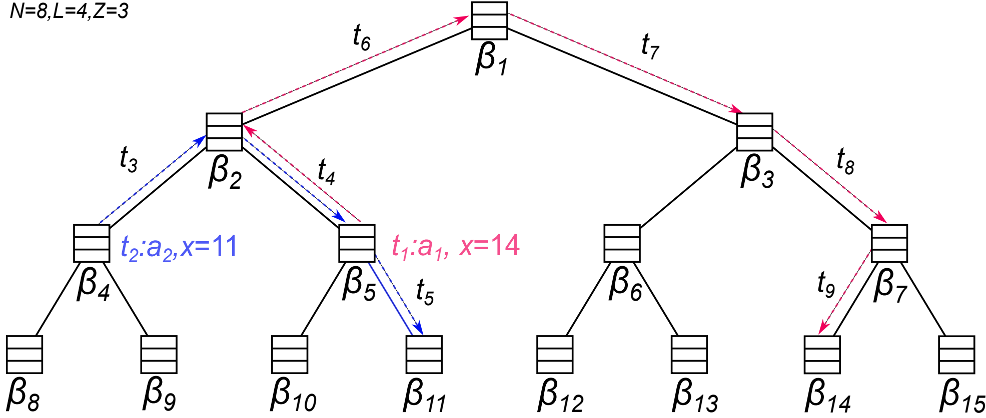


Figure 2.2: Server Tree and Routing

following path: $\beta_5 \rightarrow \beta_2 \rightarrow \beta_1 \rightarrow \beta_3 \rightarrow \beta_7 \rightarrow \beta_{14}$. Similarly, another misplaced block, a_2 must reach β_{11} from β_4.

The routing algorithm of *RouterORAM* (Algorithm 2.5) executes in a series of steps. In each step, an edge, $\mathsf{e}_{\hat{\mathsf{b}},\check{\mathsf{b}}}$, of the server tree is processed, which connects a lower-level bucket $\beta_{\check{\mathsf{b}}}$ with its parent, $\beta_{\hat{\mathsf{b}}}$. During this background routing, the edges of the server tree are processed in a fixed order, starting from the top-left edge of the binary tree and ending at the bottom-right edge (i.e., $\mathsf{e}_{1,2} \rightarrow \mathsf{e}_{1,3} \rightarrow \mathsf{e}_{2,4} \rightarrow .. \rightarrow \mathsf{e}_{\mathsf{N}-1,2\mathsf{N}-2} \rightarrow \mathsf{e}_{\mathsf{N}-1,2\mathsf{N}-1}$). Eventually, after processing each edge (i.e., $2 \times (\mathsf{N}-1)$ steps), the entire tree will have been processed, and this routing cycle continues forever.

The processing of each edge involves a series of homomorphic evaluations on the ciphertexts stored in the slots of $\beta_{\hat{\mathsf{b}}}$ and $\beta_{\check{\mathsf{b}}}$. As a result, it is possible for the server to do the routing activity correctly, without knowing the details of the blocks that are in transit. In addition, the design of our routing algorithm ensures that the server does not learn anything from the effects of the routing as well. For example, during the execution of the routing algorithm, the ciphertext content of some locations may change while the remaining locations remain unaltered. The server may try to use that information to backtrack the path of the inserted block. Our algorithm protects against of these cases.

When processing an edge, $\mathsf{e}_{\hat{\mathsf{b}},\check{\mathsf{b}}}$, the server first obliviously determines which blocks in the connecting buckets need to be transferred to the other bucket (see line 3 in Algorithm 2.5). It then rearranges these identified blocks appropriately between $\beta_{\hat{\mathsf{b}}}$ and $\beta_{\check{\mathsf{b}}}$. With each successive re-arrangement, each block-in-transit reduces the distance to its final destination. Referring to Figure 2.2, at t_4, edge $\mathsf{e}_{\hat{\mathsf{b}}=2,\check{\mathsf{b}}=5}$ is being processed. Block a_1 is currently located at β_5 and block a_2 is at β_2. After processing $\mathsf{e}_{2,5}$, a_1 reaches β_2, while a_2 ends up at β_5.

To achieve this, $\mathsf{MoveVerdict}()$ (Algorithm 2.6) homomorphically evaluates the metadata of all the blocks that currently reside in $\beta_{\hat{\mathsf{b}}}$ and $\beta_{\check{\mathsf{b}}}$ and outputs two encrypted arrays $(\mu_{\hat{\mathsf{b}}}, \mu_{\check{\mathsf{b}}})$. Each element of $\mu_{\mathsf{i}\in\{\hat{\mathsf{b}},\check{\mathsf{b}}\}}$ corresponds to the movement decision for each specific block, residing in $\beta_{\mathsf{i}\in\{\hat{\mathsf{b}},\check{\mathsf{b}}\}}$. If $\beta_{\check{\mathsf{b}}}$ is nearer to the final destination of the block, residing at $\beta_{\hat{\mathsf{b}}}[\mathsf{i}]$, then the block must be moved from $\beta_{\hat{\mathsf{b}}}$ to $\beta_{\check{\mathsf{b}}}$, which is encoded as $\mu_{\hat{\mathsf{b}}}[\mathsf{i}] = \Leftrightarrow$. Otherwise, $\beta_{\hat{\mathsf{b}}}[\mathsf{i}]$ does not require a movement and is encoded as $\not\Leftrightarrow$. Similarly, $\mu_{\check{\mathsf{b}}}[]$ is generated for $\beta_{\check{\mathsf{b}}}[]$.

Algorithm 2.5 $\mathsf{Route}()$

```
1: while True do
2:     for Each edge e_{b̂,b̌} of the tree do
3:         (μ̄_b̂, μ̄_b̌) ← MoveVerdict(β̄_b̂, β̄_b̌)    ▷ First determine, how individual blocks must be moved
4:         Move(μ̄_b̂, μ̄_b̌, β̄_b̂, β̄_b̌)    ▷ Then perform the actual movement
5:     end for
6: end while
```

Specifically, $\mathsf{MoveVerdict}()$ makes decisions about the required movement by checking, whether $\beta_{\check{\mathsf{b}}}$ is the common ancestor of the destination of the block in question. If it is, then, in accordance with our labeling strategy, $\check{\mathsf{b}}$ will correspond to the value represented by the leftmost $\ell_{\check{\mathsf{b}}}$ bits of the block's destination address (see Algorithm 2.6, lines 3 and 8). Subsequently, the server applies a homomorphic selection function (defined as $\overline{\mathsf{C}} = \mathsf{Select}(\overline{\mathsf{bit}}, \overline{\mathsf{A}}, \overline{\mathsf{B}}) = \overline{(1 - \mathsf{bit}).\mathsf{B} + \mathsf{bit}.\mathsf{A})}$ on the encrypted condition checking result $\overline{\mathsf{bit}}$. However, if the current slot contains only a dummy block, the prior decisions become irrelevant, and this situation is encoded as $\emptyset$ (Algorithm 2.6, line 6 & 12).

After this, the server moves the blocks in $\beta_{\hat{\mathsf{b}}}$ and $\beta_{\check{\mathsf{b}}}$ according to the decisions captured in $\mu_{\hat{\mathsf{b}}}$ and $\mu_{\check{\mathsf{b}}}$ (Algorithm 2.7). Our algorithm guarantees that the generated trace remains indistinguishable, no matter the conditions encountered. Each step of the algorithm produces FHE ciphertexts, which are already IND-CPA secure. We make sure that the number of iterations in the algorithm stays constant, even when it needs to make some conditional choices.

Basically, there could be three kinds of scenarios: (a) Swap a block in $\beta_{\hat{\mathsf{b}}}$, which requires a downward movement with a block in $\beta_{\check{\mathsf{b}}}$ requiring an upward movement (e.g., in Figure 2.2 processing $\mathsf{e}_{2,5}$ at t_4) (b) A block in $\beta_{\check{\mathsf{b}}}$ is required to move up, but there is no block in $\beta_{\hat{\mathsf{b}}}$ to go down. Hence, the block must be moved to an empty slot in $\beta_{\hat{b}}$ (in our example processing $\mathsf{e}_{2,4}$ at t_3) (c) Similarly, move a block down from $\beta_{\hat{\mathsf{b}}}$ to an empty slot of $\beta_{\check{\mathsf{b}}}$

(processing $\mathsf{e}_{5,11}$ at t_5). Situation (a) can be handled by simply performing a $\mathsf{Swap}()$ operation (Algorithm 2.8). Interestingly, moving a block to an empty slot means that after the movement, the original slot becomes empty. Thus, situations (b) and (c) can also be achieved by performing the $\mathsf{Swap}()$ operation (line 12 and 30 of Algorithm 2.7).

Algorithm 2.6 $\mathsf{MoveVerdict}(\overline{\beta_{\hat{\mathsf{b}}}}, \overline{\beta_{\check{\mathsf{b}}}})$

1: $\ell_{\hat{\mathsf{b}}} \leftarrow \lfloor \log(\hat{\mathsf{b}}) \rfloor + 1, \ell_{\check{\mathsf{b}}} \leftarrow \ell_{\hat{\mathsf{b}}} + 1$ ▷ Determine the levels of the connecting buckets
2: **for** *Each* $\mathsf{i} \in [1, \mathsf{Z}]$ **do**
3: $\quad \overline{\mathsf{bit}} \leftarrow \overline{(\check{\mathsf{b}} = (\beta_{\hat{\mathsf{b}}}[\mathsf{i}].\mathsf{m.x} \gg (\mathsf{L} - \ell_{\check{\mathsf{b}}})))}$ ▷ Is $\beta_{\check{\mathsf{b}}}$ a common ancestor of this block
4: $\quad \overline{\mu_{\hat{\mathsf{b}}}[\mathsf{i}]} \leftarrow \mathsf{Select}(\overline{\mathsf{bit}}, \overline{\Leftrightarrow}, \overline{\not\Leftrightarrow})$ ▷ If yes, then this block is required to be moved
5: $\quad \overline{\mathsf{bit}} \leftarrow \overline{(\beta_{\hat{\mathsf{b}}}[\mathsf{i}].\mathsf{m.x} = 0)}$ ▷ Whether the slot represents a dummy block
6: $\quad \overline{\mu_{\hat{\mathsf{b}}}[\mathsf{i}]} \leftarrow \mathsf{Select}(\overline{\mathsf{bit}}, \overline{\emptyset}, \overline{\mu_{\hat{\mathsf{b}}}[\mathsf{i}]})$ ▷ Then override the previous decision
7: **end for**
8: **for** *Each* $\mathsf{i} \in [1, \mathsf{Z}]$ **do**
9: $\quad \overline{\mathsf{bit}} \leftarrow \overline{(\check{\mathsf{b}} = (\beta_{\check{\mathsf{b}}}[\mathsf{i}].\mathsf{m.x} \gg (\mathsf{L} - \ell_{\check{\mathsf{b}}})))}$ ▷ Is $\beta_{\check{\mathsf{b}}}$ a common ancestor of this block
10: $\quad \overline{\mu_{\check{\mathsf{b}}}[\mathsf{i}]} \leftarrow \mathsf{Select}(\overline{\mathsf{bit}}, \overline{\not\Leftrightarrow}, \overline{\Leftrightarrow})$ ▷ If yes, then do not move it
11: $\quad \overline{\mathsf{bit}} \leftarrow \overline{(\beta_{\check{\mathsf{b}}}[\mathsf{i}].\mathsf{m.x} = 0)}$
12: $\quad \overline{\mu_{\check{\mathsf{b}}}[\mathsf{i}]} \leftarrow \mathsf{Select}(\overline{\mathsf{bit}}, \overline{\emptyset}, \overline{\mu_{\check{\mathsf{b}}}[\mathsf{i}]})$
13: **end for**
14: **return** $(\overline{\mu_{\hat{\mathsf{b}}}}, \overline{\mu_{\check{\mathsf{b}}}})$

Algorithm 2.7 $\mathsf{Move}(\overline{\mu_{\hat{\mathsf{b}}}}, \overline{\mu_{\check{\mathsf{b}}}}, \overline{\beta_{\hat{\mathsf{b}}}}, \overline{\beta_{\check{\mathsf{b}}}})$

1: **for** *Each* $\mathsf{i} \in [1, \mathsf{Z}]$ **do**
2:     **for** *Each* $\mathsf{j} \in [1, \mathsf{Z}]$ **do**
3:         $\overline{\mathsf{bit}} \leftarrow \overline{(\mu_{\hat{\mathsf{b}}}[\mathsf{i}] = \Leftrightarrow) \textbf{ and } (\mu_{\check{\mathsf{b}}}[\mathsf{j}] = \Leftrightarrow)}$ ▷ Situation (a): Requires swapping
4:         $(\overline{\beta_{\hat{\mathsf{b}}}[\mathsf{i}]}, \overline{\beta_{\check{\mathsf{b}}}[\mathsf{j}]}) \leftarrow \mathsf{Swap}(\overline{\mathsf{bit}}, \overline{\beta_{\hat{\mathsf{b}}}[\mathsf{i}]}, \overline{\beta_{\check{\mathsf{b}}}[\mathsf{j}]})$ ▷ Perform swapping
5:         $\overline{\mu_{\hat{\mathsf{b}}}[\mathsf{i}]} \leftarrow \mathsf{Select}(\overline{\mathsf{bit}}, \overline{\not\Leftrightarrow}, \overline{\mu_{\hat{\mathsf{b}}}[\mathsf{i}]})$ ▷ If swapped, not to be moved in next iteration
6:         $\overline{\mu_{\check{\mathsf{b}}}[\mathsf{j}]} \leftarrow \mathsf{Select}(\overline{\mathsf{bit}}, \overline{\not\Leftrightarrow}, \overline{\mu_{\check{\mathsf{b}}}[\mathsf{j}]})$
7:     **end for**
8: **end for**
9: **for** *Each* $\mathsf{i} \in [1, \mathsf{Z}]$ **do**
10:     **for** *Each* $\mathsf{j} \in [1, \mathsf{Z}]$ **do**
11:         $\overline{\mathsf{bit}} \leftarrow \overline{(\mu_{\hat{\mathsf{b}}}[\mathsf{j}] = \emptyset) \textbf{ and } (\mu_{\check{\mathsf{b}}}[\mathsf{i}] = \Leftrightarrow)}$ ▷ Situation (b): Move up $\beta_{\check{\mathsf{b}}} \rightarrow \beta_{\hat{\mathsf{b}}}$
12:         $(\overline{\beta_{\hat{\mathsf{b}}}[\mathsf{i}]}, \overline{\beta_{\check{\mathsf{b}}}[\mathsf{j}]}) \leftarrow \mathsf{Swap}(\overline{\mathsf{bit}}, \overline{\beta_{\hat{\mathsf{b}}}[\mathsf{i}]}, \overline{\beta_{\check{\mathsf{b}}}[\mathsf{j}]})$
13:         $\overline{\mu_{\hat{\mathsf{b}}}[\mathsf{j}]} \leftarrow \mathsf{Select}(\overline{\mathsf{bit}}, \overline{\not\Leftrightarrow}, \overline{\mu_{\hat{\mathsf{b}}}[\mathsf{j}]})$
14:         $\overline{\mu_{\check{\mathsf{b}}}[\mathsf{i}]} \leftarrow \mathsf{Select}(\overline{\mathsf{bit}}, \overline{\emptyset}, \overline{\mu_{\check{\mathsf{b}}}[\mathsf{i}]})$ ▷ After movement, $\beta_{\check{\mathsf{b}}}[\mathsf{j}]$ becomes empty
15:     **end for**
16: **end for**
17: **if** $\beta_{\check{\mathsf{b}}}$ is a leaf bucket **then** ▷ Invalidation
18:     **for** *Each* $\mathsf{i} \in [1, \mathsf{Z}]$ **do**
19:         **for** *Each* $\mathsf{j} \in [1, \mathsf{Z}]$ **do**
20:             $\overline{\mathsf{bit}} \leftarrow \overline{(\beta_{\check{\mathsf{b}}}.\mathsf{il}[\mathsf{j}] = \beta_{\hat{\mathsf{b}}}[\mathsf{i}].\mathsf{m.a})}$
21:             $\overline{\beta_{\hat{\mathsf{b}}}[\mathsf{i}]} \leftarrow \mathsf{Select}(\overline{\mathsf{bit}}, \overline{\mathsf{dummy}}, \overline{\beta_{\hat{\mathsf{b}}}[\mathsf{i}]})$ ▷ Delete the invalidated block
22:             $\overline{\mu_{\hat{\mathsf{b}}}[\mathsf{i}]} \leftarrow \mathsf{Select}(\overline{\mathsf{bit}}, \overline{\emptyset}, \overline{\mu_{\hat{\mathsf{b}}}[\mathsf{i}]})$ ▷ Update its movement decision
23:             $\overline{\beta_{\check{\mathsf{b}}}.\mathsf{il}[\mathsf{i}]} \leftarrow \mathsf{Select}(\overline{\mathsf{bit}}, \overline{0}, \overline{\beta_{\check{\mathsf{b}}}.\mathsf{il}[\mathsf{i}]})$ ▷ Clear it from the invalidation list
24:         **end for**
25:     **end for**
26: **end if**
27: **for** *Each* $\mathsf{i} \in [1, \mathsf{Z}]$ **do**
28:     **for** *Each* $\mathsf{j} \in [1, \mathsf{Z}]$ **do**
29:         $\overline{\mathsf{bit}} \leftarrow \overline{(\mu_{\hat{\mathsf{b}}}[\mathsf{i}] = \Leftrightarrow) \textbf{ and } (\mu_{\check{\mathsf{b}}}[\mathsf{j}] = \emptyset)}$ ▷ Situation (c): Move down $\beta_{\hat{\mathsf{b}}} \rightarrow \beta_{\check{\mathsf{b}}}$
30:         $(\overline{\beta_{\hat{\mathsf{b}}}[\mathsf{i}]}, \overline{\beta_{\check{\mathsf{b}}}[\mathsf{j}]}) \leftarrow \mathsf{Swap}(\overline{\mathsf{bit}}, \overline{\beta_{\hat{\mathsf{b}}}[\mathsf{i}]}, \overline{\beta_{\check{\mathsf{b}}}[\mathsf{j}]})$
31:         $\overline{\mu_{\check{\mathsf{b}}}[\mathsf{j}]} \leftarrow \mathsf{Select}(\overline{\mathsf{bit}}, \overline{\not\Leftrightarrow}, \overline{\mu_{\check{\mathsf{b}}}[\mathsf{j}]})$
32:     **end for**
33: **end for**

Algorithm 2.8 $\mathsf{Swap}(\overline{\mathsf{bit}}, \overline{\mathsf{d}_0}, \overline{\mathsf{d}_1})$

1: $\overline{\mathsf{tmp}} \leftarrow \mathsf{Select}(\overline{\mathsf{bit}}, \overline{\mathsf{d}_0}, \overline{\mathsf{d}_1})$
2: $\overline{\mathsf{d}_0} \leftarrow \overline{\mathsf{d}_0} + \overline{\mathsf{d}_1} - \overline{\mathsf{tmp}}$
3: $\overline{\mathsf{d}_1} \leftarrow \overline{\mathsf{tmp}}$
4: **return** $(\overline{\mathsf{d}_0}, \overline{\mathsf{d}_1})$

Note that, during writing a block, the client must remove all the existing replicas of it. However, at that moment, all the replicas might not yet have reached their mapped leaf bucket. So, the client is required to somehow notify the routing process to invalidate those replicas, upon arrival. The client achieves that by adding the block's identity in the invalidation list ($\beta.\mathsf{il}$) of the mapped buckets (Algorithm 2.3, line:16).

Therefore, before moving down any block to a leaf node, Move() performs some additional operations (Algorithm 2.7, line 17-26). It evaluates the condition whether the identity of the block-in-transit ($\beta_{\check{\mathsf{b}}}.\mathsf{m.a}$) matches with any of the elements in $\check{\beta}.\mathsf{il}$ (line:20). If there is a match, then the block is not moved down but deleted by replacing the corresponding slot with $\overline{\mathsf{dummy}}$.

2.5 Privacy Analysis

We analyze the privacy properties of *RouterORAM* against an honest but curious adversary, which is the common threat model for ORAM [15, 19, 21, 23, 26, 27, 31, 32]. Since all data remains encrypted in remote storage, the adversary $\mathcal{A}$ (in this instance, the server) is unable to learn about the outsourced data content. However, $\mathcal{A}$ can observe a series of ciphertext changes at specific locations within the server's storage, which is the trace generated during the execution of the *RouterORAM* protocol.

Similar to other ORAM implementations, our objective is to prevent $\mathcal{A}$ from gaining any insights from the generated trace, as the client accesses the server storage. Furthermore, *RouterORAM* ensures that $\mathcal{A}$ cannot learn anything during the execution of the routing process either. We will now analyze the privacy properties of our protocol in a series of gradual steps.

Theorem 2.1 (Privacy of Remove()). *If* FHE *is IND-CPA secure,* $\mathcal{A}$ *learns nothing during a* Remove() *call, except* x*, the label of the touched leaf bucket.*

Proof. During Remove(), the client touches a leaf bucket, β_{x}. Since the remote storage is controlled by $\mathcal{A}$, it can notice the touched location, x. Depending on the input parameter

of the $\mathsf{Remove}()$-call, the actual plaintext content of β_{x} changes differently (e.g., if $\mathsf{a}^{\mathsf{R}} = 0$, then nothing is changed at all). Moreover, occasionally, the $\mathsf{Remove}()$ may fail.

Regardless, the client always re-encrypts all the contents of the touched bucket. So, $\mathcal{A}$ only observes that the entire ciphertext of the touched bucket changes from $\overline{\beta_{\mathsf{x}}}$ to $\overline{\beta'_{\mathsf{x}}}$. Since FHE is IND-CPA secure and λ is its security parameter, we can say:

$$\forall_{\mathsf{i} \in [\mathsf{Z}]} \left| \Pr\left[\mathcal{A}(\overline{\beta_{\mathsf{x}}}, \overline{\beta'_{\mathsf{x}}}) \rightarrow 1 \mid \beta_{\mathsf{x}}.[\mathsf{i}] = \beta'_{\mathsf{x}}.[\mathsf{i}]\right] - \Pr\left[\mathcal{A}(\overline{\beta_{\mathsf{x}}}, \overline{\beta'_{\mathsf{x}}}) \rightarrow 1 \mid \beta_{\mathsf{x}}.[\mathsf{i}] \neq \beta'_{\mathsf{x}}.[\mathsf{i}]\right] \right| \leq \operatorname{negl}(\lambda)$$

Similarly,

$$\forall_{\mathsf{i} \in [\mathsf{Z}]} \left| \Pr\left[\mathcal{A}(\overline{\beta_{\mathsf{x}}}, \overline{\beta'_{\mathsf{x}}}) \rightarrow 1 \mid \beta_{\mathsf{x}}.\mathsf{li}[\mathsf{i}] = \beta'_{\mathsf{x}}.\mathsf{li}[\mathsf{i}]\right] - \Pr\left[\mathcal{A}(\overline{\beta_{\mathsf{x}}}, \overline{\beta'_{\mathsf{x}}}) \rightarrow 1 \mid \beta_{\mathsf{x}}.\mathsf{li}[\mathsf{i}] \neq \beta'_{\mathsf{x}}.\mathsf{li}[\mathsf{i}]\right] \right| \leq \operatorname{negl}(\lambda)$$

In other words, any change in any slot of the touched leaf bucket or any change in the invalidation list metadata remains indistinguishable to $\mathcal{A}$. Thus, $\mathcal{A}$ learns nothing except the label x during a $\mathsf{Remove}()$ call. □

Corollary 2.1. *A real and a dummy removal are indistinguishable from each other.*

Proof. Suppose $\mathsf{Remove}_1()$ is a real removal which touches β_{x_1} and $\mathsf{Remove}_2()$ a dummy removal (i.e., a $\mathsf{Remove}(\mathsf{a}^{\mathsf{R}} = 0)$-call) which touches β_{x_2}. According to Theorem: 2.1, $\mathcal{A}$ only learns x_1 and x_2. Since the client chooses both x_1 and x_2 uniformly randomly from the same distribution ($[2^{\mathsf{L}-1}, 2^{\mathsf{L}} - 1]$), $\mathcal{A}$ cannot differentiate them. □

Theorem 2.2 (Privacy of $\mathsf{Insert}()$)**.** *If* FHE *is IND-CPA secure,* $\mathcal{A}$ *learns nothing during an* $\mathsf{Insert}()$ *call, except* w*, the label of the touched non-leaf bucket.*

Proof. The argument is similar to Theorem: 2.1. As the client re-encrypts all the content of the touched non-leaf bucket from $\overline{\beta_{\mathsf{w}}}$ to $\overline{\beta'_{\mathsf{w}}}$, it is impossible for $\mathcal{A}$ to guess which part of β_{w} has changed, or whether anything has changed at all. Hence, $\mathcal{A}$ only learns w. □

Corollary 2.2. *A real and a dummy insertion are indistinguishable from each other.*

Proof. The argument is similar to Corollary: 2.1. In both situations, the client chooses the touched non-leaf bucket uniformly randomly from the same distribution. □

Corollary 2.3. *During a completely successful* $\mathsf{Access}()$ *call,* $\mathcal{A}$ *learns nothing except* x *and* w*, the label of the touched leaf bucket, and the label of the touched non-leaf bucket.*

Proof. During a completely successful $\mathsf{Access}()$, no retrial is required and the client only makes one successful $\mathsf{Remove}()$ call and one successful $\mathsf{Insert}()$ call. Since none of them reveal anything except x and w, $\mathsf{Access}()$ also reveals nothing more. □

Theorem 2.3 (Unnoticeable failure during an $\mathsf{Access}()$ call)**.** *After observing a single* $\mathsf{Access}()$*-call, $\mathcal{A}$ cannot tell whether it has completed with any failure or not.*

Proof. $\mathsf{Access}()$ consists of one $\mathsf{Remove}()$ and one $\mathsf{Insert}()$. On rare occasions, either of them may fail. As a result, overall $\mathsf{Access}()$ may fail as well. In the case of failure, another $\mathsf{Access}()$ is invoked (for retrial), but during the current $\mathsf{Access}()$-call $\mathcal{A}$ cannot determine whether there is a failure or not.

$\mathsf{Remove}()$ fails when the client expects the block a^R ($\neq 0$) at its mapped leaf bucket β_x, but cannot find it there. In that case, $\mathsf{Remove}()$ returns $\mathsf{Failure}$ to the client, and the client issues a dummy insertion (Algorithm 2.2, line: 6) instead of inserting the actual a^I. However, $\mathcal{A}$ can neither notice $\mathsf{Remove}()$ has failed (Theorem: 2.1), nor that the subsequent insertion is a dummy one (Corollary: 2.2). Thus, $\mathcal{A}$ cannot determine whether the $\mathsf{Access}()$ call completed with a failed removal or not.

$\mathsf{Insert}()$ fails, when the randomly chosen bucket β_w is already full, and a^I ($\neq 0$) cannot be inserted into β_w. Note that $\mathsf{Insert}(\mathsf{a}^\mathsf{I}, \mathbb{D}(\mathsf{a}^\mathsf{I}))$ is invoked only after a successful $\mathsf{Remove}()$, and no additional step is performed within the same $\mathsf{Access}()$-call, in the case of insertion failure. Since $\mathcal{A}$ cannot notice the failed insertion (Theorem: 2.2), $\mathcal{A}$ cannot determine whether the observed $\mathsf{Access}()$ call completed with a failed insertion. □

Corollary 2.4 (Indistinguishable retrial)**.** *If $\mathcal{A}$ observes two successive* $\mathsf{Access}()$ *calls, it cannot determine whether they are two independent* $\mathsf{Access}()$ *calls or if the second one is a retrial due to the failure of the first call.*

Proof. Suppose $\mathsf{Access}_2()$ is a retrial call, invoked because of the failure of either $\mathsf{Remove}()$ or $\mathsf{Insert}()$ during $\mathsf{Access}_1()$. So, $\mathcal{A}$ notices a pair of buckets $(\beta_{\mathsf{x}_1}, \beta_{\mathsf{w}_1})$ touched during $\mathsf{Access}_1()$ and another pair $(\beta_{\mathsf{x}_2}, \beta_{\mathsf{w}_2})$ touched during $\mathsf{Access}_2()$. The client randomly chooses the labels of the touched leaf and non-leaf buckets from uniform distributions during a single access (Algorithm 2.4, line 2); hence, those are statistically independent. We have to show that during the retrial as well, the touched bucket locations are statistically independent.

If $\mathsf{Remove}()$ fails during $\mathsf{Access}_1()$, x_1 is removed from $\mathsf{pos}[\mathsf{a}^\mathsf{R}]$ and $\mathsf{Access}_2()$ touches β_{x_2}, the mapped location of next replica of a^R. Since the mapped locations of all the replicas of a^R are chosen randomly and independently, x_2 and x_1 are statistically independent. On the other hand, in the case of $\mathsf{Insert}()$ failure in $\mathsf{Access}_1()$, a dummy removal is made during $\mathsf{Access}_2()$, which also chooses x_2 randomly and independently (Algorithm 2.2, line: 11 and Algorithm 2.3, line: 25). As a result, x_1 and x_2 are always independent of each other.

Regarding the non-leaf bucket, irrespective of the nature of failure in $\mathsf{Access}_1()$, the $\mathsf{Insert}()$ call during $\mathsf{Access}_2()$, always chooses w_2 randomly and independently. Hence, there

is no relation between w_1 and w_2. Therefore, $\mathsf{x}_1, \mathsf{x}_2, \mathsf{w}_1$, and w_2 always remain statistically independent. Moreover, in case of a failure, waiting for a random amount of time before invoking the second Access() ensures that the timing of two successive Access() calls does not reveal any information. □

Lemma 2.1. *By observing a series of* Access() *calls, $\mathcal{A}$ only learns the labels of a sequence of the bucket labels, nothing else.*

Corollary 2.5 (Access pattern privacy)**.** *RouterORAM meets ORAM privacy definition: 2.1*

Proof. From $\mathsf{ORAM}(\tilde{\mathsf{y}})$, $\mathcal{A}$ only learns a sequence of touched bucket labels $\tilde{\mathsf{b}} = \langle(x_A, w_A), .., (x_1, w_1)\rangle$ (*lemma: 2.1*). Irrespective of the situation (i.e., irrespective of the input parameters, success or failure, original or retrial, etc.), labels of all the touched buckets (i.e., $\{x_1, .., x_A\} \cup \{w_1, ..w_A\}$) are statistically independent of each other. Thus, $\mathcal{A}$ cannot distinguish between two such series of ORAM accesses: $\mathsf{ORAM}(\tilde{\mathsf{y}_1})$ and $\mathsf{ORAM}(\tilde{\mathsf{y}_2})$, when $|\tilde{\mathsf{y}_1}| = |\tilde{\mathsf{y}_2}|$. □

Theorem 2.4 (Privacy of Route())**.** *$\mathcal{A}$ learns nothing while performing the background routing.*

Proof. In general, the execution flow of an algorithm can vary depending on its input (e.g., the number of iterations, branching decisions, etc.). $\mathcal{A}$ may attempt to closely monitor the execution flow of the routing algorithm to gain insights. However, even when the routing algorithm must make conditional decisions, the design of *RouterORAM* guarantees that the execution flow of all routing-related algorithms (Algorithm 2.5 - 2.8) remains constant, ensuring that all the steps are executed all the time. As a result, $\mathcal{A}$ cannot derive any information by observing the routing process's execution flow.

In *RouterORAM*, not only the outsourced data (and the metadata) but also any information derived while performing the routing activity (i.e., $\mu_{\hat{\mathsf{b}}}$, $\mu_{\check{\mathsf{b}}}$, the computed bit value in Algorithm 2.6, 2.7, etc.) always remain encrypted under FHE. By definition, the server is unable to access the underlying plaintext; it can only process this information homomorphically.

While the server can monitor the ciphertext values both before and after processing, this visibility does not provide the adversary, $\mathcal{A}$, with any useful information, as the homomorphic encryption scheme, FHE, maintains IND-CPA security. $\mathcal{A}$ is not only unaware of the underlying plaintext value but also unsure about whether any modifications to the underlying plaintext have occurred or not.

Nonetheless, $\mathcal{A}$ may discern which ciphertexts have been altered and which remain unmodified during the background routing process, potentially gaining some insight from

this information. To address this concern, *RouterORAM* guarantees that, regardless of the circumstances, all ciphertexts stored on the server are modified in a predetermined order. As a result, $\mathcal{A}$ is unable to derive any meaningful information from the routing-generated trace.

Thus, $\mathcal{A}$ cannot learn anything, while performing the background routing. □

2.6 Analysis of Block Access Failures

Previously, we stated that in *RouterORAM*, the client may occasionally encounter failures while accessing remote storage. In this section, we analyze these failures in detail and assess their impact on latency.

2.6.1 Effect of Access Frequency on Failure

The frequency with which the client accesses remote storage is the primary factor influencing failures in both the Remove() and Insert() operations. This access frequency impacts background routing congestion, which can occur when a block needs to be moved, but the target bucket is already full, thereby impeding the movement. Each Access() call leads the client to misplace a block in a non-leaf bucket. As the client accesses the remote storage more frequently, an increasing number of slots in these non-leaf buckets become occupied by the misplaced blocks, which in turn raises the likelihood of routing congestion.

When routing congestion occurs, it causes delays in the arrival of in-transit block replicas to their designated leaf buckets. Consequently, the client may be unable to locate the replica even after the expected arrival time, potentially leading to Remove() failures (Algorithm 2.3 line 15). However, it's important to note that routing congestion does not necessarily result in a Remove() failure. Such failures *only occur if* the routing congestion affects the specific block replica that the client attempts to access next.

As the client places a block into a non-leaf bucket slot with each access, the expected number of full non-leaf buckets increases as access frequency rises. Consequently, the likelihood that the randomly selected β_{w} during the Insert() is full, also increases with the frequency of access. This can be interpreted as the failure probability of the Insert() operation. Since Access() fails, whenever either of Insert() or Remove() fails, the failure probability of Access() is the sum of the failure probabilities of Insert() and Remove().

To better understand the extent of failures, we developed a simulator for our protocol, available at [38]. We simulate the steady, long-term (∼5 months) behaviors of

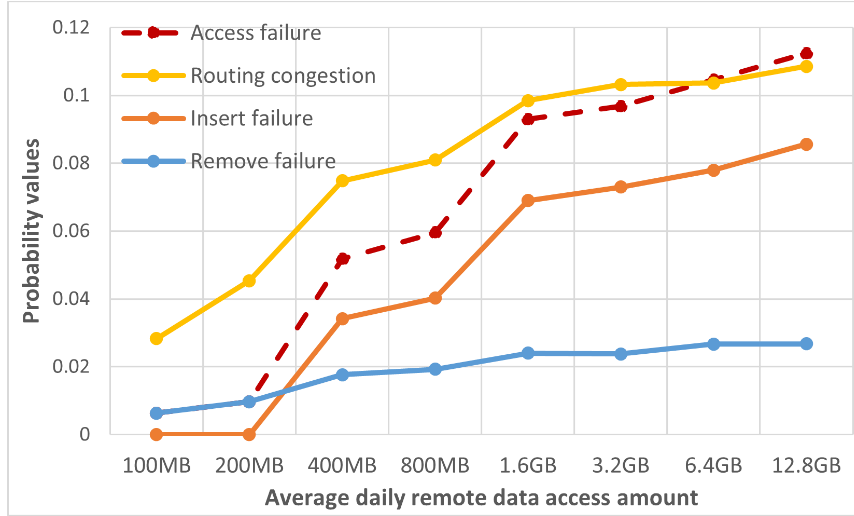


Figure 2.3: Effect of Access Frequency on Failures

RouterORAM with different access frequencies. Specifically, we vary the client's daily remote usage from 100MB/day to 12.8GB/day. The detailed simulation results are presented in Figure 2.3.

2.6.2 Effect of Failure on Latency

In the event of a failure during either Remove() or Insert() operation, the client subsequently invokes another Access(), necessitating two additional bucket touches. As a result, if the failure rate is excessively high, the client might find themselves effectively touching a significantly large number of buckets per access. If this is the case, the *effective* latency and client workload may no longer remain at $O(1)$.

In fact, the retrial of the Access() call may also fail, and the probability of this failure will be identical to that of the original call, potentially leading to an infinite series of attempts. Let $\mathsf{F_A}$ represent the failure probability of the Access() call. Following the principles of geometric distribution, we can calculate the expected number of bucket touches per access as follows:

$$2 \times \sum_{k=1}^{\infty} k \times F_A^{k-1} \times (1 - F_A) = 2(1 - F_A) \sum_{k=1}^{\infty} k F_A^{k-1} \tag{2.1}$$

Now, by expanding the power series and then simplifying that, we get:

$$2(1 - \mathsf{F_A}) \sum_{k=1}^{\infty} \mathsf{k}\mathsf{F_A}^{k-1} = \frac{2(1 - \mathsf{F_A})}{(1 - \mathsf{F_A})^2} = \frac{2}{(1 - \mathsf{F_A})} \quad (2.2)$$

The exact value of the Access()-failure probability, $\mathsf{F_A}$, depends mainly on the access frequency. By combining the results of Figure 2.3 and Equation: 2.2 we get Figure 2.4, which depicts the effective number of bucket touches with varying access frequencies.

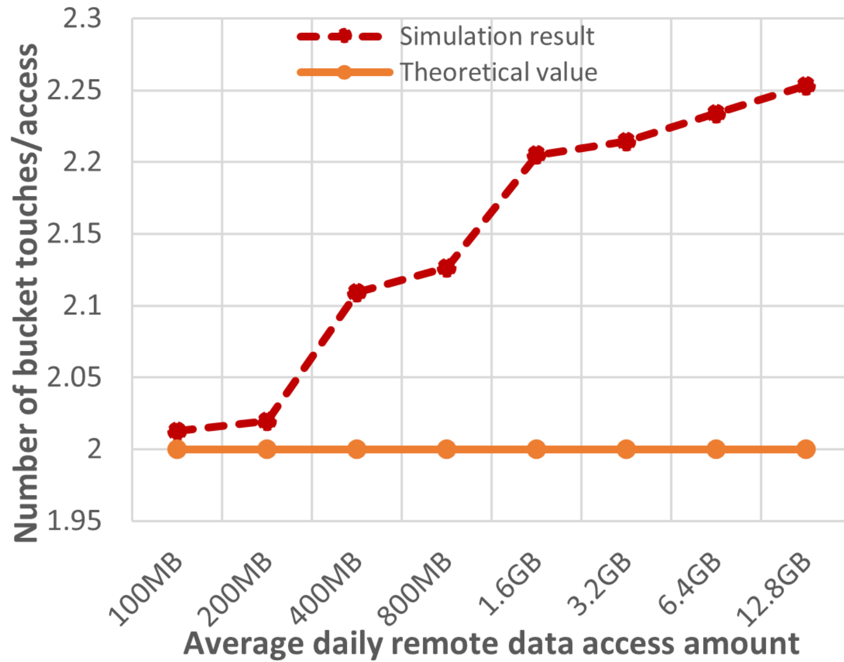


Figure 2.4: Number of effective bucket touches

From this, it can be observed that the expected number of bucket touches increases with access frequency. However, the rate of increase is very slow, and the expected number of bucket touches per access remains under 2.26 (instead of the theoretical value of 2), even for very high usage settings (12.8GB of remote data access/day). As a result, it can be concluded that *RouterORAM*'s latency and client work remain truly $O(1)$ in practical usage scenarios.

In our simulation, we conservatively assume a slow disk speed of 40 MB/s. This means that for daily access of X-MB of remote data, the server will spend $\frac{\mathsf{X}}{40}$ seconds performing the disk I/O. Therefore, the remaining time - $(24 * 60 * 60 - \frac{\mathsf{X}}{40})$ seconds - will be available

for the background routing process. Consequently, *RouterORAM* is anticipated to experience fewer failures with faster disks, as the server will require less time to manage client I/O requests, allowing it to devote more time to routing functions.

2.7 Discussion

We propose *RouterORAM*, which safeguards access patterns from the storage server in a private database setting with only $O(1)$ latency and $O(1)$ client workload. Although it does not have any restriction regarding the client's access pattern, its benefits can be exploited maximally for a bursty and read-dominated remote access pattern. This is a very common remote storage access pattern [34–36]. *RouterORAM* is based on the unique idea of server-assisted routing of deliberately misplaced blocks. It utilizes the server's unconsumed computation capability with homomorphic evaluation to minimize the access latency and the client's burden without compromising any privacy.

2.7.1 Summary of Theoretical Contribution

While it has been established that the server workload of an ORAM scheme cannot fall below $O(\log \mathsf{N})$, Apon et al. theoretically show that in the server computation model, the bandwidth overhead can be reduced to $O(1)$ [29]. The latest notable advancement in this model is the Onion-ring ORAM [31]. However, despite this reduction in bandwidth overhead, both latency and client workload remain at $O(\log \mathsf{N})$. In fact, it is still uncertain whether it is feasible to reduce latency below $O(\log \mathsf{N})$ for all possible access patterns.

Like ours, BurstORAM [27] aims to address bursty access patterns and achieved latency of $O(1)$. However, the effective client workload and bandwidth blowup in BurstORAM are still $O(\log \mathsf{N})$, because the client must remain active after its bursty access period to handle pending shuffling tasks. Additionally, BurstORAM requires the client to store a significant portion of the outsourced data ($O(\sqrt{\mathsf{N}})$) locally. Table 2.1 compares *RouterORAM* with existing ORAM schemes. Although WoORAM and PRO-ORAM also have $O(1)$ latency and client work, none of them support both reading and writing.

This raises an important question: can an ORAM that supports both read and write operations effectively lower latency, client workload, and bandwidth to $O(1)$ simultaneously? This is exactly what we have achieved in our research. In the read/write model, Panacea recently minimized the total client work to $O(1)$ by removing offline work altogether. However, its latency is still proportional to the amount of the outsourced data.

Table 2.1: Comparisons

Scheme	Allowed modes	Bandwidth blowup	Total server storage	Total client work	Total server work	Latency
PathORAM†[17]	R/W	$O(\log \mathsf{N})$	$O(\mathsf{N})$	$O(\log \mathsf{N})$	$O(\log \mathsf{N})$	$O(\log \mathsf{N})$
PRO-ORAM [24]	R-only	$O(1)$	$O(\mathsf{N})$	$O(1)$	$O(\sqrt{\mathsf{N}})$	$O(1)$
WoORAM [23]	W-only	$O(1)$	$O(\mathsf{N})$	$O(1)$	$O(1)$	$O(1)$
BurstORAM [27]	R/W	$O(\log \mathsf{N})$	$O(\mathsf{N})$	$O(\log \mathsf{N})$	$O(\log \mathsf{N})$	$O(1)$
Onion-Ring ORAM [31]	R/W	$O(1)$	$O(\mathsf{N})$	$O(\log \mathsf{N})$	$O(\log \mathsf{N})$	$O(\log \mathsf{N})$
Panacea ‡[32]	R/W	$O(1)$	$O(\mathsf{N})$	$O(1)$	$O(\mathsf{N}/\mathsf{k})$	$O(\mathsf{N}/\mathsf{k})$
RouterORAM	R/W	$O(1)$	$O(\#^{\mathrm{Re}})$	$O(1)$	$O(\log \mathsf{N})$	$O(1)$

† In non-recursive setting, where the client holds the entire position map
‡ It works in a batched setting, and k is the batch size.

From that aspect, *RouterORAM* is the first read/write ORAM scheme that achieves $O(1)$-client work, $O(1)$-bandwidth blowup and $O(1)$-latency together. The total amount of server work does not go away. It is still $O(\log \mathsf{N})$ due to the theoretical limit. Specifically, *RouterORAM* shows how, by maintaining multiple replicas in the server storage, the required work is not necessarily required to be done between two successive accesses. Rather, it can be *done independently by the server in the background*, as a later operation.

2.7.2 Practical Considerations

Asymptotically, our scheme reduces the client overheads without imposing any additional burden on the server side (refer to the Table 2.1). However, the actual values of these overheads and the absolute savings values depend on the implementation details, such as which FHE scheme is used or the block size etc. These implementation details also affect the actual financial costs and benefits of our solution. Moreover, the deployment details also influence the financial aspects of our solution. For instance, if our solution is deployed on a cloud server, then the hourly computation, storage and bandwidth costs of that cloud provider determine the overall cost of our solution.

However, our scheme has some benefits from an implementation standpoint. At each level of the server tree, the routing of the edges connecting the non-sibling buckets does not have any interdependencies. As a result, the background routing activity is highly parallelizable and can be completed in a shorter time by launching multiple threads. Specifically, the movement along all the left edges at each layer of the server tree can occur indepen-

dently. Likewise, all the right edges can be processed concurrently. Considering these points while implementing our solution will make it much faster on multi-core systems, which is the case for most server-side hardware.

Since *RouterORAM* may store multiple replicas of each block, its server storage requirement is higher than that of other ORAM schemes. However, with storage costs continuing to decrease [39], this is less of an issue. Additionally, given a fixed access frequency, the server storage should not exceed a certain constant factor. When a client accesses certain blocks more frequently than others, it keeps multiple replicas of those more frequently accessed blocks. Consequently, for a given access frequency, the client tends to access other blocks much less often. This allows for a reduction in the number of replicas for the less frequently accessed blocks. Thus, with a steady average access rate, the total number of replicas ($\#^{\mathrm{Re}}$) remains constant, irrespective of the access pattern, and the server storage needs are proportional to that total.

In *RouterORAM*, the client must maintain the entire position map locally as with other ORAM schemes. Theoretically, this requires $O(\mathsf{N})$ of client storage. However, in practice, this represents a relatively small amount of storage, as each block identity can be represented by a compact integer. For example, when outsourcing 100GB of data divided into 4096-byte blocks to the server, the client needs to store less than 100MB locally for the position map. While this level of client storage is manageable in most situations, it may not be universally applicable. In such cases, the position map can also be stored on the server and accessed recursively. Consequently, similar to other ORAM schemes, the complexity increases by a factor of $O(\log \mathsf{N})$.

Regarding non-targeted Access Patterns

As discussed in Section 2.4.6, the write complexity is directly proportional to the number of existing replicas of the written block (i.e., $\#(\mathsf{a})$). Ideally, $\#(\mathsf{a})$ should be a small constant for a bursty read-dominated access, which is the targeted usage scenario of *RouterORAM*. As a result, the write complexity is expected to be $O(1)$, in *RouterORAM*. However, if the client opts to use *RouterORAM* for a write-dominated access pattern, the performance of *RouterORAM* will become inferior compared to other ORAM schemes.

Theoretically, for a write-dominated access pattern, the client's choice for $\#(\mathsf{a})$ can escalate to $O(\mathsf{N})$. In this scenario, the complexity of the write operation would similarly rise to $O(\mathsf{N})$. Specifically, selecting $\#(\mathsf{a}) = O(\mathsf{N})$ is reasonable only if the client generates continuous (rather than bursty) high traffic of write operations directed at only one specific block a. We acknowledge that under these conditions, the efficiency of the write opera-

tion in *RouterORAM* diminishes, although the read operation will still maintain an $O(1)$ complexity.

Although using *RouterORAM* for a non-bursty access pattern is not limited, its effectiveness will degrade. To support continuous access, especially with a very high access rate, $\#^{\text{Re}}$ may be much larger than N. In this case, along with the server storage, the actual time to complete one round of background routing will also increase. To accommodate those large amounts of replicas, each bucket of the server tree must have a much larger number of slots. Therefore, *RouterORAM* will no longer be as effective as in the bursty scenario.

2.7.3 Other Applications of *RouterORAM*

Interestingly, the acronym ORAM originates from **O**blivious **R**andom **A**ccess **M**achine [16], rather than *oblivious random access memory*. The original purpose of the ORAM research was to ensure that an honest but curious executor of an obscured program cannot learn any information from the access pattern [16], which is an inherent side-channel effect of program execution. This consideration is particularly relevant in the realm of TEE-based execution. While TEEs effectively conceal both data and code, they cannot hide access patterns. This leakage of access patterns is the primary source of several side-channel attacks targeting TEEs [40].

ORAM can be advantageous in this context, and actually several research efforts have tried to combine ORAM with TEE to tackle this challenge [41, 42]. However, it is important to note that all these solutions still experience an $O(\log \mathsf{N})\times$ slowdown due to the established lower limit of ORAM. Nonetheless, substituting the ORAM schemes used in these solutions with *RouterORAM* could lead to a substantial improvement in execution speed in most of the use.

This observation holds true because, in most instances of program execution, the volume of load operations significantly surpasses that of store operations [43]. This conclusion aligns seamlessly with the targeted access pattern of *RouterORAM*. As a result, *RouterORAM* has the potential to substantially enhance the speed of side-channel leakage-free program execution by reducing the slow-down of these executions from $O(\log \mathsf{N})\times$ to $O(1)\times$.

Chapter 3

Efficient PIR schemes to Protect Access Patterns in a Public Database

In this chapter, we shift our focus to public databases. The information contained within these databases is available to anyone, enabling multiple concurrent clients' access, while allowing the server to monitor all the accesses. Since both the content and access to these databases are open for everyone, including the server, hiding access patterns becomes significantly more challenging.

3.1 Motivation: Efficient Protection of Access Patterns in Public Databases

We regularly access such public databases. For instance, we might explore e-commerce websites before making purchases and browse through video libraries. Although nothing confidential resides within these databases, accessing them still poses a privacy risk, since an honest-but-curious server could readily observe which data items are being accessed by specific clients. By monitoring this information over time, the server can ascertain several private details about the clients' interests [44–46] and potentially exploit this information for malicious purposes. Therefore, it is important to find a way for the clients to retrieve a specific item from a public database, without revealing to the server which item they are accessing, even if the server is delivering that item, itself.

A simple solution to this problem is for each client to download the entire database and access the specific item needed locally, discarding the rest. Using this approach, the server

cannot determine which particular item is being accessed, thereby protecting the clients' access patterns. However, it is obvious that this approach is quite costly. The computation and communication overhead for each database access amounts to $O(\mathsf{N})$, where N represents the database size.

Chor et al. first introduced the term Private Information Retrieval (PIR). An effective PIR scheme conceals the clients' read-only access patterns to a public database from the server. Their PIR solution [47] employs two non-colluding servers, which are individually honest-but-curious. In their scheme, a PIR client requests both the servers to provide a constant-size response based on approximately half of the database items.

These database items appear random to the servers. After receiving the responses from both the servers, the client combines them locally to decode the intended item. While this method reduces the communication cost to a constant amount, the combined server computation cost remains $O(\mathsf{N})$. This raises an important question: Is there any efficient way to safeguard the clients' access patterns while interacting with the public database? In the following chapters, we propose a sequence of schemes to reduce this overhead.

3.2 *Basic Scheme:* Reducing Server Overhead to $O(\sqrt{\mathsf{N}})$

Now we discuss our first PIR scheme, which reduces server overhead down to $O(\sqrt{\mathsf{N}})$ while keeping client overhead at $O(1)$. Before discussing our proposal, we review the related work that is specifically aimed at reducing server overhead.

3.2.1 Related Work for *Basic Scheme*, for reducing server overhead

Over the next twenty-five years from the initial work on PIR, several schemes have been proposed with the intention of reducing the server computation overhead. However, none of them were able to reduce server overhead below $O(\mathsf{N})$, without creating privacy issues.

Theoretical bound

In 2004, Beimel et al. focused on how the data is stored in the database. In the conventional *balls-and-bins* model, without any specialized encoding or preprocessing, the database is viewed as a set of *bins*, with each data block represented as a *ball*. The server operates

as a passive storage device and is only allowed to store a single ball within a single bin. Whenever requested, it provides the requested ball from the designated bin. They proved that in this balls-and-bins model, total server computation cannot go below $O(\mathsf{N})$ [48].

Possibility of Reducing Server Computation

Beimel et al., in the same paper, introduced a new theoretical direction: *PIR with pre-processing*. In this model, the server(s) (or both the server(s) and the clients) may pre-perform some computations earlier to reduce the online computation overhead. A key aspect of this pre-processing approach is that it requires additional storage to maintain the pre-computed results. However, modern storage costs have decreased to make this more attractive [39].

Therefore, this pre-processing model has gained immense popularity in PIR research, and several sub-research directions have been proposed. For example, researchers have studied: whether the pre-processing is global [49] or client-specific [50], whether the pre-processing task is required to be done once [51] or periodically [52], etc. Recently, Zhou et al. proposed a PIR solution with periodic pre-processing, PIANO [53], which achieves amortized $\widetilde{O}(\sqrt{\mathsf{N}})$[1] server computation. An empirical analysis of the implementation of their scheme shows very impressive online performance.

However, in their scheme, along with the server, the client must also perform $\widetilde{O}(\sqrt{\mathsf{N}})$ computation/communication per query, on average. Moreover, the client is required to download $O(\mathsf{N})$ amount of pre-processed information (also known as a "hint") before issuing any actual query. This poses a practical challenge, particularly for devices with limited resources, such as mobile phones. In these cases, it might have been more beneficial to use the trivial solution of downloading the entire database instead.

PIANO is a single-server solution, and it has been proven that for any single-server PIR scheme that incorporates pre-processing, the expected online time, denoted as t, must satisfy the condition $\mathsf{t} \times \mathsf{r} = \Omega(\mathsf{N} \log \mathsf{N})$ [54]. Here, r represents the size of the "hint" that the client is required to download. More concerningly, the lower bound remains applicable even if the scheme operates with an error probability of $\frac{1}{\mathsf{n}^2}$.

On the other hand, no known lower bound exists for a pre-processing-based PIR scheme utilizing multiple servers. In fact, there exists a multi-server pre-processing based PIR scheme [51] where $\mathsf{t} + \mathsf{r} = \widetilde{O}(\sqrt{\mathsf{N}})$. This makes the multi-server, pre-processing-based PIR scheme an interesting research direction for more practical PIR schemes.

[1]The notation $\widetilde{O}(\cdot)$ hides arbitrary polylogarithmic factors

Utilizing ORAM to achieve efficient PIR

In the context of access pattern privacy, Oblivious RAM (ORAM) is another important cryptographic primitive that we previously introduced in Chapter 2. Interestingly, ORAM has a much lower computation overhead [15] than PIR. However, the detailed usage scenarios of PIR and ORAM are different. An ORAM hides a *single* client's *read-write* access pattern for a remotely held *private* database. In contrast, PIR is about hiding *multiple* clients' *read-only* access patterns for a remotely held *public* database.

In an ORAM, unlike PIR, each data item remains encrypted within the database. The client periodically re-encrypts the data items and also shuffles their locations. This makes linking specific data items impossible over time, either by location or content. The client locates the required item's ciphertext, using a locally-stored item-position mapping, retrieves the item and then also decrypts it locally. The combination of encryption and shuffling allows the ORAM client to hide access patterns, with much lower computation overhead than PIR.

However, simply using an ORAM-like strategy, in a multi-client scenario to solve the PIR problem, poses several challenges. The ORAM client holds the secret decryption key and the secret position map. Unsurprisingly, implementing a PIR system by sharing those secrets among multiple PIR clients makes the scheme vulnerable to a single dishonest client. Indeed, active security [55] cannot be achieved, which is a common requirement for PIR systems. This active security protects against *fake clients*, who might help the server to determine the access patterns of other legitimate PIR clients.

Ostrovsky and Shoup introduced a Distributed ORAM (DORAM) scheme [55], in which multiple non-colluding servers replace a single ORAM server. They demonstrated how this DORAM could be utilized in a multi-client environment. Their scheme splits a traditional ORAM client's secrets (i.e., the decryption key and position map) across the non-colluding DORAM servers, ensuring that no server can independently access any information about them. The ORAM client now exists as a virtual entity across the participating servers, with shuffling and encryption/decryption activities implemented using multi-party computation (MPC) techniques [56].

As a result, multiple external PIR clients can now be supported, interacting with the virtual ORAM client service, implemented over the DORAM servers, utilizing MPC. This virtual ORAM client retrieves the requested item and delivers it to the external party, the PIR-client. None of the servers gain knowledge of the PIR request or response, as communication between the PIR client and the participating servers occurs through MPC. This scheme also ensures that individual server secret shares remain concealed from the PIR client, thereby protecting against potential fake clients.

Following the initial research, the research focus shifted to enhancing DORAM to achieve access pattern privacy in MPC. We have not encountered any additional studies that leverage DORAM specifically for the development of PIR. However, we observe a connection between *PIR achieved through DORAM* and *PIR with pre-processing*.

Specifically, PIR via DORAM can be viewed as a particular case of multi-server, global pre-processing-based PIR. In this context, the pre-processing phase entails the encryption and shuffling of the entire database. As there is no established lower bound on the server workload for multi-server, pre-processing-based PIR, we are motivated to develop a more practical PIR scheme that leverages the DORAM strategy.

3.2.2 Overview: A Three Server Architecture with Shuffling and Homomorphic Evaluations

In our first PIR scheme, we utilize three non-colluding servers, namely $\mathsf{S}_\alpha, \mathsf{S}_\beta$, and S_γ, which are individually honest-but-curious. The data items are stored and accessed in fixed-size blocks of B bits from the storage server, S_α. While the content of the blocks does not need to be concealed from the storage server, our solution stores blocks in (randomized) encrypted and shuffled format in the storage server to hide access patterns.

Our primary concept is to route all PIR requests from multiple PIR clients through a single ORAM client, thereby ensuring privacy of access patterns from the storage server. In our scheme, the ORAM client itself remains uninformed, as we do not treat it as a singular entity. Instead, we distribute the responsibilities and confidential information associated with the ORAM client across two non-colluding servers, S_β and S_γ.

This distribution ensures that neither server can independently decipher the actual PIR request. S_β primarily maintains the secret position map, π, which dictates the arrangement of the shuffled database residing in S_α. Additionally, S_β holds the decryption key, $\mathsf{sk_F}$, and functions as the decryption oracle. On the other hand, S_γ is responsible for determining the shuffled location corresponding to the PIR request.

Typically, π is stored as an array of shuffled locations. Since a standard ORAM client knows the plaintext index of the requesting block, I, it can determine the corresponding shuffled location, L, by referencing the Ith index of $\pi[]$. However, in our scenario, it is crucial that I remains confidential from all three servers. But the requested block's shuffled location, $\mathsf{L} = \pi[\mathsf{I}]$, still needs to be determined correctly before accessing the block ciphertext from the storage server. We harness the homomorphic evaluation to solve this challenge.

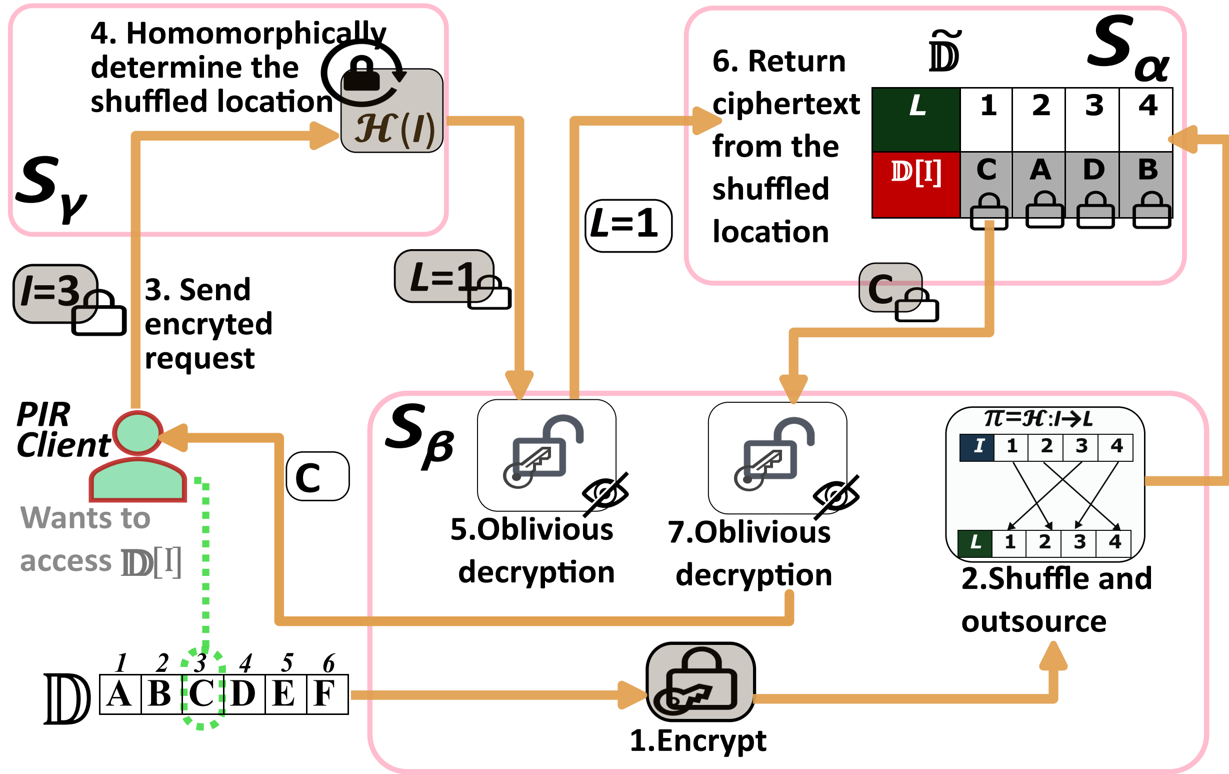


Figure 3.1: Overview of *Basic Scheme*

However, the array lookup process itself may inadvertently reveal access patterns, even when performed under the cloak of homomorphic evaluation. Consequently, the honest-but-curious S_γ could infer I from the access pattern generated during the position map lookup. To mitigate this risk, instead of representing the position map π as an array, we utilize a constant-time, collision-resistant, keyed, one-way function, $\mathcal{H}$. Thus, instead of homomorphic lookup process, S_γ calculates the shuffled location by homomorphically evaluating $\mathsf{L} = \mathcal{H}(\mathsf{I})$. Figure 3.1 provides a rough overview of our solution in a sample scenario, while Table 3.1 illustrates the distribution of secrets across three servers.

Before it starts processing requests from the PIR clients, S_β initializes the encrypted and shuffled database, $\widetilde{\mathbb{D}}$, in S_α (in Figure 3.1, Stage 1 and Stage 2). During fetching a block, the PIR client, $\mathbb{C}$, sends $\bar{\mathsf{I}}$, the requested block's index encrypted with a public key FHE scheme, to S_γ (Stage 3). Even though the original index remains confidential, S_γ can compute $\bar{\mathsf{L}}$, which is the ciphertext of the corresponding shuffled location, by performing a homomorphic evaluation of $\mathcal{H}(\mathsf{I})$ (Stage 4).

Table 3.1: Secret Allocations to the Three Servers

Server	Known	Unknown	Comments
S_α	L	$\mathsf{I}, \mathcal{H}, \mathsf{sk_F}$	$\mathcal{H}$'s key is unknown. Hence cannot determine I from L.
S_β	$\mathcal{H}, \mathsf{sk_F}$	I, L	Does not see $\bar{\mathsf{I}}$ or $\bar{\mathsf{L}}$. Thus, having $\mathcal{H}, \mathsf{sk_F}$ does not help.
S_γ	-	$\mathsf{I}, \mathcal{H}, \mathsf{L}, \mathsf{sk_F}$	Only able to homomorphically determine $\bar{\mathsf{L}}$ from $\bar{\mathsf{I}}$.

Yet, to return the proper item, S_α must know the plaintext location, L. Hence, S_β, the owner of the FHE decryption key, must decrypt $\bar{\mathsf{L}}$ and return its plaintext value to S_α (Stage 5). This raises privacy concerns since S_β also knows the mappings of all the blocks. As a result, after decryption, S_β may use L to perform inverse mapping to determine I. Our scheme mitigates this problem by introducing the concept of *oblivious decryption* (details in Section: 3.2.6), which ensures that S_β cannot know the actual plaintext value of L, even if it decrypts $\bar{\mathsf{L}}$.

Consequently, S_α can retrieve the ciphertext of the requested block from the correct location (Stage 6) in the shuffled database and send it to S_β. Following that, S_β carries out another oblivious decryption on the fetched ciphertext (Stage 7), and delivers the plaintext block content to $\mathbb{C}$.

Notice that only S_α knows which locations are touched in the shuffled database during processing PIR requests. As a result, although S_α cannot know which block is being fetched, it may try to figure out whether a block is requested more than once or not, by observing whether any shuffled location is touched more than once or not. To safeguard against this, our solution maintains a cache containing IND-CPA secure ciphertexts of all the items that have been accessed previously.

We refer to this dedicated cache area as the *shelter* within our framework (not shown in the figure for simplicity). When a request is made, our protocol begins by scanning the entire shelter, $\mathsf{sh}[]$, to locate the desired item. Only if not found, the actual shuffled location, L, is delivered to S_α. Conversely, if the item is found within the shelter, our protocol delivers a previously unseen dummy location, L', to S_α.

We integrate ORAM with FHE to realize a PIR with reduced overhead. We adopt the square-root ORAM [15] as the foundational ORAM strategy for our approach. The top-level flow of our scheme is illustrated in Algorithm 3.1. In our approach, the PIR clients aim to retrieve items that are organized into fixed-size blocks within a public database, $\mathbb{D}$. However, to obscure the access patterns, our protocol retrieves the requested item

from the encrypted and shuffled database, $\widetilde{\mathbb{D}}$. These extra blocks are required as part of the obfuscation of access patterns. In this $\widetilde{\mathbb{D}}$, we incorporate an additional $\sqrt{\mathsf{N}}$ *dummy* blocks alongside the N *real* blocks from $\mathbb{D}$. Similar to the square-root ORAM, our scheme also reshuffles $\widetilde{\mathbb{D}}$ after each *epoch*. In line with square-root ORAM, we define an epoch as consisting of $\sqrt{\mathsf{N}}$ PIR requests; beyond this point, the access pattern may become susceptible to leakage.

Algorithm 3.1 Top-level flow of our $O(\sqrt{\mathsf{N}})$-PIR solution

```
1: One-time Initialization
2: for Each epoch do
3:     (Re-)encrypt and (re-)shuffle entire D̃
4:     for i ∈ [1, √N] do
5:         Fetch ciphertext of block I_i from the Server Storage ▷ Either from D̃ or from sh[]
6:         Return plaintext response to the PIR-client
7:     end for
8: end for
```

During the ith access within an epoch, the PIR client requests the block having index $\mathsf{I_i}$. Our scheme retrieves the ciphertext of the requested block from either the corresponding shuffled location, $\mathsf{L_i}$, within $\widetilde{\mathbb{D}}$, or from the shelter, which stores ciphertexts of the previously accessed blocks. Importantly, our method ensures that it does not disclose to any server whether the retrieval is made from the shelter or the shuffled database. Subsequently, the protocol returns the relevant plaintext content to the PIR, again without revealing this information to any servers.

We will discuss each individual step mentioned in Algorithm 3.1 in detail in the subsequent sections. Before this, we will cover the storage details. We summarize the notation used in our solution in Table A.2.

3.2.3 Storage Details

Each block $\mathsf{I} \in [1, \mathsf{N}]$, is stored as an encrypted pair $\overline{\{\mathsf{I}, \mathbb{D}[\mathsf{I}]\}}$ within the server-side storage, where $\mathbb{D}[\mathsf{I}]$ denotes the data content of the block having index I. The index part, I, is $\log \mathsf{N}$-bits long, and the size of $\mathbb{D}[\mathsf{I}]$ is B bits. The server-side storage means the shuffled database, $\widetilde{\mathbb{D}}$ and the shelter, $\mathsf{sh}[]$. Both $\widetilde{\mathbb{D}}$ and $\mathsf{sh}[]$, are divided into buckets, with each bucket holding one encrypted pair.

$\widetilde{\mathbb{D}}$ consists of N encrypted pairs corresponding to actual data blocks of $\mathbb{D}$. To hide the access pattern, $\widetilde{\mathbb{D}}$ must also contain at least $\sqrt{\mathsf{N}}$ dummy pairs. However, to take certain advantage (details in section 3.2.5), we increase the number of dummy pairs so that the total number of buckets, M, in $\widetilde{\mathbb{D}}$, becomes a prime number. Specifically, we choose M to be the next prime number after $(\mathsf{N} + \sqrt{\mathsf{N}})$.

$\widetilde{\mathbb{D}}$ is entirely kept in S_α. However, S_β periodically re-encrypts and re-shuffles it. $\mathsf{sh}[]$ is maintained exclusively by S_γ and serves as a cache that stores the encrypted pairs retrieved from $\widetilde{\mathbb{D}}$ during the current epoch. After each PIR request, S_γ appends a new encrypted pair to $\mathsf{sh}[]$. Consequently, the size of the shelter can increase up to $\sqrt{\mathsf{N}}$. The notation $\mathsf{sh}^{\mathsf{i}}[]$ represents the state of the shelter following the processing of the ith PIR request.

3.2.4 One-time Initialization

As pointed out in Algorithm 3.1, initialization is a one-time process. During this initialization process, S_β first determines a prime number $\mathsf{M} > (\mathsf{N} + \sqrt{\mathsf{N}})$ and chooses a random generator $\mathsf{g} \in \mathbb{Z}_\mathsf{M}$. Next, M and g are published since other parties will require them while executing our protocol. An IND-CPA-secure fully homomorphic encryption scheme, FHE, performs all of the encryptions in our solution. S_β generates a key-pair $(\mathsf{pk_F}, \mathsf{sk_F}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$ and publishes the public key $\mathsf{pk_F}$, keeping the decryption-key, $\mathsf{sk_F}$, secret. The public key, $\mathsf{pk_F}$, serves as both the encryption key and the homomorphic evaluation key.

3.2.5 (Re-)encrypt and (re-)shuffle entire $\widetilde{\mathbb{D}}$

At the beginning of each epoch, $\widetilde{\mathbb{D}}$ is re-encrypted and re-shuffled. Dummy pairs are stored as $\overline{\{0, \{0\}^\mathsf{B}\}}$, because 0 is not a valid block index. Since FHE is IND-CPA-secure, the ciphertext of real and dummy pairs are indistinguishable from each other.

We represent the secret position map as a constant-time function to safeguard against access pattern leakage during the shuffled location lookup. This function must be collision-resistant, as each bucket can store only one pair. Additionally, it must be a one-way function; otherwise, S_α, could potentially back-track the index of the requested block from the observed shuffled location. We use the properties of $\mathbb{Z}_\mathsf{M}$ in this context. Since we choose M as a prime number, the exponential of the block index, $(\mathsf{g}^\mathsf{I} \mod \mathsf{M})$, serves as a collision-resistant function, where g is the generator of $\mathbb{Z}_\mathsf{M}$.

However, whether $\mathsf{g}^{\mathsf{I}} \bmod \mathsf{M}$ is one-way or not depends on the size of the prime number M. To learn the preimage, the adversary needs to break the discrete logarithm problem, which is computationally difficult, only if M is large enough. For practical purposes, having a very large M is not feasible, as it will excessively increase the required storage space in S_α. Ideally, the amount of required storage in S_α should be in $O(\mathsf{N})$.

If, instead, we use a keyed-hash function, $\mathcal{H} : \mathcal{K} \times [1, \mathsf{N}] \rightarrow \mathbb{Z}_{\mathsf{M}}$, *and* keep the key secret, then $\mathcal{H}$ becomes preimage-resistant, irrespective of the size of M. Thus, we define the keyed-hash function as: $\mathcal{H}(\langle \mathsf{e}, \mathsf{s} \rangle, \mathsf{I}) = (\mathsf{g}^{\mathsf{I}+\mathsf{e}} + \mathsf{s}) \bmod \mathsf{M}$, which determines the shuffled location L of the block I. Here, e and s serve as the components of the secret key and remain unchanged throughout the epoch. At the conclusion of each epoch, S_β updates e and s to enable a new shuffling.

We specifically utilize the formulation $\mathcal{H}(\langle \mathsf{e}, \mathsf{s} \rangle, \mathsf{I}) = (\mathsf{g}^{\mathsf{I}+\mathsf{e}} + \mathsf{s}) \bmod \mathsf{M}$ because it guarantees that if e is added to I before the exponential operation, even if an adversary successfully solves the discrete logarithm problem after observing L, they can only determine $(\mathsf{I} + \mathsf{e})$. Since e remains secret, the adversary (S_α in this situation) cannot deduce the value of I. However, in a given epoch, by observing two touched locations L_i and L_j, the adversary can infer $(\mathsf{I}_\mathsf{i} + \mathsf{e})$ and $(\mathsf{I}_\mathsf{j} + \mathsf{e})$, which enables it to extract information about the difference $(\mathsf{I}_\mathsf{i} - \mathsf{I}_\mathsf{j})$. To mitigate this risk, we added an additional secret, s, after completing the exponential operation.

Figure 3.2 represents the sequence diagram of the entire shuffling process. At the beginning of the shuffling process, S_β first chooses the secret parameters, e, and s, to define $\mathcal{H}$, which determines the shuffling for the entire epoch. Subsequently, it computes g^{e} and sends FHE-encrypted $\overline{\mathsf{g}^{\mathsf{e}}}, \overline{\mathsf{s}}$ to S_γ. Later, during the fetching phase, these ciphertexts allow S_γ to homomorphically compute $\mathcal{H}()$.

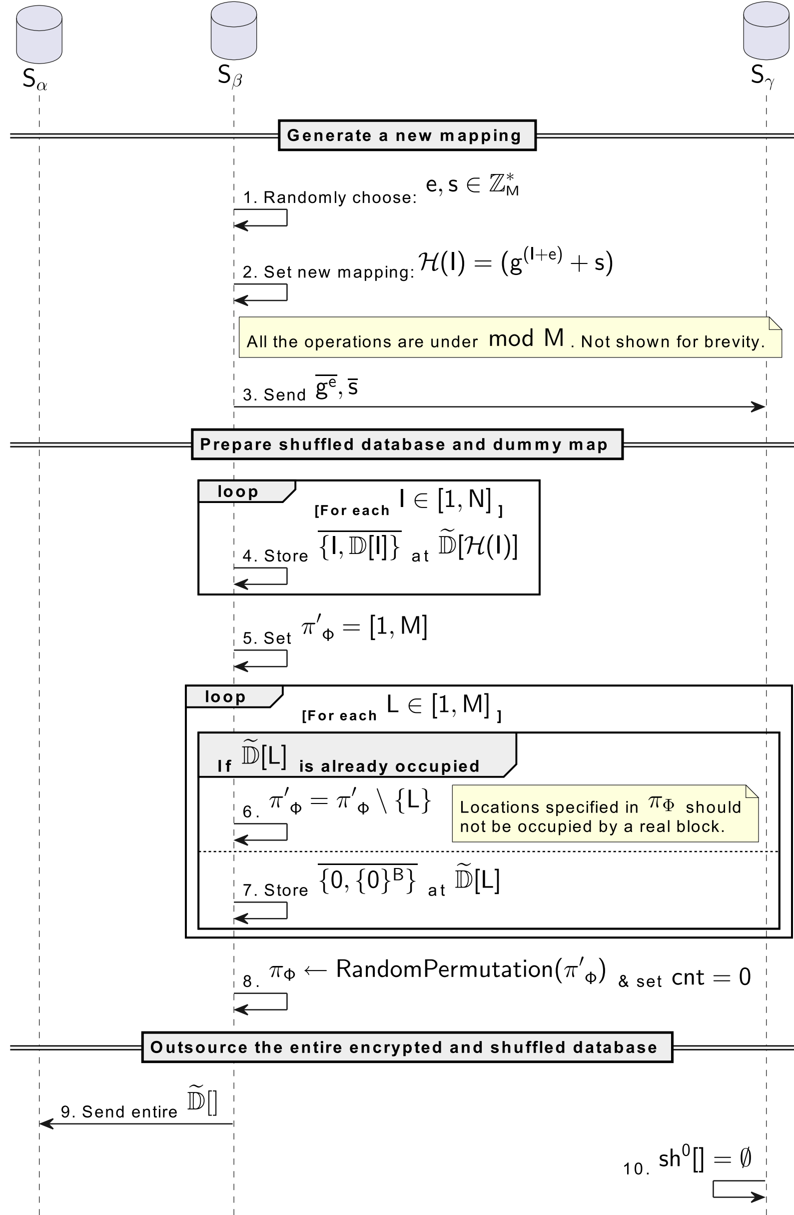


Figure 3.2: Shuffling Process Sequence Diagram

S_β then locally prepares $\widetilde{\mathbb{D}}$. For each block $I \in [1, N]$, the corresponding plaintext pair is encrypted with FHE, and the encrypted pair, $\overline{\{I, \mathbb{D}[I]\}}$, is placed at the shuffled location, determined by $\mathcal{H}$. After placing all the real pairs, $(M - N)$ empty buckets exist within

$\widetilde{\mathbb{D}}$. S_β fills those with encrypted dummy pairs and initializes the dummy map, π_Φ, with a random permutation of those $(\mathsf{M}-\mathsf{N})$ dummy locations. Finally, S_β sends the entire $\widetilde{\mathbb{D}}$ (consisting of both reals and dummies) to S_α. During this phase, S_γ also prepares itself, by clearing the existing shelter contents.

3.2.6 Fetch Block Ciphertext from the Server Storage

After receiving the ith PIR request, S_γ scans the entire shelter. If the requested block, $\mathsf{I_i}$, is already accessed within the current epoch, then it should be present within the shelter. If so, then S_γ obliviously sets $\mathsf{L_i}$ with an untouched dummy location within $\widetilde{\mathbb{D}}$. On the other hand, if the block is not found within the shelter, S_γ obliviously sets $\mathsf{L_i}$ with $\mathcal{H}(\mathsf{I_i})$, which is the location that actually contains block $\mathsf{I_i}$. Next, S_α learns $\mathsf{L_i}$ and returns the encrypted pair stored at that location. Regardless of whether $\mathsf{I_i}$ is a repeated request, our protocol guarantees that S_α never observes any location within $\widetilde{\mathbb{D}}$ being accessed more than once.

While determining $\mathsf{L_i}$, S_γ carries out the entire process without any awareness of specific details: it remains unaware of the value of $\mathsf{I_i}$ and does not know whether the determined value of $\mathsf{L_i}$ refers to an actual block or a dummy one. To achieve this, we utilize homomorphic evaluation. Specifically, we apply the homomorphic selection function discussed earlier in the Section 2.4.7. This function is defined as $\overline{\mathsf{C}} = \mathsf{Select}(\overline{\mathsf{bit}}, \overline{\mathsf{A}}, \overline{\mathsf{B}}) = \overline{(1-\mathsf{bit})\cdot\mathsf{B}+\mathsf{bit}\cdot\mathsf{A}}$, which operates on the encrypted condition-checking $\overline{\mathsf{bit}}$ to homomorphically select one of A or B. Figure 3.3 illustrates the complete details of the ciphertext fetching phase.

To initiate the request processing, the PIR-client, $\mathbb{C}$, computes FHE ciphertexts $\overline{\mathsf{I_i}}$ and $\overline{\mathsf{g}^{\mathsf{I_i}} \bmod \mathsf{M}}$ using the public parameters (g, M and $\mathsf{pk_F}$) and sends them to S_γ. Next, S_γ scans through the entire shelter according to the algorithm $\mathsf{OScan}()$ (Algorithm 3.2) to obliviously check whether the PIR request matches with any of the stored pairs within the shelter. It is to be noted that, since FHE is IND-CPA secure, S_γ observes that $\overline{\mathsf{fnd_{sh}}}$ & $\overline{\mathsf{pair_{sh}}}$ change during each iteration of the $\mathsf{OScan}()$ execution, regardless of whether the underlying plaintext remains unaltered.

Upon the completion of $\mathsf{OScan}()$, the ciphertext variable $\overline{\mathsf{fnd_{sh}}}$ takes on the encryption of the bit value 1 only if a match is found. In this scenario, $\overline{\mathsf{pair_{sh}}}$ stores the corresponding matched encrypted pair (Step 2 of Figure 3.3). If no match is identified, both $\overline{\mathsf{fnd_{sh}}}$ and $\overline{\mathsf{pair_{sh}}}$ will retain the ciphertexts representing the bit value 0 and a dummy pair, $\{0, \{0\}^{\mathsf{B}}\}$, respectively. In this context, it is important to clarify that the shelter may include multiple replicas of the same block if that block is accessed more than once within an epoch. However, this is not an issue. The $\mathsf{OScan}()$ algorithm will function as intended, regardless of the number of replicas present within the shelter.

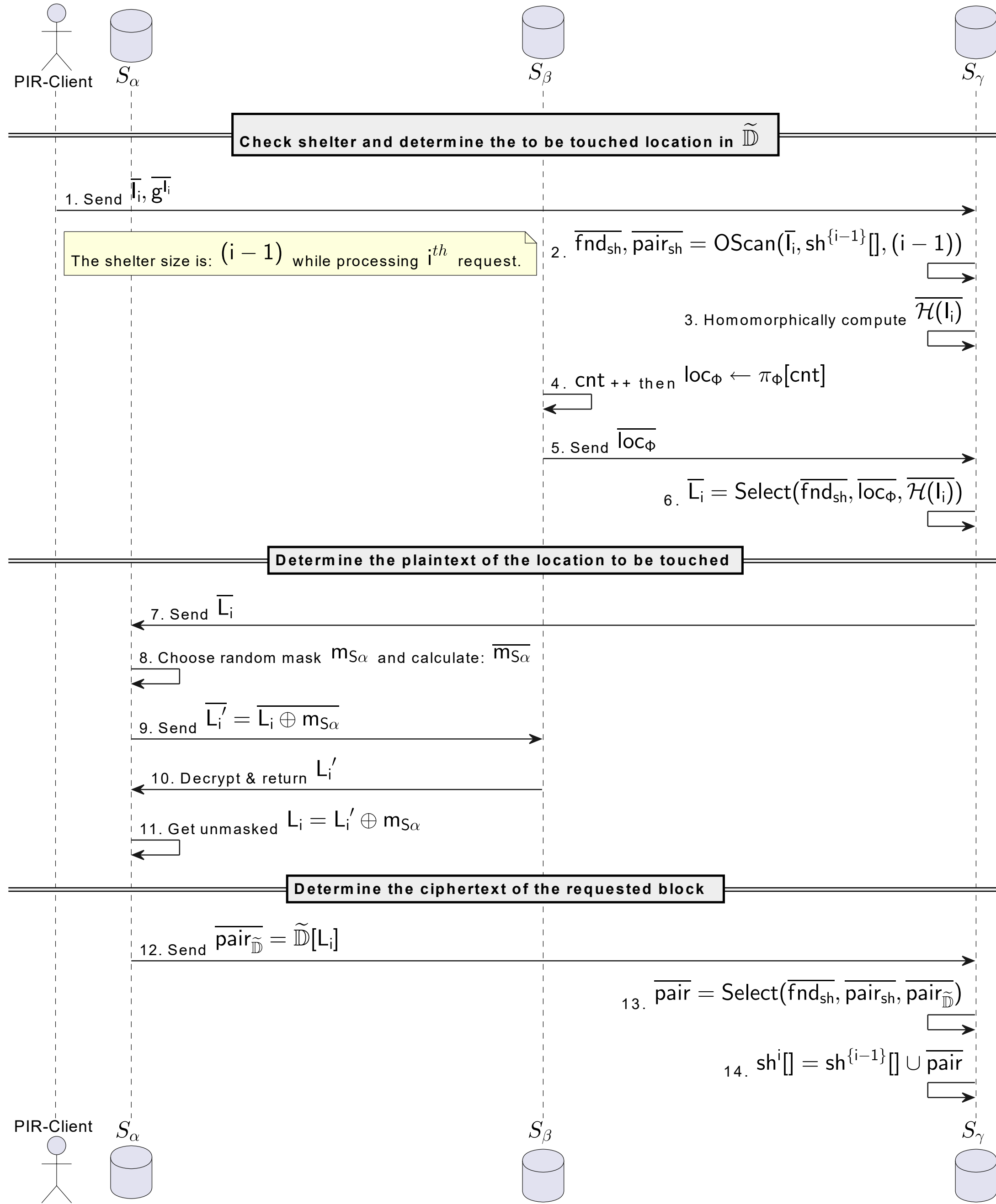


Figure 3.3: Fetch Block Ciphertext Sequence Diagram

Algorithm 3.2 $\mathsf{OScan}(\overline{\mathsf{I_i}}, \mathsf{sh}^{\{i-1\}}[], \mathsf{CurShSz})$

1: $\overline{\mathsf{pair_{sh}}} = \overline{\{0, \{0\}^B\}}$ ▷ Initialize with dummy
2: **for** $j \in [1, \mathsf{CurShSz}]$ **do** ▷ Scan all shelter pairs
3: $\quad \overline{\mathsf{fnd_{sh}}} \leftarrow \overline{(\mathsf{sh}^{\{i-1\}}[j].\mathsf{idx} == \mathsf{I_i})}$ ▷ Is the requested block index, $\mathsf{I_i}$, present here?
4: $\quad \overline{\mathsf{pair_{sh}}} \leftarrow \mathsf{Select}(\overline{\mathsf{fnd_{sh}}}, \overline{\mathsf{sh}^{\{i-1\}}[j]}, \overline{\mathsf{pair_{sh}}})$ ▷ $\mathsf{pair_{sh}}$ is changed to $\mathsf{sh}^{\{i-1\}}[j]$, only if it is present
5: **end for**
6: **return** $(\overline{\mathsf{fnd_{sh}}}, \overline{\mathsf{pair_{sh}}})$

Next, S_γ homomorphically computes $\overline{\mathcal{H}(\mathsf{I_i})}$. Since $\mathcal{H}(\mathsf{I_i}) = (\mathsf{g}^{\mathsf{I_i}+\mathsf{e}} + \mathsf{s}) \bmod \mathsf{M}$, S_γ first homomorphically multiplies $\overline{\mathsf{g^e}}$ (received earlier from S_β, during the shuffling phase) with the newly received $\overline{\mathsf{g}^{\mathsf{I_i}}}$ and then adds $\bar{\mathsf{s}}$ to that to obtain $\overline{\mathcal{H}(\mathsf{I_i})}$. S_γ also receives $\overline{\mathsf{loc}_\Phi}$ from S_β (Step 5 of Figure 3.3), the ciphertext of a new dummy location chosen from the dummy map, π_Φ. Now, depending on whether the requested block is found within the shelter (i.e., the plaintext value of $\mathsf{fnd_{sh}}$), S_γ selects either of $\overline{\mathcal{H}(\mathsf{I_i})}$ and $\overline{\mathsf{loc}_\Phi}$ as $\overline{\mathsf{L_i}}$ and sends that to S_α. Although our scheme does not allow revealing any of the plaintexts, S_γ can still (obliviously) make the correct selection by executing $\mathsf{Select}(\overline{\mathsf{fnd_{sh}}}, \overline{\mathsf{loc}_\Phi}, \overline{\mathcal{H}(\mathsf{I_i})})$ (Step 6).

However, to return the ciphertext from the intended location of the shuffled database, S_α must get the received $\overline{\mathsf{L_i}}$ decrypted from S_β, the owner of the decryption key. This raises privacy concerns since S_β also knows the secret parameters e, s. This will allow S_β to determine $\mathcal{H}^{-1}$ and then calculate $\mathsf{I_i}$ as $\mathcal{H}^{-1}(\mathsf{L_i})$ from the newly decrypted value of $\mathsf{L_i}$.

To protect against this, we use the concept of *oblivious decryption*. In which, S_α first encrypts a randomly selected secret mask m_{S_α} by using the public key of FHE. Before requesting for the decryption of $\overline{\mathsf{L_i}}$, S_α homomorphically XORs $\overline{\mathsf{m}_{S_\alpha}}$ with $\overline{\mathsf{L_i}}$ to perform masking (Step 9). Therefore, after decryption, S_β only sees the masked value (i.e., $\mathsf{L_i} \oplus \mathsf{m}_{S_\alpha}$), which information-theoretically hides $\mathsf{L_i}$. However, S_α can easily unmask it and derive $\mathsf{L_i}$, since it already knows the plaintext value of m_{S_α} (Step 11).

Now, S_α retrieves the ciphertext stored at $\widetilde{\mathbb{D}}[\mathsf{L_i}]$ and returns that to S_γ. However, only one ciphertext between this newly retrieved $\overline{\mathsf{pair}_{\widetilde{\mathbb{D}}}}$ and $\overline{\mathsf{pair_{sh}}}$ (earlier returned by $\mathsf{OScan}()$) is the actual ciphertext of the requested block and the other one is dummy. S_γ must determine it correctly but again obliviously. We achieve this, again, by using the $\mathsf{Select}()$ function and utilizing $\overline{\mathsf{fnd_{sh}}}$ as the ciphertext of the selection bit. At the end, S_γ updates the shelter state by appending $\overline{\mathsf{pair}}$ to it (Step 14).

3.2.7 Return Block Plaintext to the PIR-client

By the end of the previous stage, our scheme sets the ciphertext variable, $\overline{\mathsf{pair}}$, with the encrypted pair corresponding to the requested block, $\overline{\{\mathsf{I_i}, \mathbb{D}[\mathsf{I_i}]\}}$, without revealing the request details to any of the three servers. Our scheme must now return the plaintext version of that fetched ciphertext to the PIR client, again, without revealing the information to anyone else. Again, we achieve this using oblivious decryption, and the entire sequence is depicted in the Figure 3.4.

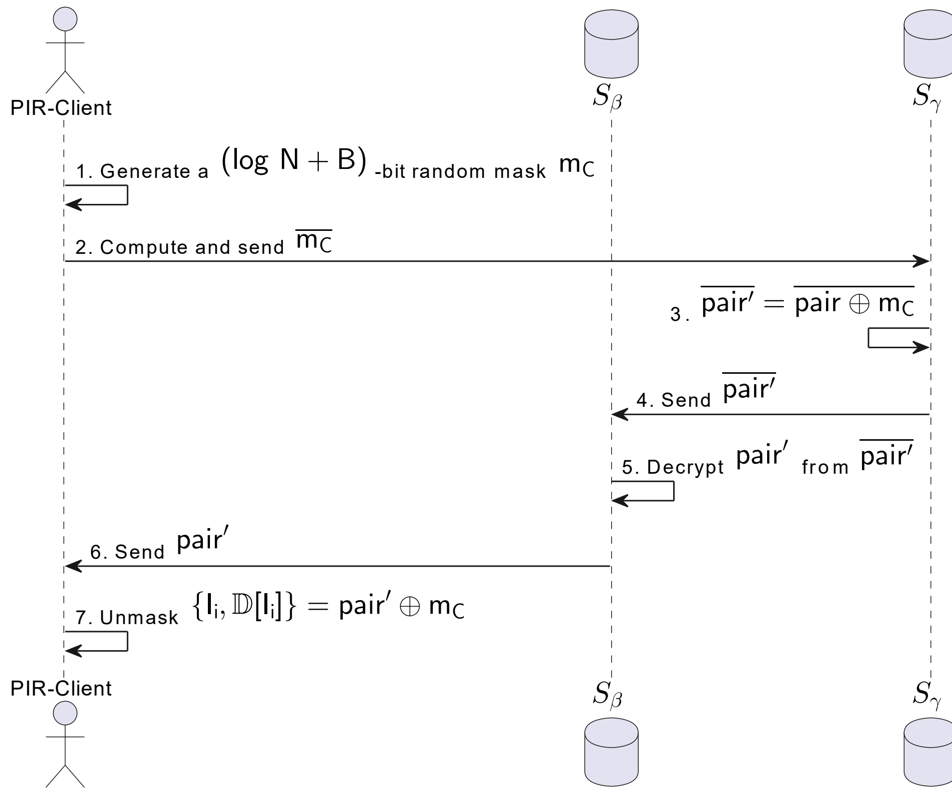


Figure 3.4: Return Block Plaintext Sequence Diagram

Here, $\mathbb{C}$ generates a random secret mask, $\mathsf{m_C}$, having the same size as the retrieved pair. Next, it encrypts the mask with the public key, $\mathsf{pk_F}$, and sends the ciphertext to S_γ. Since S_γ does not have a corresponding private key, it cannot decrypt the ciphertext but can homomorphically mask $\overline{\mathsf{pair}}$ with that. Then, S_γ sends the resulting ciphertext to S_β for decryption. S_β decrypts that ciphertext and returns that to $\mathbb{C}$. After getting $\mathsf{pair} \oplus \mathsf{m_C}$

from S_β, $\mathbb{C}$ XORs the plaintext, m_C. As a result, $\mathbb{C}$ obtains the actual plaintext content of the requested block.

3.2.8 Asymptotic Overhead Analysis

Overall, our solution achieves a PIR scheme with total server computation and communication overhead amounting to $O(\sqrt{\mathsf{N}})$ while keeping client overheads constant. In achieving this overhead reduction, we need to increase the server-side total storage from $O(\mathsf{N})$ to $O(\mathsf{M}) \approx O(\mathsf{N} + \sqrt{\mathsf{N}})$. We summarize our analysis in Table 3.2 and explain in the following text.

Table 3.2: Summary of the overheads

Overhead type	**Client**	S_α	S_β	S_γ	**Total-server**
Computation during request processing	$\boldsymbol{O(1)}$	$O(1)$	$O(1)$	$O(\sqrt{\mathsf{N}})$	$\boldsymbol{O(\sqrt{\mathsf{N}})}$
Amortized computation†	-	$O(1)$	$O(\sqrt{\mathsf{N}})$	$O(1)$	$\boldsymbol{O(\sqrt{\mathsf{N}})}$
Overall computation	$\boldsymbol{O(1)}$	$O(1)$	$O(\sqrt{\mathsf{N}})$	$O(\sqrt{\mathsf{N}})$	$\boldsymbol{O(\sqrt{\mathsf{N}})}$
Communication during request processing	$\boldsymbol{O(1)}$	$O(1)$	$O(1)$	$O(1)$	$\boldsymbol{O}(1)$
Amortized communication†	-	$O(\sqrt{\mathsf{N}})$	$O(\sqrt{\mathsf{N}})$	$O(1)$	$\boldsymbol{O(\sqrt{\mathsf{N}})}$
Overall communication	$\boldsymbol{O(1)}$	$O(\sqrt{\mathsf{N}})$	$O(\sqrt{\mathsf{N}})$	$O(\sqrt{\mathsf{N}})$	$\boldsymbol{O(\sqrt{\mathsf{N}})}$
Total Storage	$\boldsymbol{O(1)}$	$O(\mathsf{M})$	$O(\mathsf{M})$	$O(\sqrt{\mathsf{N}})$	$\boldsymbol{O(\mathsf{M})}$

† Amortized Overhead is: $\frac{\text{Total overhead incurred during per-epoch operations}}{\text{Total number of PIR queries processed in an epoch}}$

Overhead during each request processing

Minimizing clients' efforts is also crucial for a practical PIR scheme. The PIR client has minimal involvement in our scheme (see Figure 3.3 and Figure 3.4) and performs a fixed number of operations, regardless of the size of the public database, N. Regarding local storage requirements, the client, similar to other PIR schemes, must at least generate and consequently store the index of the requested block. Given that the size of this index is

$\lceil \log \mathsf{N} \rceil$ bits, the client's storage technically cannot fall below $O(\log \mathsf{N})$. However, since this value is considerably smaller than the block size (B bits) in practice, we consider the $\log \mathsf{N}$-bit block index as $O(1)$ storage, akin to other PIR schemes.

S_γ is responsible for maintaining the shelter and scanning *all* elements of it during processing each PIR request (see Algorithm 3.2). As a result, both the computation and storage overhead associated with S_γ depend on the current size of the shelter. Since the shelter size increases with each request, reaching up to $\sqrt{\mathsf{N}}$, S_γ incurs a storage overhead of $O(\sqrt{\mathsf{N}})$. S_γ also incurs the average computation overhead of $O(\sqrt{\mathsf{N}})$, while processing each request. In terms of communication complexity, Figure 3.3 and Figure 3.4 illustrate that the number of messages exchanged among $\mathsf{S}_\alpha, \mathsf{S}_\beta, \mathsf{S}_\gamma$, and the PIR client remains constant. Therefore, our solution requires $O(1)$ communication complexity during processing each PIR request.

Overhead for per-epoch operations

Our scheme requires specific operations at the begining each epoch (refer to Section 3.2.5). Although the client does not require any involvement during these pre-epoch operations, an overall $O(\mathsf{M}) \approx O((\mathsf{N} + \sqrt{\mathsf{N}}))$ overhead is needed in terms of both computation and communication for the servers.

S_β is responsible for preparing the entire shuffled database (see Figure 3.2). This database comprises M encrypted pairs. As a result, the total computational overhead for S_β is $O(\mathsf{M})$. Since S_β transfers this entire shuffled database to S_α over the network, the communication overhead for both S_α and S_β also amounts to $O(\mathsf{M})$. Consequently, both S_α and S_β require $O(\mathsf{M})$ in storage. In contrast, aside from emptying the existing shelter, S_γ does not need to perform significant actions in this phase. The shelter is maintained as an array. Hence, emptying the existing shelter can be done without actually deleting all the items. Instead, S_γ just moves the pointer to the beginning of the array. Which means, S_γ incurs only $O(1)$ overhead, during this process.

Since, these overheads occur only once, after processing $\sqrt{\mathsf{N}}$ number of PIR requests, we can compute the amortized value of these overheads as:

$$O\left(\frac{\mathsf{M}}{\sqrt{\mathsf{N}}}\right) \approx O\left(\frac{(\mathsf{N} + \sqrt{\mathsf{N}})}{\sqrt{\mathsf{N}}}\right) = O\left(\frac{\mathsf{N}}{\sqrt{\mathsf{N}}} + 1\right) = O(\sqrt{\mathsf{N}})$$

In conclusion, by analyzing both the per-request and per-epoch overheads, we find that the servers' computation and communication overhead remains at $O(\sqrt{\mathsf{N}})$, while the client overhead stays at $O(1)$.

3.3 *DP-Variant Scheme*: Extending to a Differentially Private Scheme with Even Lower Overhead

While our initial scheme, described in the previous section, achieves the lowest asymptotic server overhead observed to date, it may not be practical for every scenario. This limitation arises primarily because, during the processing of each PIR request, S_γ is required to perform $O(\sqrt{\mathsf{N}})$ homomorphic evaluations within the $\mathsf{OScan}()$ algorithm. Based on the current state of FHE implementations [57], each iteration of the $\mathsf{OScan}()$ algorithm takes approximately three milliseconds. Although we have not yet implemented the full scheme, we conducted a mini-experiment and incorporated the results into our asymptotic analysis. From this[2], we estimate that our initial scheme will yield an end-to-end latency of around 8 seconds to retrieve a 512-bit block from a 100GB database. Although this estimated figure represents an improvement over other PIR schemes, it nonetheless still remains impractical for many scenarios.

Therefore, we want to reduce server overhead further. If absolute privacy is not a strict requirement, we can refine our initial scheme to introduce a mechanism that allows for a trade-off between privacy and the burden on the server. Toledo et al. [60] applied the concept of differential privacy [61] (DP) to PIR, establishing the definition for differentially private information retrieval systems (DP-IR).

For access pattern privacy, the neighborhood is defined over the sequence of requests over the entire epoch. Two neighboring inputs can actually differ in several ways [62]. Toledo et al. used the idea of bounded DP [63]. Specifically, one neighboring input (in this case, an epoch) can be obtained by changing the value of exactly one request of the other epoch. Then, based on this neighborhood definition, they designed an indistinguishability game.

We summarize that indistinguishability game briefly here: $\mathcal{A}$ provides the target PIR client with two request choices, I' and I''. The target PIR client secretly selects one of them and issues that request. The adversary controls all other PIR requests (denoted by $\langle \mathsf{I}^* \rangle$) of the epoch. Because of the execution of the target PIR-client's request and all other

[2] The most costly aspect of handling a PIR request lies in executing the $\mathsf{OScan}()$ algorithm. Our experiments with the OpenFHE library [58] reveal that each iteration of the $\mathsf{OScan}()$ algorithm takes about 3ms when run on a single-core processor (Intel Xeon @2494 MHz). Given a 100GB database, we estimate it contains approximately 1.68 billion blocks, each sized at 512 bits and $\sqrt{\mathsf{N}} \approx 41000$. This means that executing $\mathsf{OScan}()$ with S_γ requires about 123 seconds of CPU time. Most data centers currently employ 16-core CPUs (Intel Xeon @2494 MHz) in most situations [59], so we've assumed that S_γ operates on such a setup, completing its execution in $\sim$7.7 seconds. Additionally, we estimate that all other operations and network transfers should wrap up within 300 milliseconds.

adversarially controlled requests, the PIR-system generates an observable trace $\mathcal{O}$ to $\mathcal{A}$, from which $\mathcal{A}$ aims to guess whether the client has issued I' or I''.

If a scheme is an (ϵ, δ)-private PIR system, then for all possible adversary-controlled requests $\mathsf{I}', \mathsf{I}''$ and $\langle \mathsf{I}^* \rangle$, and for all possible adversarial observations, $\mathcal{O}$, the following must be true:

$$\Pr[\mathcal{O}|\mathsf{I}'] \leq \mathsf{e}^{\epsilon} \times \Pr[\mathcal{O}|\mathsf{I}''] + \delta \tag{3.1}$$

Where ϵ and δ are non-negative constants, with lower values indicating improved privacy. From the adversary's standpoint, it strategically selects $\mathsf{I}', \mathsf{I}''$ and $\langle \mathsf{I}^* \rangle$ to maximize privacy leakage. In this section, we propose a PIR scheme that meets the equation: 3.1 and results in $O(\mathsf{K})$ overhead for the server, where the exact value of K can be adjusted to achieve varying degrees of privacy.

It is worth mentioning that the only randomness in the indistinguishability property comes from the mechanism of our scheme itself. Which, in turn, depends on random numbers independently generated by each server using their respective secret sources. We do not specify any specific randomness generation mechanism. However, we assume that the randomness generation process is secure (e.g., hardware PRG). The servers may also use other random numbers for other purposes, such as cryptographic operations. However, those randoms are generated independently and are not used for the DP mechanism.

3.3.1 Related work Regarding Differentially Private Schemes

Toledo et al. [60] not only established the definition of DP-IR but also proposed multiple DP-IR schemes. All of their schemes utilize multiple servers. The fundamental principle behind these schemes is that the client sends different sets of requests to each server, with each set consisting of multiple fake queries along with, optionally, the actual query.

This approach enables differential privacy with considerably less overhead than $O(\mathsf{N})$. Additionally, they leveraged the co-existence of multiple clients within a PIR system and demonstrated that integrating an anonymous communication channel into their foundational schemes could further reduce overhead, while maintaining the same level of privacy. These solutions reduce individual server's overhead but increase the workload for PIR clients due to the need for fake query generation.

Albab et al. [64] addressed this issue by routing PIR queries through a series of mixnet servers. In this method, each client encrypts their query and sends it to the first server. This server adds several fake query ciphertexts and then forwards the entire batch to

the next server, after shuffling the queries. This process continues, allowing the batch of encrypted and shuffled queries to travel through the chain of servers. Once the batch reaches the final server, all of the queries are decrypted, and the corresponding responses are retrieved. This method is particularly effective when handling large batch sizes that include queries from numerous clients. When the batch size exceeds N, a PIR system can be achieved with $O(1)$ overhead. However, in scenarios with smaller batch sizes, which are quite common, this scheme still incurs an $O(\mathsf{N})$ overhead, even though it reduces the client load.

In their work on DP-IR, Patel et al. [65] presented several significant theoretical findings. They demonstrated that if the PIR system is permitted to provide *incorrect answers* to queries occasionally, it is possible to establish an $O(1)$ overhead DP-IR system without the need for batching, achieving a privacy parameter of $\epsilon = \Omega(\log \mathsf{N})$. However, surprisingly, for any single-server DP-IR system that is required always to deliver the correct answer, regardless of the value of ϵ, the overall computational effort required by the server cannot be reduced below $O(\mathsf{N})$. In contrast, this restriction does not apply in a multi-server setting.

3.3.2 Overview: Reducing Overhead by Allowing Limited re-touching

In our *Basic Scheme*, the overhead of $O(\sqrt{\mathsf{N}})$ arises from two primary factors: a) the need to scan $O(\sqrt{\mathsf{N}})$ items within the shelter for each request processed, and b) the requirement to reshuffle the entire shuffled database, which has a size of $O(\mathsf{N} + \sqrt{\mathsf{N}})$, after processing $\sqrt{\mathsf{N}}$-PIR requests. In our *DP-Variant Scheme*, our objective is to reduce the overhead to $O(\mathsf{K})$ by searching only K (where $\mathsf{K} < \sqrt{\mathsf{N}}$) items within the shelter and performing the costly reshuffling operation less frequently. Since this scheme does not search all shelter items, it may not find the requested block even if it is present within the shelter. Consequently, this scheme may result in multiple accesses to specific buckets within the shuffled database.

If a bucket is re-touched in the shuffled database, an adversary can infer that the same block is being accessed again. To address this issue, we have made a slight adjustment to the storage structure within this scheme. Each bucket of the shuffled database is now divided into Z slots (with $\mathsf{Z} \geq 2$). Each slot is capable of storing one encrypted pair, allowing an entire bucket to accommodate Z encrypted pairs. This modification ensures that accessing the same bucket does not necessarily indicate that the same block has been accessed, thereby reducing potential privacy leakage.

We also take advantage of the fact that, as is generally assumed in PIR systems, the client only reads the database. This keeps the content of each block static. Building on

this principle, instead of retaining a single copy of each block, the shuffled database now contains Z replicas of every individual block, which are shuffled independently. As a result, even if a block needs to be accessed multiple times within an epoch (and is not found in the shelter due to the limited searching operation), it remains possible to avoid touching the same bucket by retrieving a different replica from another bucket within the shuffled database.

The overall flow of this differentially private scheme closely mirrors that of our previous scheme (Algorithm 3.1), with the key difference being an increase in the number of requests processed during an epoch, which has risen from $\sqrt{\mathsf{N}}$ to $\frac{\mathsf{M}\times\mathsf{Z}}{\mathsf{K}}$. In addition to this modification to the top-level flow, the specifics of the individual steps have been refined. The upcoming sections elaborate on the necessary changes, compared with the initial scheme, for the storage structure, shuffling process, and the ciphertext-fetching procedure. It is worth noting that the initialization and plaintext return processes remain unchanged from our *Basic Scheme*.

3.3.3 Differences in the Storage Details

Each bucket in the shuffled database is now organized into Z-slots, with each slot capable of storing an encrypted pair. There is no longer a need to include additional dummy pairs in the shuffled database. As a result, the number of buckets, represented by the prime number M, can now be reduced to a prime number $\geq \mathsf{N}$, instead of the previous requirement of $\mathsf{M} > (\mathsf{N} + \sqrt{\mathsf{N}})$.

On the other hand, the structure of the shelter remains unchanged. This scheme also appends the response ciphertext to the shelter after processing each request. However, the maximum size of the shelter has expanded from $\sqrt{\mathsf{N}}$ to $\frac{\mathsf{M}\times\mathsf{Z}}{\mathsf{K}}$ due to the increase in epoch size in this scheme.

3.3.4 Differences in the Shuffling Process

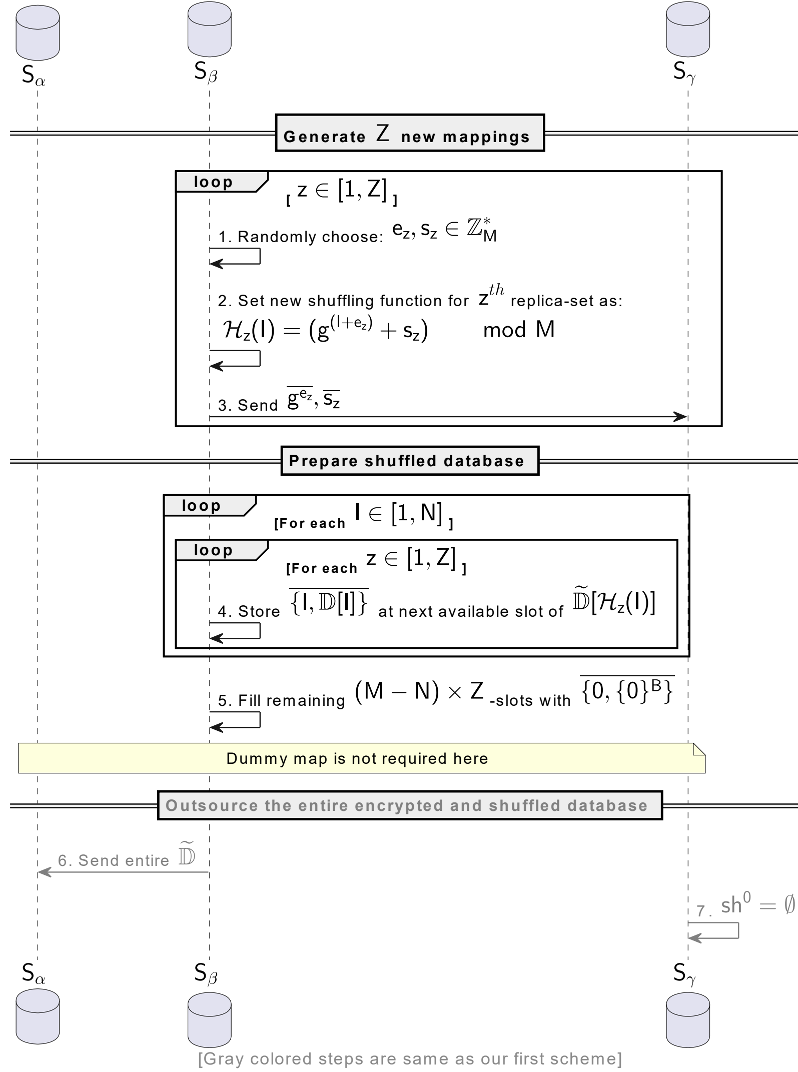


Figure 3.5: Shuffling Process of Differentially Private Scheme

Figure 3.5 shows the shuffling process of the *DP-Variant Scheme*. Instead of a single one, there are now Z-independent functions $\{\mathcal{H}_1, \mathcal{H}_2, .., \mathcal{H}_Z\}$, where $\mathcal{H}_z$ is used for determining the target buckets of zth replica set (i.e., z^{th} replica of all the blocks, where $z \in [1, Z]$). Although Z-different keys differentiate these Z-functions, the overall structure of the functions

remains the same as the previous (i.e., $\mathcal{H}_z(I) = g^{e_z+I} + s_z \mod M$).

At first, S_β randomly chooses Z-different keys: $\langle\{e_1, s_1\}, \{e_2, s_2\}, .., \{e_Z, s_Z\}\rangle$ and then sends corresponding ciphertext pairs $\forall_{z\in[1,Z]}\{\overline{g^{e_z}}, \overline{s_z}\}$ to S_γ (Figure 3.5, Step 3). Next, S_β prepares the shuffled database locally. For z^{th} replica of a block, S_β uses $\mathcal{H}_z$ to determine the location within $\widetilde{\mathbb{D}}$ (i.e., which bucket) and then places the corresponding encrypted pair in an available slot of that bucket (Step 4).

Each shuffling function is individually collision-resistant. Hence, no more than Z block replicas can be mapped to a single bucket. However, after placing all the block replicas, $(M - N) \times Z$-slots will still remain empty. This might give S_α some adversarial advantages. Hence, to hide the distribution of the empty slots among the buckets, S_β fills the remaining $(M - N) \times Z$ empty slots with the ciphertexts of garbage values. Due to IND-CPA security of FHE, the adversary cannot distinguish them.

After completing the preparation of the shuffled database locally, S_β transfers the entire shuffled database to S_α (Step 6), as with our previous scheme. It is important to note that, unlike the first scheme, touching a location more than once is now permitted. Consequently, there is no necessity for dummy pairs and maintaining the dummy mapping.

The goal is to reduce the computation complexity to $O(K)$, where $K < \sqrt{N}$, which also includes the amortized complexity of the shuffling process. In this case, although the number of buckets, M, reduces from $> (N + \sqrt{N})$ to $\geq N$, the size of each bucket increases by Z-times. As a result, the complexity of the shuffling operation in this scheme is: $O(M \times Z)$. Therefore, to achieve an amortized shuffling overhead of $O(K)$, the size of the epoch, E, is increased to: $\frac{M \times Z}{K}$.

3.3.5 Differences in the Fetching Process

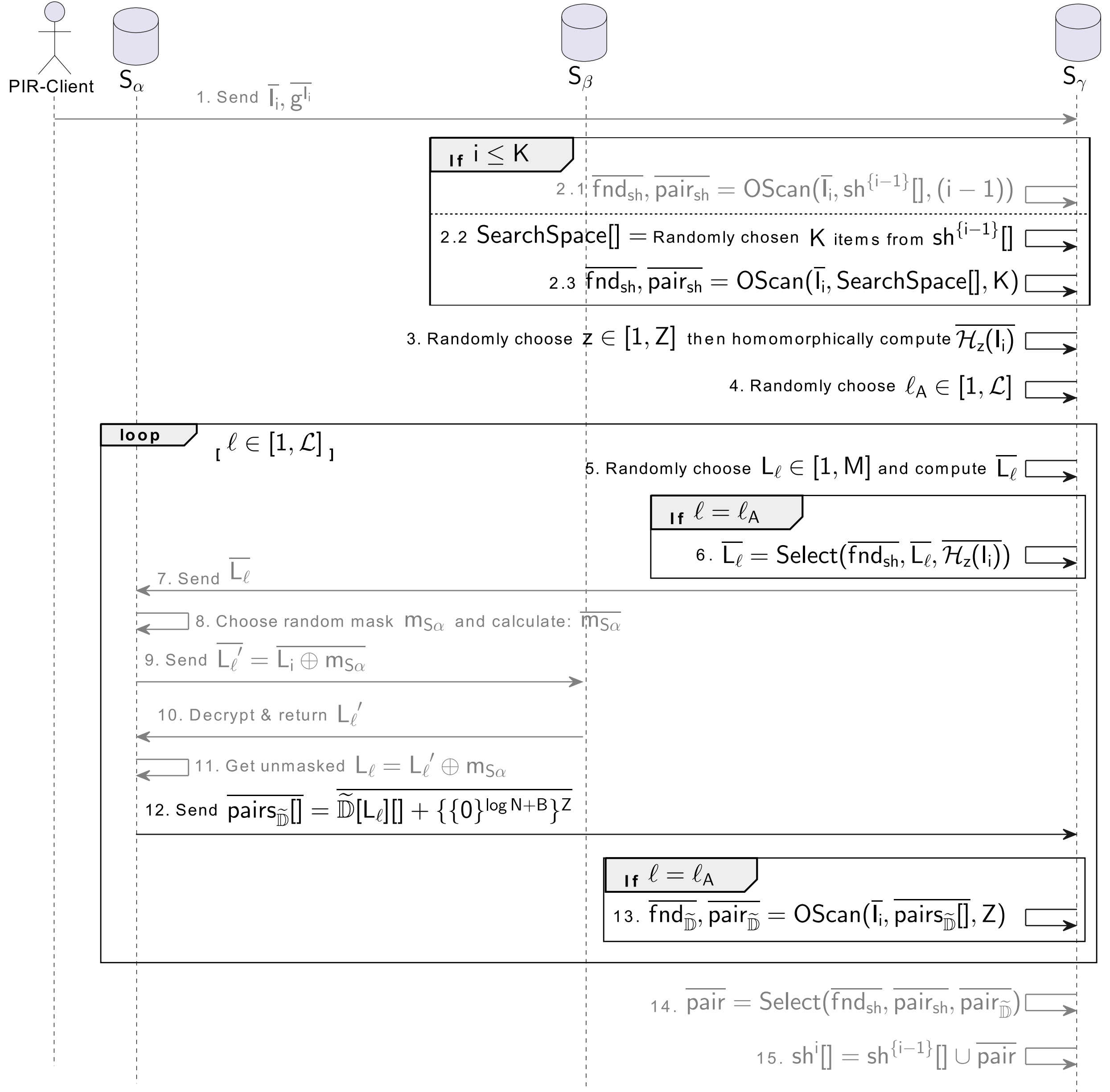


Figure 3.6: Fetching Process of Differentially Private Scheme

This phase differs significantly from *Basic Scheme*, as illustrated in Figure 3.6. In our *DP-Variant Scheme*, S_γ scans up to K items in the shelter, ensuring that the computational overhead remains at $O(\mathsf{K})$. Another notable modification in this scheme is that instead of requesting a single ciphertext from one specific location in the shuffled database, S_γ requests contents of $\mathcal{L}$ buckets from the shuffled database, where $\mathcal{L} = \frac{\mathsf{K}}{\mathsf{Z}}$. Consequently, when processing each request, S_α may observe $\mathcal{L}$ different locations.

As before, the client initiates the phase by sending $\overline{\mathsf{I_i}}$ and $\overline{\mathsf{g}^{\mathsf{I_i}} \mod \mathsf{M}}$ to S_γ. When the shelter's size is below K, S_γ searches all the items of the shelter. However, if the shelter's size exceeds K, S_γ randomly selects K items from the shelter and executes $\mathsf{OScan}()$ on just those selected items (Step 2.2 and 2.3 of Figure 3.6).

In this scheme, Z-replicas of each block exist in the shuffled database. While fetching the block ciphertext, S_γ randomly chooses the shuffled location of one of the replicas of the requested block, $\mathsf{I_i}$. Specifically, S_γ does this by randomly choosing one of the Z-different encrypted key-pairs $\langle \{\mathsf{e_1}, \mathsf{s_1}\}, \{\mathsf{e_2}, \mathsf{s_2}\}, .., \{\mathsf{e_Z}, \mathsf{s_Z}\} \rangle$ (received earlier from S_β) while homomorphically determining the shuffled location of $\mathsf{I_i}$ (Step 3). As a result, even if the same block is requested more than once (and it is not found in the shelter), the same bucket may not get re-touched.

Although there are no special dummy pairs, this scheme also generates dummy locations during each access. Moreover, instead of producing a single dummy location for each request, S_γ now generates $\mathcal{L}$ dummy locations (Step 5). Since there are no designated dummy buckets in this scheme, S_γ selects one of the available M-locations as the dummy location at random for each choice.

If the requested block is located within the shelter, S_γ requests buckets from all of the $\mathcal{L}$-dummy locations. Conversely, if the shelter search is not successful, S_γ requests buckets from $(\mathcal{L} - 1)$ dummy locations, in addition to the bucket containing the actual requested block, which is situated at location $\mathcal{H}_\mathsf{z}(\mathsf{I_i})$. To achieve this, S_γ randomly selects one of the previously identified dummy locations to substitute for $\mathcal{H}_\mathsf{z}(\mathsf{I_i})$ (Steps 4-6). Consequently, after observing the $\mathcal{L}$ locations, S_α will be unable to determine which location *might* contain the requested block. This substitution is executed using the $\mathsf{Select}()$ function, and since FHE is IND-CPA secure, S_γ remains unaware of whether the substitution has occurred.

S_α then retrieves the plaintext versions of all $\mathcal{L}$-received locations one by one, employing a similar method of oblivious decryption (Steps 8-11). It subsequently returns the ciphertext stored in the buckets located at those positions to S_γ. To obscure whether the same bucket is returned multiple times, S_α homomorphically adds ciphertexts of zeros to the contents of the buckets before sending them to S_γ (Step 12). Consequently, while the actual plaintexts remain unchanged, the appearances of the ciphertexts are modified.

Therefore, S_γ cannot determine whether it is observing a new bucket or not.

In turn, S_γ disregards all buckets except the one that might contain the requested block. One of the Z-ciphertexts of this particular bucket may correspond to the requested block. Consequently, S_γ performs an oblivious scan on that bucket to determine $\overline{\mathsf{pair}_{\widetilde{\mathbb{D}}}}$ (Step 13). Finally, akin to our *Basic Scheme*, S_γ determines the ciphertext of the requested pair using $\mathsf{Select}()$ and appends it to the shelter (Step 14-15). Consequently, similar to the previous scheme, if the same block is accessed multiple times, the corresponding pair appears multiple times within the shelter.

3.3.6 Effect on Overheads

We present a summary of the overheads for our *DP-Variant Scheme* in Table 3.3. While the storage overhead and online communication for the servers have increased, compared to our initial scheme, this *DP-Variant Scheme* reduces the total server computation and communication overhead from $O(\sqrt{\mathsf{N}})$ to $O(\mathsf{K})$, while maintaining the same overhead for clients.

Table 3.3: Summary of the overheads of our *DP-Variant Scheme*†

Overhead type	**Client**	S_α	S_β	S_γ	**Total-server**
Computation during request processing	$\boldsymbol{O(1)}$	$O(\mathsf{K})$	$O(\frac{\mathsf{K}}{\mathsf{Z}})$	$O(\mathsf{K})$	$\boldsymbol{O(\mathsf{K})}$
Amortized computation	-	$O(1)$	$O(\mathsf{K})$	$O(1)$	$\boldsymbol{O(\mathsf{K})}$
Overall computation	$\boldsymbol{O(1)}$	$O(\mathsf{K})$	$O(\mathsf{K})$	$O(\mathsf{K})$	$\boldsymbol{O(\mathsf{K})}$
Communication during request processing	$\boldsymbol{O(1)}$	$O(\mathsf{K})$	$O(\frac{\mathsf{K}}{\mathsf{Z}})$	$O(\mathsf{K})$	$O(\mathsf{K})$
Amortized communication	-	$O(\mathsf{K})$	$O(\mathsf{K})$	$O(1)$	$\boldsymbol{O(\mathsf{K})}$
Overall communication	$\boldsymbol{O(1)}$	$O(\mathsf{K})$	$O(\mathsf{K})$	$O(\mathsf{K})$	$\boldsymbol{O(\mathsf{K})}$
Total Storage	$\boldsymbol{O(1)}$	$O(\mathsf{MZ})$	$O(\mathsf{MZ})$	$O(\frac{\mathsf{MZ}}{\mathsf{K}})$	$\boldsymbol{O(\mathsf{MZ})}$

† In this table, the red cells indicate increased overheads compared to *Basic Scheme*, while the green cells represent reduced overheads.

S_α, S_β, and S_γ participate in $\mathcal{L}$ rounds of communication for each request (as depicted in Figure 3.6), and transferring each bucket now involves the transfer of Z ciphertexts. As a result, the online communication burden is now $O(\mathsf{K})$. However, the computation has been reduced to $O(\mathsf{K})$ by limiting the shelter search operation. Moreover, this scheme reduces amortized communication and computation to $O(\mathsf{K})$ by increasing the epoch size, thereby decreasing the frequency of the costly per-epoch operations.

By adjusting the value of K, the system administrator can effectively manage the tradeoff between privacy levels and overhead. This is discussed in the next subsection.

3.3.7 Privacy-Overhead Tradeoff Analysis

In this scheme, S_α may observe the same bucket being accessed multiple times within a single epoch. This situation could potentially lead to some privacy leakage, which we will analyze in this section. We show that our design meets the definition of (ϵ, δ)-PIR system, defined by Toledo et al. [60]. First, we figure out the adversary's best possible attack strategy. Next, we analyze the ϵ bound and then the δ bound based on this strategy.

However, it is important to clarify that the views of S_β and S_γ remain nearly identical to those in *Basic Scheme*, and neither gains any additional insights. Specifically, S_γ has access only to IND-CPA-secure ciphertexts, while the contents are information theoretically concealed from S_β due to the application of random masks. Therefore, whenever we refer to the term "adversary" ($\mathcal{A}$) in the subsequent discussion of this differentially private scheme, we are specifically referring to the honest-but-curious S_α, who seeks to guess the PIR clients' requests.

Adversary's strategy for choosing $\mathsf{I}', \mathsf{I}'' \& \langle \mathsf{I}^* \rangle$:

The adversary is not aware of any of the shuffling functions $(\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_\mathsf{Z})$. When processing the ith request, $\mathcal{A}$ can only observe a set of bucket locations, $\mathbb{L}_\mathsf{i}$. Here, $\mathbb{L}_\mathsf{i}$ represents the set of all $\mathcal{L}$ locations that are accessed by S_α during the handling of the ith request (refer to Step 11 of Figure 3.6). Within $\mathbb{L}_\mathsf{i}$, only one bucket *might* contain the requested block, while the others are randomly selected by S_γ. Consequently, as a strategy, $\mathcal{A}$ can only cause the targeted PIR client to reaccess the same block during the jth request. Then $\mathcal{A}$ tracks whether there is any overlapping buckets between $\mathbb{L}_\mathsf{i}$ and $\mathbb{L}_\mathsf{j}$.

Since each block has several replicas, distributed among different buckets, merely requesting the same block does not guarantee that the same bucket will be re-touched.

Additionally, because of the shelter's presence, when requesting the same block a second time, it might actually be retrieved from the shelter instead. In that case, all the bucket locations specified in $\mathbb{L}_{\mathsf{j}}$ will consist solely of dummy locations.

However, our scheme scans only K-items from the shelter while processing each request. This design minimizes the probability of locating a re-accessed block within the shelter, when the shelter size is maximized. The shelter reaches its maximum size of $(\mathsf{E}-1)$ during the processing of the final request of the epoch, I_{E}. Therefore, $\mathcal{A}$ issues $\langle \mathsf{I}^* \rangle$ as the first $(\mathsf{E}-1)$ requests of the epoch (i.e., $\mathsf{I}_1\mathsf{I}_2\ldots\mathsf{I}_{\mathsf{E}-1}$) and submits the targeted PIR-client's request as I_{E}. Additionally, $\mathcal{A}$ assigns $\mathsf{I}' = \mathsf{I}_1$ and $\mathsf{I}'' \neq \mathsf{I}_1$.

The probability of locating a block within the shelter also depends on the number of replicas of that block present within the shelter. Hence, $\mathcal{A}$ minimizes the probability of finding I' within the shelter by ensuring that I_1 does not reappear in the sequence $\langle \mathsf{I}^* \rangle$. This approach guarantees that only one replica of I_1 exists within the shelter, thereby minimizing the chances of its discovery during shelter searching while processing I_{E}.

If $\mathsf{I}_{\mathsf{E}} = \mathsf{I}''$ and I'' is already present within the shelter, then all items in $\mathbb{L}_{\mathsf{E}}$ correspond to randomly designated dummy locations. Conversely, if I'' is not found within the shelter, I'' must be retrieved from $\widetilde{\mathbb{D}}$. Among the $(\mathsf{Z} \times \mathcal{L})$ slots in $\mathbb{L}_1$, at least one slot cannot contain I'' (since at least one slot must be occupied by I_1). As a result, the absence of I'' within the shelter reduces the chances of accessing a location in $\mathbb{L}_1$. Thus, the adversary selects I'', which is not included in $\langle \mathsf{I}^* \rangle$.

This strategy maximizes the probability of finding a common item between $\mathbb{L}_1$ and $\mathbb{L}_{\mathsf{E}}$ if the client selects I'. Conversely, if the client chooses I'', the probability of identifying a common item between $\mathbb{L}_1$ and $\mathbb{L}_{\mathsf{E}}$ is minimized. $\mathcal{A}$ leverages this difference in probabilities to infer whether the client has selected I' or I''.

The ϵ bound:

Suppose that $\mathcal{A}$ observes $\mathbb{L}_1 \cap \mathbb{L}_{\mathsf{E}} \neq \emptyset$. As a consequence, $\mathcal{A}$ is more inclined to believe that $\mathsf{I}_{\mathsf{E}} = \mathsf{I}'$ rather than $\mathsf{I}_{\mathsf{E}} = \mathsf{I}''$. The condition $\mathbb{L}_1 \cap \mathbb{L}_{\mathsf{E}} \neq \emptyset$ implies that $|\mathbb{L}_1 \cap \mathbb{L}_{\mathsf{E}}| \geq 1$. However, the probability of having $|\mathbb{L}_1 \cap \mathbb{L}_{\mathsf{E}}| > 1$ is quite low. Thus, when computing ϵ, we assume that $|\mathbb{L}_1 \cap \mathbb{L}_{\mathsf{E}}| = 1$ (i.e., $\mathcal{A}$ observes only one common item between $\mathbb{L}_1$ and $\mathbb{L}_{\mathsf{E}}$). Conversely, if $\mathcal{A}$ observes more than one common item, the privacy leakage could exceed our calculated ϵ. Consequently, we represent the probability of such events as δ.

Therefore, for our scheme:

$$\mathsf{e}^{\epsilon} = \frac{\Pr[(|\mathbb{L}_1 \cap \mathbb{L}_{\mathsf{E}}| = 1)|(\mathsf{I}_{\mathsf{E}} = \mathsf{I}')]}{\Pr[(|\mathbb{L}_1 \cap \mathbb{L}_{\mathsf{E}}| = 1)|(\mathsf{I}_{\mathsf{E}} = \mathsf{I}'')]} \tag{3.2}$$

It is important to note that, just like a standard DP mechanism, here also $\mathcal{A}$ has no prior knowledge regarding $\Pr(\mathsf{I}_{\mathsf{E}} = \mathsf{I}')$ or $\Pr(\mathsf{I}_{\mathsf{E}} = \mathsf{I}'')$.

Determining $\Pr[(|\mathbb{L}_1 \cap \mathbb{L}_{\mathsf{E}}| = 1)|(\mathsf{I}_{\mathsf{E}} = \mathsf{I}')]$**:** Suppose, $\mathsf{L}_1 \in \mathbb{L}_1$ and the bucket $\widetilde{\mathbb{D}}[\mathsf{L}_1][]$ actually contains I_1, the first requested block of the epoch. Therefore, $\Pr[(|\mathbb{L}_1 \cap \mathbb{L}_{\mathsf{E}}| = 1)|(\mathsf{I}_{\mathsf{E}} = \mathsf{I}')]$ can be written as:

$$\begin{aligned}&\Big\{\Pr[\mathsf{I}' \notin \mathsf{SearchSpace}[]] \cap \Pr[\mathcal{H}_{\mathsf{z}}(\mathsf{I}_1) = \mathcal{H}_{\mathsf{z}}(\mathsf{I}')] \cap \Pr[\{\mathbb{L}_{\mathsf{E}} \setminus \{\mathcal{H}_{\mathsf{z}}(\mathsf{I}')\}\} \cap \mathbb{L}_1\} = \emptyset]\Big\} + \\ &\Big\{\Pr[\mathsf{I}' \in \mathsf{SearchSpace}[]] \cap \Pr[(|\mathbb{L}_{\mathsf{E}} \cap \mathbb{L}_1| = 1)|(\mathsf{I}' \in \mathsf{SearchSpace}[])]\Big\}\end{aligned} \tag{3.3}$$

Only K randomly chosen items from the shelter (which is denoted as $\mathsf{SearchSpace}[]$ in figure: 3.6) are searched during each request. While processing I_{E}, there are $(\mathsf{E}-1)$ items in the shelter, and only one of them is $\mathsf{I}' = \mathsf{I}_1$. Combining these two facts, we get $\Pr[\mathsf{I}' \in \mathsf{SearchSpace}[]]$ as:

$$\frac{\mathsf{K}}{(\mathsf{E}-1)} \tag{3.4}$$

Consequently, $\Pr[\mathsf{I}' \notin \mathsf{SearchSpace}[]]$ is:

$$\left(1 - \frac{\mathsf{K}}{\mathsf{E}-1}\right) \tag{3.5}$$

There exists Z-available replicas of $\mathsf{I}_1 = \mathsf{I}'$ and only one of them exists at the location, which was selected while processing I_1. Hence, $\Pr[\mathcal{H}_{\mathsf{z}}(\mathsf{I}_1) = \mathcal{H}_{\mathsf{z}}(\mathsf{I}')]$ is:

$$\frac{1}{\mathsf{Z}} \tag{3.6}$$

Please note that $((|\mathbb{L}_{\mathsf{E}} \cap \mathbb{L}_1| = 1)|(\mathsf{I}' \in \mathsf{SearchSpace}[]))$ means that S_{γ} selects exactly one location from $\mathbb{L}_1$ while choosing $\mathcal{L}$ dummy locations and probability of this event follows a binomial distribution. Given that the size of $\mathbb{L}_1$ is $\mathcal{L}$, the probability that a randomly

chosen dummy location belongs to $\mathbb{L}_1$ during each trial is $\frac{\mathcal{L}}{\mathsf{M}}$. Consequently, we can express $\Pr[(|\mathbb{L}_\mathsf{E} \cap \mathbb{L}_1| = 1)|(\mathsf{I}' \in \mathsf{SearchSpace}[])]$ as:

$$\binom{\mathcal{L}}{1} \times \left(\frac{\mathcal{L}}{\mathsf{M}}\right)^1 \times \left(1 - \frac{\mathcal{L}}{\mathsf{M}}\right)^{(\mathcal{L}-1)} = \left(\frac{\mathcal{L}^2}{\mathsf{M}}\right) \times \left(1 - \frac{\mathcal{L}}{\mathsf{M}}\right)^{(\mathcal{L}-1)} \tag{3.7}$$

Similarly, $\Pr[\{\mathbb{L}_\mathsf{E} \setminus \{\mathcal{H}_\mathsf{z}(\mathsf{I}')\}\} \cap \mathbb{L}_1\} = \emptyset]$ is:

$$\binom{\mathcal{L}}{0} \times \left(\frac{\mathcal{L}}{\mathsf{M}}\right)^0 \times \left(1 - \frac{\mathcal{L}}{\mathsf{M}}\right)^{(\mathcal{L}-1)} = \left(1 - \frac{\mathcal{L}}{\mathsf{M}}\right)^{(\mathcal{L}-1)} \tag{3.8}$$

By putting all the individual probability values in Equation 3.3, we get:

$$\left\{\left(1 - \frac{\mathsf{K}}{\mathsf{E}-1}\right) \times \frac{1}{\mathsf{Z}} \times \left(1 - \frac{\mathcal{L}}{\mathsf{M}}\right)^{(\mathcal{L}-1)}\right\} + \left\{\frac{\mathsf{K}}{(\mathsf{E}-1)} \times \left(\frac{\mathcal{L}^2}{\mathsf{M}}\right) \times \left(1 - \frac{\mathcal{L}}{\mathsf{M}}\right)^{(\mathcal{L}-1)}\right\} \tag{3.9}$$

Determining $\Pr[(|\mathbb{L}_1 \cap \mathbb{L}_\mathsf{E}| = 1)|\mathsf{I}'']$**:** This can be represented as:

$$\Big\{\Pr[\mathcal{H}_\mathsf{z}(\mathsf{I}'') \in \mathbb{L}_1] \cap \Pr[\{\mathbb{L}_\mathsf{E} \setminus \{\mathcal{H}_\mathsf{z}(\mathsf{I}'')\}\} \cap \mathbb{L}_1\} = \emptyset]\Big\} +$$
$$\Big\{\Pr[\mathcal{H}_\mathsf{z}(\mathsf{I}'') \notin \mathbb{L}_1] \cap \Pr[(|\mathbb{L}_\mathsf{E} \cap \mathbb{L}_1| = 1)|(\mathsf{I}'' \in \mathsf{SearchSapce}[])]\Big\}$$

Z replicas of I'' are randomly distributed among $(\mathsf{M} \times \mathsf{Z})$-available slots within $\widetilde{\mathbb{D}}$ with one exception: one slot among the buckets, specified in $\mathbb{L}_1$ cannot contain I''.

Therefore, $\Pr[\mathcal{H}_\mathsf{z}(\mathsf{I}'') \in \mathbb{L}_1]$ is:

$$\frac{1}{\mathsf{Z}} \times \frac{(\mathcal{L} \times \mathsf{Z} - 1)}{\mathsf{M}} \tag{3.10}$$

Consequently, $\Pr[\mathcal{H}_\mathsf{z}(\mathsf{I}'') \notin \mathbb{L}_1]$ is:

$$1 - \left\{\frac{1}{\mathsf{Z}} \times \frac{(\mathcal{L} \times \mathsf{Z} - 1)}{\mathsf{M}}\right\} \tag{3.11}$$

By combining these values with Equation 3.7 and Equation 3.8 we get:

$$\begin{aligned}&\left\{\frac{(\mathcal{L} \times \mathsf{Z} - 1)}{\mathsf{M} \times \mathsf{Z}} \times \left(1 - \frac{\mathcal{L}}{\mathsf{M}}\right)^{(\mathcal{L}-1)}\right\} + \\ &\left[\left\{1 - \frac{(\mathcal{L} \times \mathsf{Z} - 1)}{\mathsf{M} \times \mathsf{Z}}\right\} \times \left(\frac{\mathcal{L}^2}{\mathsf{M}}\right) \times \left(1 - \frac{\mathcal{L}}{\mathsf{M}}\right)^{(\mathcal{L}-1)}\right]\end{aligned} \tag{3.12}$$

By considering Equation 3.9 and Equation 3.12, we found:

$$\mathsf{e}^{\epsilon} = \frac{\left\{\left(1-\frac{\mathsf{K}}{(\mathsf{E}-1)}\right)\times\frac{1}{\mathsf{Z}}\times\left(1-\frac{\mathcal{L}}{\mathsf{M}}\right)^{(\mathcal{L}-1)}\right\}+\left\{\frac{\mathsf{K}}{(\mathsf{E}-1)}\times\left(\frac{\mathcal{L}^2}{\mathsf{M}}\right)\times\left(1-\frac{\mathcal{L}}{\mathsf{M}}\right)^{(\mathcal{L}-1)}\right\}}{\left\{\frac{(\mathcal{L}\times\mathsf{Z}-1)}{\mathsf{M}\times\mathsf{Z}}\times\left(1-\frac{\mathcal{L}}{\mathsf{M}}\right)^{(\mathcal{L}-1)}\right\}+\left[\left\{1-\frac{(\mathcal{L}\times\mathsf{Z}-1)}{\mathsf{M}\times\mathsf{Z}}\right\}\times\left(\frac{\mathcal{L}^2}{\mathsf{M}}\right)\times\left(1-\frac{\mathcal{L}}{\mathsf{M}}\right)^{(\mathcal{L}-1)}\right]}$$

Then we plug the value of $\mathsf{E}=\frac{\mathsf{MZ}}{\mathsf{K}}$ and $\mathcal{L}=\frac{\mathsf{K}}{\mathsf{Z}}$ and after simplification [66], we find ϵ as:

$$\epsilon = \ln\left[\frac{\mathsf{M}^2\mathsf{Z}^3\left\{\mathsf{MZ}^2(\mathsf{MZ}-\mathsf{K}-\mathsf{K}^2)+\mathsf{K}^4\right\}}{(\mathsf{MZ}-\mathsf{K})\mathsf{MZ}^2\left\{(\mathsf{K}-1)\mathsf{MZ}^2+\mathsf{K}^2\mathsf{MZ}-\mathsf{K}^2(\mathsf{K}-1)\right\}}\right] \tag{3.13}$$

The δ bound:

The privacy leakage will exceed the above-calculated ϵ value when $\mathcal{A}$ observes more than one common item between $\mathbb{L}_1$ and $\mathbb{L}_\mathsf{E}$, as a result of $\mathsf{I}_\mathsf{E}=\mathsf{I}'$. Therefore, $\Pr[(|\mathbb{L}_1\cap\mathbb{L}_\mathsf{E}|>1)|\mathsf{I}']$ represents the δ value of our scheme and can be expressed as:

$$\begin{aligned}&\Big[\Pr[\mathsf{I}'\notin\mathsf{SearchSpace}[]]\cap\Pr[(\mathbb{L}_\mathsf{E}\cap\mathbb{L}_1\neq\emptyset)|(\{\mathcal{H}_\mathsf{i}(\mathsf{I}')|\mathsf{i}=1,2,\ldots,\mathsf{Z}\}\cap\mathbb{L}_1\neq\emptyset)]\cap\\&\left\{\sum_{\mathsf{x}=1}^{\mathcal{L}-1}\Pr[|\{\mathbb{L}_\mathsf{E}\setminus\{\mathcal{H}_\mathsf{z}(\mathsf{I}')\}\}\cap\mathbb{L}_1|=\mathsf{x}]\right\}\Big]+\\&\left[\Pr[\mathsf{I}\in\mathsf{SearchSpace}[]]\cap\left\{\sum_{\mathsf{x}=2}^{\mathcal{L}}\Pr[|\{\mathsf{L}_\ell|\ell=1,2,\ldots,\mathcal{L}\}\cap\mathbb{L}_1|=\mathsf{x}]\right\}\right]\end{aligned} \tag{3.14}$$

We already determined that $\Pr[\mathsf{I}'\in\mathsf{SearchSpace}[]]$ and $\Pr[\mathsf{I}'\notin\mathsf{SearchSpace}[]]$ as $\frac{\mathsf{K}}{(\mathsf{E}-1)}$ and $\left(1-\frac{\mathsf{K}}{\mathsf{E}-1}\right)$, respectively. The unconditional probability, $\Pr[\mathbb{L}_\mathsf{E}\cap\mathbb{L}_1\neq\emptyset]$ is: $\frac{\mathcal{L}}{\mathsf{M}}$.

$\Pr[\{\mathcal{H}_\mathsf{i}(\mathsf{I}')|\mathsf{i}=1,2,\ldots,\mathsf{Z}\}\cap\mathbb{L}_1\neq\emptyset]$ can be written as:

$$\begin{aligned}&1-\Pr[\{\mathcal{H}_\mathsf{z}(\mathsf{I}')|\mathsf{i}=1,2,\ldots,\mathsf{Z}\}\cap\mathbb{L}_1=\emptyset]\\&=1-\binom{\mathsf{Z}}{0}\times\left(\frac{\mathcal{L}}{\mathsf{M}}\right)^0\times\left(1-\frac{\mathcal{L}}{\mathsf{M}}\right)^\mathsf{Z}=1-\left(1-\frac{\mathcal{L}}{\mathsf{M}}\right)^\mathsf{Z}\end{aligned}$$

As a result, $\Pr[(\mathbb{L}_\mathsf{E}\cap\mathbb{L}_1\neq\emptyset)|(\{\mathcal{H}_\mathsf{i}(\mathsf{I}')|\mathsf{i}=1,2,\ldots,\mathsf{Z}\}\cap\mathbb{L}_1\neq\emptyset)]$ is:

$$\frac{\frac{\mathcal{L}}{\mathsf{M}}}{1-\left(1-\frac{\mathcal{L}}{\mathsf{M}}\right)^\mathsf{Z}} \tag{3.15}$$

$\left\{\sum_{x=1}^{\mathcal{L}-1} \Pr[|\{\mathbb{L}_E \setminus \{\mathcal{H}_z(l')\}\} \cap \mathbb{L}_1| = x]\right\}$ can be written as: $1 - \Pr[|\{\mathbb{L}_E \setminus \{\mathcal{H}_z(l')\}\} \cap \mathbb{L}_1| = \emptyset]$. By utilizing Equation 3.8, we can write its value as:

$$= 1 - \left(1 - \frac{\mathcal{L}}{M}\right)^{(\mathcal{L}-1)} \tag{3.16}$$

On the other hand, $\left\{\sum_{x=2}^{\mathcal{L}} \Pr[|\{L_\ell | \ell = 1, 2, \ldots, \mathcal{L}\} \cap \mathbb{L}_1| = x]\right\}$ can be written as: $\{1 - \Pr[|\{L_\ell | \ell = 1, 2, \ldots, \mathcal{L}\} \cap \mathbb{L}_1| = 0] \ -\Pr[|\{L_\ell | \ell = 1, 2, \ldots, \mathcal{L}\} \cap \mathbb{L}_1| = 1]\}$. Which is:

$$\begin{aligned} & 1 - \left\{\binom{\mathcal{L}}{0} \times \left(\frac{\mathcal{L}}{M}\right)^{0} \times \left(1 - \frac{\mathcal{L}}{M}\right)^{\mathcal{L}}\right\} - \left\{\binom{\mathcal{L}}{1} \times \left(\frac{\mathcal{L}}{M}\right)^{1} \times \left(1 - \frac{\mathcal{L}}{M}\right)^{(\mathcal{L}-1)}\right\} \\ & = 1 - \left(1 - \frac{\mathcal{L}}{M}\right)^{\mathcal{L}} - \left(\frac{\mathcal{L}^2}{M}\right) \times \left(1 - \frac{\mathcal{L}}{M}\right)^{(\mathcal{L}-1)} \end{aligned} \tag{3.17}$$

By putting all the probabilities into Equation 3.14, we get δ as:

$$\begin{aligned} & \left[\left\{1 - \frac{K}{E-1}\right\} \times \frac{\frac{\mathcal{L}}{M}}{1 - (1 - \frac{\mathcal{L}}{M})^{Z}} \times \left\{1 - \left(1 - \frac{\mathcal{L}}{M}\right)^{(\mathcal{L}-1)}\right\}\right] + \\ & \left[\frac{K}{(E-1)} \times \left\{1 - \left(1 - \frac{\mathcal{L}}{M}\right)^{\mathcal{L}} - \left(\frac{\mathcal{L}^2}{M}\right) \times \left(1 - \frac{\mathcal{L}}{M}\right)^{(\mathcal{L}-1)}\right\}\right] \end{aligned}$$

After plugging the values $E = \frac{MZ}{K}$ and $\mathcal{L} = \frac{K}{Z}$, we get:

$$\begin{aligned} & \left[\left\{1 - \frac{K^2}{(MZ-K)}\right\} \times \frac{\frac{K}{MZ}}{1 - (1 - \frac{K}{MZ})^{Z}} \times \left\{1 - \left(1 - \frac{K}{MZ}\right)^{(\frac{K}{Z}-1)}\right\}\right] + \\ & \left[\frac{K^2}{(MZ-K)} \times \left\{1 - \left(1 - \frac{K}{MZ}\right)^{\frac{K}{Z}} - \left(\frac{K^2}{MZ^2}\right) \times \left(1 - \frac{K}{MZ}\right)^{(\frac{K}{Z}-1)}\right\}\right] \end{aligned} \tag{3.18}$$

Discussion on the advantages of our scheme:

We substituted concrete values into the Equation 3.13 and obtained the actual ϵ values (Figure 3.7). Our analysis focused on a database size of 100 GB, which corresponds to

1.68 billion blocks of size 512 bits. As this scheme aims to reduce overhead to a level lower than $\sqrt{\mathsf{N}}$, we investigated the impact on privacy leakage by varying the expected overhead parameter, K, up to $\sqrt{\mathsf{N}}$. In this estimation, we assumed that the number of replicas for each block, Z, is set at 3.

We also compare our scheme with the DP-IR schemes proposed by Toledo et al. [60], specifically their Direct Request and Sparse PIR schemes. We have chosen not to consider their other schemes because those rely on the existence of anonymous channels, which we do not use in our solution. Our scheme operates with three non-colluding servers. To ensure a fair comparison, we utilize the corresponding parameters when assessing the privacy leakage of Toledo et al.'s approach.

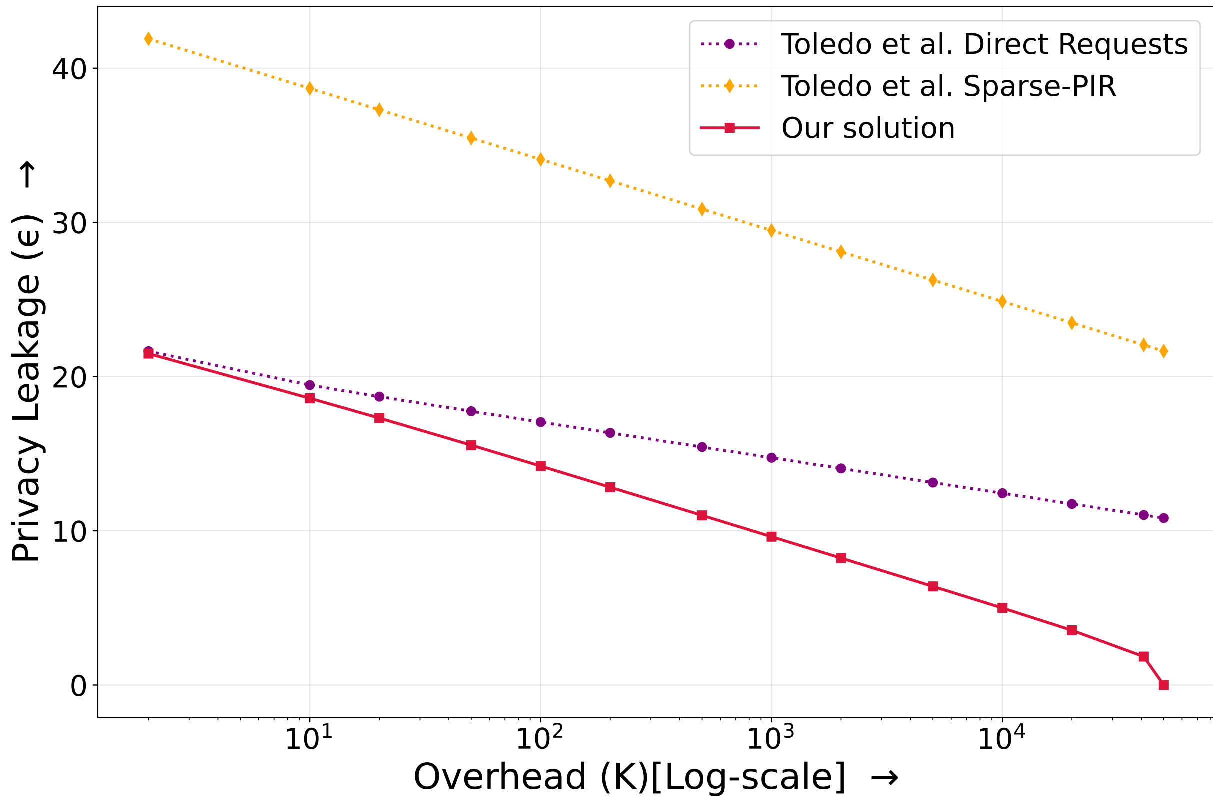


Figure 3.7: Privacy-Overhead Tradeoff Analysis

Compared with other solutions, our scheme allows servers to provide the desired privacy guarantee (the ϵ value) with considerably reduced overhead (the K value). Moreover,

Table 3.4: δ-values of our scheme ($\mathsf{N} = 1.68$ billion)

K-values $\rightarrow$	$\mathbf{10^1}$	$\mathbf{10^2}$	$\mathbf{10^3}$	$\mathbf{10^4}$
$\mathbf{Z = 2}$	6×10^{-9}	7.3×10^{-7}	7.4×10^{-5}	7.2×10^{-3}
$\mathbf{Z = 3}$	1.5×10^{-9}	2.1×10^{-7}	2.2×10^{-5}	2.2×10^{-3}
$\mathbf{Z = 5}$	2.4×10^{-10}	4.5×10^{-8}	4.7×10^{-6}	4.7×10^{-4}
$\mathbf{Z = 10}$	1.4×10^{-26}	5.4×10^{-9}	5.9×10^{-7}	5.9×10^{-5}

unlike other schemes, our method eliminates the necessity for clients to generate additional dummy queries, making it a more attractive option for clients. Additionally, the δ values achieved in our scheme are reasonably small (refer to Table: 3.4). These combined advantages render our solution an great choice for practical applications.

3.4 *DPF-Variant Scheme*: Enhancing the Practicality by Utilizing DPF

Our differentially private scheme significantly decreases the number of homomorphic computations required for processing each request, thereby making the PIR solution much more practical. However, if we aim for stronger privacy guarantees, our first scheme shows asymptotic improvements over existing methods but still requires $O(\sqrt{\mathsf{N}})$ homomorphic computations. Moreover, although in theory the FHE can perform any computation on the ciphertext, in practice, the bootstrapping process after each computation is not yet efficient. This presents a practical challenge.

This raises an important question: is it possible to achieve better outcomes? Specifically, can we develop a PIR scheme that maintains server overhead at $O(\sqrt{\mathsf{N}})$ while reducing the number of costly operations, such as homomorphic computations, to $O(1)$? If possible, we can additionally bypass the expensive bootstrapping operations by using somewhat homomorphic encryption schemes or an FHE scheme in leveled mode.

Fortunately, the answer to this question is "Yes" and the remaining sections show how this can be achieved.

3.4.1 Related Work Regarding utilization of DPF in PIR and DORAM

The *Ostrovsky-Shoup* compiler [55], mentioned earlier in Section 3.2.1, can indeed be utilized with any multi-server ORAM to implement a PIR system. However, this approach necessitates a significant number of multiparty computations. Consequently, while it may theoretically yield a PIR scheme with sub-linear computational overhead, each individual computation remains quite costly. Furthermore, in this case, not only the servers, but the clients are also burdened with this computational expense.

Gilboa et al. introduced a novel cryptographic primitive known as the distributed point function (DPF) [67], which is useful when an oblivious operation is required. DPF enables a trusted *dealer* to split a point function among two or more *evaluators* such that each evaluator can only compute a secret share of the function output, while remaining unaware of the function or its output (see appendix: B for details). This primitive is particularly interesting, practically, based on its efficiency and suitability for the PIR problem [68].

Although in a DPF-based PIR-solution [68, Appendix A], $O(\mathsf{N})$ DPF evaluations (versus $O(\log \mathsf{N})$ MPC evaluations) are required by the servers, the overall execution time re-

mains significantly lower [69]. Therefore, when evaluating the practicality of a PIR scheme, it is essential to consider not only the total number of operations but also the number of operations that are particularly *expensive*. In our application, we want to minimize the number of expensive operations.

If DPF is to be used in the DORAM scenario, it is essential for the servers to first collaboratively execute the trusted dealer's tasks within an MPC context, followed by performing $O(\mathsf{N})$ local DPF evaluations. Building on this foundation, Doerner and Shelet introduced FLORAM [69]. Although their approach involves an additional $O(\log \mathsf{N})$ number of MPC evaluations, they reduced the complexity of the function being evaluated under MPC to be significantly less. Additionally, their FLO**ROM** construction, which has more similarities with a PIR scheme, improved the server communication to $O(\log \mathsf{N})$.

The concept of FLORAM is further enhanced in DOURAM [70] using pre-processing techniques to implement parts of the DPF functionality. This reduces inter-server communication overhead down to $O(1)$ per access. However, for m accesses, $O(m\log \mathsf{N})$ amount of communication is required for the pre-processing phase. As a result, even though the *online* bandwidth between servers decreases to $O(1)$, the amortized server communication overhead remains at $O(\log \mathsf{N})$ per access. Most importantly, if write operations are completely eliminated from the DOURAM framework, which is fine for PIR, the external PIR client might not need to engage in any MPC whatsoever. In fact, the overall performance approaches that of the original DPF-based PIR proposed by Boyle et al. [68], resulting in $O(\mathsf{N})$ *inexpensive* DPF evaluations.

All DPF-based DORAMs require $O(\mathsf{N})$ evaluations, with static database arrangements on the server. However, Braun et al. [71] observed that, by implementing a *reshuffle-after-a-certain-time* strategy (like square-root ORAM [15]), access patterns can still be concealed, while reducing the number of DPF evaluations. They significantly reduced the number of inexpensive server computations down to $O(\sqrt{\mathsf{N}}\log\left(\mathsf{N}+\sqrt{\mathsf{N}}\right))$. Their strategy requires a caching strategy, where a *shelter* is maintained, storing items that have already been fetched. This results in a slight increase in storage requirements. Overall, this is less significant, with current storage costs being as low as they are [39].

Therefore, utilizing DPF demonstrates a significant improvement in the development of DORAM. However, deploying DORAMs in a PIR scenario presents the adversarial servers with enhanced capabilities. These servers can independently simulate fake clients, which may enable them to uncover the access patterns of legitimate clients. On a positive note, within a PIR context, DORAM does not necessitate write functionality. Consequently, adopting a DORAM strategy to address the PIR problem could result in even greater reductions in overhead.

3.4.2 Overview: Performing DPF-based Shelter Searching

This scheme is based on five non-colluding servers: $\mathsf{S}_\alpha, \mathsf{S}_\beta, \mathsf{S}_\gamma, \mathsf{S}_\delta,$ and S_ϵ. It utilizes DPF to minimize the number of homomorphic evaluations to $O(1)$, thereby enhancing the practicality of the solution. The core idea of this scheme is to maintain two identical replicas of the shelter, each of size $O(\sqrt{\mathsf{N}})$, on two different servers, and then use DPF for the oblivious shelter search operation. Since each individual operation performed during the shelter searching process is a cheap DPF evaluation, this scheme offers a PIR solution with $O(\sqrt{\mathsf{N}})$ inexpensive operations.

In this scheme, S_β first locally creates the shuffled database, $\widetilde{\mathbb{D}}$, containing N-real and $\sqrt{\mathsf{N}}$-dummy blocks. Along with the actual block content, each element of $\widetilde{\mathbb{D}}$ holds a unique plaintext tag. These tags are later used to locate the client-requested blocks. To hide the block contents from the storing servers, S_β derives two different databases, $\widetilde{\mathbb{D}}_\alpha$ and $\widetilde{\mathbb{D}}_\gamma$ from $\widetilde{\mathbb{D}}$. $\widetilde{\mathbb{D}}_\alpha$ and $\widetilde{\mathbb{D}}_\gamma$ are generated by distributing the secret shares of the block content of $\widetilde{\mathbb{D}}$ (but keeping the tags intact). S_α and S_γ are responsible for storing $\widetilde{\mathbb{D}}_\alpha$ and $\widetilde{\mathbb{D}}_\gamma$, respectively.

While processing the PIR client's request, our protocol determines T_I, the tag of the requested block, without leaking any information about the request to any server, and then fetches the block shares associated with the tag from $\widetilde{\mathbb{D}}_\alpha$ and $\widetilde{\mathbb{D}}_\gamma$. During the fetching process, $\mathsf{S}_{\mathsf{i}\in\{\alpha,\gamma\}}$ can see the plaintext value of T_I. However, they are unable to deduce any details regarding the PIR-request, I, since the tag was originally generated using a one-way function. Furthermore, due to the secret-sharing, $\mathsf{S}_{\mathsf{i}\in\{\alpha,\gamma\}}$ also cannot learn anything about the actual block content.

This scheme also incorporates a shelter, $\mathsf{sh}[]$, and first searches for the requested item within the shelter. Only if not found, the actual tag, T_I, is provided to $\mathsf{S}_{\mathsf{i}\in\{\alpha,\gamma\}}$. Conversely, if the item is found in the shelter, our protocol delivers a previously unseen tag, T_ϕ, to $\mathsf{S}_{\mathsf{i}\in\{\alpha,\gamma\}}$. This T_ϕ is associated with a dummy block within $\widetilde{\mathbb{D}}_{\mathsf{i}\in\{\alpha,\gamma\}}$.

However, the structure of the shelter is modified from our previous schemes. Instead of storing the FHE-ciphertexts of previously accessed items, the shelter contains their masked values. Two identical replicas of $\mathsf{sh}[]$ are maintained across $\mathsf{S}_{\mathsf{i}\in\{\alpha,\gamma\}}$ and DPF is leveraged to perform an oblivious search for the corresponding masked item. Each item in $\mathsf{sh}[]$ is also accompanied by a unique plaintext tag (but different from the tags in $\widetilde{\mathbb{D}}_{\mathsf{i}\in\{\alpha,\gamma\}}$), which acts as the *keyword* during the DPF-based shelter search operation.

S_β also generates a mask database, $\mathcal{M}$, containing $\sqrt{\mathsf{N}}$-random masks. S_β shares this mask database with S_δ and S_ϵ. $\mathsf{S}_{\mathsf{i}\in\{\delta,\epsilon\}}$ independently perform DPF-based searching on $\mathcal{M}$, to obliviously figure out the mask corresponding to the found item. Then our protocol,

in a server-oblivious way, combines the found mask along with the found shelter item to derive the plaintext response.

As a result, compared to our *Basic Scheme*, this scheme completes the shelter searching operation by conducting $O(\sqrt{\mathsf{N}})$ inexpensive DPF evaluations, rather than $O(\sqrt{\mathsf{N}})$ expensive homomorphic evaluations.

High-Level Flow of Request Processing

The high-level flow of the processing of a PIR request in this scheme is shown in Figure 3.8. After receiving a PIR request, our scheme first searches for the item in the shelter. To do this, corresponding shelter tag, $\widehat{\mathsf{T}_\mathsf{I}}$, is computed (Figure 3.8, Phase 1). In this phase, $\mathsf{S}_\alpha, \mathsf{S}_\beta, \mathsf{S}_\gamma$ and the PIR-client, $\mathbb{C}$ - collaborate and S_β learns $\widehat{\mathsf{T}_\mathsf{I}}$.

S_β utilizes DPF to search for $\widehat{\mathsf{T}_\mathsf{I}}$ within the shelter (Phase 2). During this search process, actually two rounds of DPF evaluations occur. The first one occurs between S_α and S_γ, who determine the secret shares of the masked response. The second round occurs between S_δ and S_ϵ, who determine the secret shares of the mask.

Because of the security properties of DPF, none of the servers learns about the search result. Our protocol obliviously combines the shelter search result and the mask. Then the FHE encrypted shelter search result, $\overline{\mathsf{SR}}_\mathsf{sh}$ is revealed to S_α. $\overline{\mathsf{SR}}_\mathsf{sh}$ is the ciphertext of the found item, or an encrypted dummy value, when the item is not found within the shelter.

S_α utilizes homomorphic evaluation to obliviously select the tag T_* (Phase 3). The associated block shares are next fetched from $\widetilde{\mathbb{D}}_{\mathsf{i}\in\{\alpha,\gamma\}}$ (Phase 4). Our protocol ensures that, S_α obliviously assigns the actual tag, T_I to T_*, if the item is not found in the shelter. In that case, the shares of the actual requested item are fetched. Otherwise, the tag of an untouched dummy block, T_ϕ, is assigned to T_* and shares of a dummy block are retrieved.

In Phase 5, S_γ combines the fetched shares together. Our protocol uses homomorphic evaluation during this combination process. Therefore, S_γ can neither learn what the combined block value is, nor can it determine whether the block is a dummy or not. By the end of this combination, S_γ produces $\overline{\mathsf{SR}}_{\widetilde{\mathbb{D}}}$, which is the FHE ciphertext of the combined fetched block from the shuffled database.

Between the two ciphertexts, $\overline{\mathsf{SR}}_\mathsf{sh}$ (produced at the end of the shelter searching) and $\overline{\mathsf{SR}}_{\widetilde{\mathbb{D}}}$, only one actually corresponds to the requested block, with the other one just being an encrypted dummy block. S_γ selects the correct one, again, by using the homomorphic evaluation (Phase 6). As a result, S_γ cannot determine from among $\overline{\mathsf{SR}}_\mathsf{sh}$ and $\overline{\mathsf{SR}}_{\widetilde{\mathbb{D}}}$, which ciphertext got actually selected.

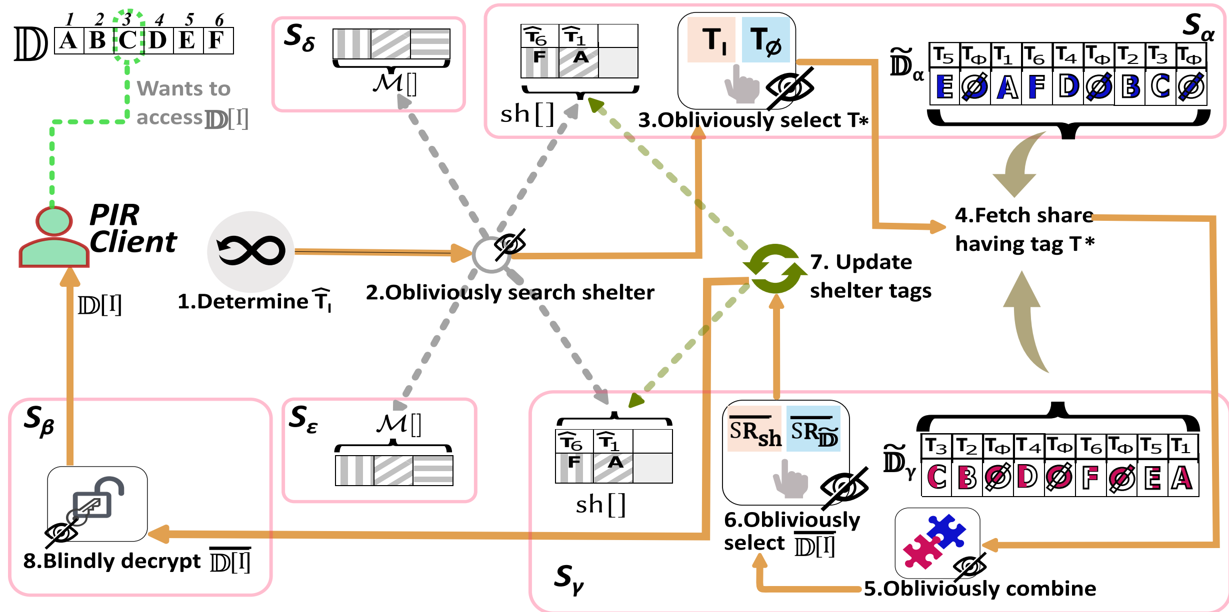


Figure 3.8: High-Level Flow of Request Processing

Next, our protocol updates both replicas of the shelter by appending the masked response value to sh[] and attaching a newly-calculated unique shelter tag, $\widehat{\mathsf{T}}_*$, to it. Since a different mask is used to mask each element of sh[], no repetition within sh[] can be observed. Our protocol also re-randomizes the tags of all existing shelter elements after each request (Phase 7). It ensures that S_β does not encounter any repetition while processing the next request (i.e., specifically during the Phase 1 for the next request). Finally, the response ciphertext is blindly decrypted by S_β and the plaintext result is delivered to the client (Phase 8), and the corresponding masked value is appended to the shelter.

This scheme also begins with an initialization operation, and specific operations must be performed once per epoch. Following these per-epoch operations, the scheme processes requests from PIR clients, as outlined in the high-level flow diagram in Figure 3.8. In the sections that follow, we will explore the details of each of them. Additionally, we will examine the structure of the shuffled database, the mask database, the shelter, and the tags used. A summary of all the notation for this scheme can be found in Table A.2.

3.4.3 One-time Initialization

We use $\mathbb{G}$, which is a q-order cyclic subgroup of $\mathbb{Z}_{\mathsf{p}}^*$, for tag creation, where p is a prime number and q is a safe prime. In this scheme, we use the El-Gamal encryption scheme [72], to cheaply perform homomorphic multiplication and exponentiation (see appendix C). Specifically, we use $\mathcal{E}$ & $\mathcal{E}_{\mathsf{q}}$, the El-Gamal encryption scheme in two different settings. $\mathcal{E}$ is the El-Gamal encryption scheme working in $\mathbb{G}$, while $\mathcal{E}_{\mathsf{q}}$ works in $\mathbb{Z}_{\mathsf{q}}^*$. While the El-Gamal encryption scheme in a non-prime order group is generally considered insecure, we will discuss later, why using $\mathcal{E}_{\mathsf{q}}$ does not pose any security concerns in our scheme.

During the one-time initialization process, depending on the required computational security, the bit-lengths of p & q are chosen ($\|\mathsf{p}\|$ & $\|\mathsf{q}\|$, respectively) and $\mathbb{G}$ is defined. Subsequently, $\mathsf{g} \in \mathbb{G}$ & $\mathsf{g}_{\mathsf{q}} \in \mathbb{Z}_{\mathsf{q}}^*$ are chosen randomly as the generators. Our scheme chooses another prime number r, of bit-length $\|\mathsf{r}\|$ (where $\|\mathsf{r}\| < \|\mathsf{q}\|$) for further improving the efficiency of DPF-evaluations. After initialization, $\mathsf{p}, \mathsf{q}, \mathbb{G}, \mathsf{g}, \mathsf{g}_{\mathsf{q}}$ & r remain fixed.

S_β uses $\mathsf{p}, \mathsf{q}, \mathsf{g}$& g_{q} to generate two El-Gamal key-pairs $(\mathsf{pk}_{\mathcal{E}}, \mathsf{sk}_{\mathcal{E}}) \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^\lambda)$ and $(\mathsf{pk}_{\mathcal{E}_{\mathsf{q}}}, \mathsf{sk}_{\mathcal{E}_{\mathsf{q}}}) \leftarrow \mathcal{E}_{\mathsf{q}}.\mathsf{KeyGen}(1^\lambda)$. Subsequently S_β publishes the public keys $\mathsf{pk}_{\mathcal{E}}$ and $\mathsf{pk}_{\mathcal{E}_{\mathsf{q}}}$. During the initialization, S_β also generates the FHE-key pair $(\mathsf{pk}_{\mathsf{F}}, \mathsf{sk}_{\mathsf{F}}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$ and publishes pk_{F}. This FHE public key can be used for both encryption and homomorphic evaluation purposes. In our scheme, only S_β knows the secret keys, $\mathsf{sk}_{\mathcal{E}}, \mathsf{sk}_{\mathcal{E}_{\mathsf{q}}}$ & sk_{F}.

3.4.4 Shuffled Database, Mask Database, Shelter and Tag Details

Each of the secret-shared shuffled databases, $\widetilde{\mathbb{D}}_{\mathsf{i} \in \{\alpha,\gamma\}}$, consists of $(\mathsf{N} + \sqrt{\mathsf{N}})$-elements, N of which represent real blocks, with the remainder being dummy blocks. S_α and S_γ store the databases $\mathbb{D}_\alpha$ and $\mathbb{D}_\gamma$, respectively. Each element of these databases is a tuple having structure $(\mathsf{T}_\mathsf{I}, \langle \mathbb{D}[\mathsf{I}] \| \mathsf{I} \rangle)$. T_I is a unique plaintext tag of size $\|\mathsf{p}\|$-bits. The second element of the tuple represents a secret share of $(\mathbb{D}[\mathsf{I}] \| \mathsf{I})$, which is the concatenated representation of the B-bit block content and $(\log \mathsf{N})$-bit block index. Since $\mathsf{I} = 0$ is not a valid block index, dummy blocks are represented as $\{0\}^{\mathsf{B} + \log \mathsf{N}}$. Note that the combined shuffled database, $\widetilde{\mathbb{D}}$, is a temporary database. S_β only requires this while preparing $\widetilde{\mathbb{D}}_{\mathsf{i} \in \{\alpha,\gamma\}}$.

S_δ & S_ϵ hold two identical replicas of the mask database, $\mathcal{M}$, having $\sqrt{\mathsf{N}}$-elements. Each element of $\mathcal{M}$ is a random mask of size $(\mathsf{B} + \log \mathsf{N})$-bits, generated by S_β. Two identical replicas of the shelter, $\mathsf{sh}[]$, are maintained in S_α & S_γ. A new element is appended to both shelter replicas after each request is processed. Since the epoch size of this scheme is $\sqrt{\mathsf{N}}$, the size of $\mathsf{sh}[]$ increases up to $\sqrt{\mathsf{N}}$. If the block having the index I was returned while processing i^{th} request ($\mathsf{i} \in [1, \sqrt{\mathsf{N}}]$) of the epoch, then $\mathsf{sh}[(\mathsf{i}-1)]$ is a tuple of the form

$(\widehat{\mathsf{T}}_\mathsf{I}, \widehat{\mathsf{t}}_\mathsf{I}, \ddot{\mathsf{d}}_\mathsf{I})$. $\ddot{\mathsf{d}}_\mathsf{I}$ represents the masked representation of $(\mathbb{D}[\mathsf{I}]||\mathsf{I})$ (i.e., $\ddot{\mathsf{d}}_\mathsf{I} = (\mathbb{D}[\mathsf{I}]||\mathsf{I}) \oplus \mathcal{M}[\mathsf{i}]$). $\widehat{\mathsf{T}}_\mathsf{I}$ is a tag having size $\|\mathsf{p}\|$-bits and $\widehat{\mathsf{t}}_\mathsf{I}$ is the shorter ($\|\mathsf{r}\|$-bit) tag, derived from $\widehat{\mathsf{T}}_\mathsf{I}$.

Tag Details

The database tag, T_I, and the shelter tag, $\widehat{\mathsf{T}}_\mathsf{I}$, both represent the index I. The database tag, T_I, is generated using a one-way function on the index I, to ensure that the storing servers ($\mathsf{S}_{\mathsf{i}\in\{\alpha,\gamma\}}$) cannot ascertain the actual index represented by these tags. The used one-way function must be both deterministic and collision-resistant. This requirement stems from the fact that $\mathsf{S}_{\mathsf{i}\in\{\alpha,\gamma\}}$ must uniquely identify the requested block's secret share within $\widetilde{\mathbb{D}}_{\mathsf{i}\in\{\alpha,\gamma\}}$. The tag generation function also needs to be computed with each request. However, the client must not be required to disclose the requested index, I, to any servers, nor should the servers share the function's secret parameters with the client.

The tag generation function: We observe that certain exponentiation-based one-way functions exist, whose security can be established under known hardness assumptions. Furthermore, it is feasible to utilize the semi-homomorphic exponentiation property of El-Gamal encryption to compute these functions in an oblivious yet cost-effective manner.

We drew inspiration from Naor-Reingold's pseudorandom generator [73]:

$\mathsf{f_a}(\mathsf{x}) = \mathsf{g}^{\mathsf{a_0}.\mathsf{a_1}^{\mathsf{x_1}}.\mathsf{a_2}^{\mathsf{x_2}}\ldots\mathsf{a_n}^{\mathsf{x_n}}}$

Notably, this can also function as a one-way function by treating $\mathsf{a_0}, \mathsf{a_1}..\mathsf{a_n}$, as the components of the secret parameter and $\mathsf{x_1}, \mathsf{x_2}..\mathsf{x_n}$, as the message bits. However, unlike original specification, we do not mandate pseudorandomness, since the adversary is aware that the tags will not be random in any case. Therefore, we have selected the tag generation function as $\mathsf{T}_\mathsf{I} = (\mathsf{g}^{(\rho^\mathsf{I}) \bmod \mathsf{q})}) \bmod \mathsf{p}$, where $\rho \in \mathbb{Z}_\mathsf{q}^*$ is the secret parameter of the function and it is only known to S_β. Since there is no randomization involved, $\mathsf{g}^{\rho^\mathsf{I}}$ produces a unique and deterministic tag. Hence, $\mathsf{g}^{\rho^\mathsf{I}}$ fulfills our requirements.

Although S_β knows the secret parameter, ρ, our protocol ensures that S_β cannot observe the function output, $\mathsf{g}^{\rho^\mathsf{I}}$. As a result, S_β cannot learn anything about the function input, I, either. Conversely, $\mathsf{S}_{\mathsf{i}\in\{\alpha,\gamma\}}$ does have access to the function output. However, in order to derive I from it, $\mathsf{S}_{\mathsf{i}\in\{\alpha,\gamma\}}$ must resolve two discrete logarithm problems. Due to the recognized difficulty of this task, $\mathsf{g}^{\rho^\mathsf{I}}$ acts as a one-way function. Moreover, because $\mathsf{S}_{\mathsf{i}\in\{\alpha,\gamma\}}$ lacks knowledge of ρ, they cannot generate $\mathsf{g}^{\rho^\mathsf{I}}$ for any arbitary I, without navigating through our protocol. Consequently, they are unable to pre-compute all potential tags and perform a lookup attack.

Our protocol ensures that $\mathsf{S}_{i\in\{\alpha,\gamma\}}$, does not come across any repeated tags. However, they might learn multiple index-tag pairs by using fake clients. Consequently, when processing a legitimate client request, they may try to determine if the requested index has any *mathematical* relation to the known indices.

For instance, suppose the adversary has already seen the tags $\mathsf{T_a} = \mathsf{g^A}$ and $\mathsf{T_b} = \mathsf{g^B}$, which correspond to the known indices $\mathsf{I_a}$ and $\mathsf{I_b}$, where $\mathsf{A} = \rho^{\mathsf{I_a}}$ & $\mathsf{B} = \rho^{\mathsf{I_b}}$. Later on, while processing the targeted client's request, the adversary comes across the tag $\mathsf{T_c}$. Let's assume that the unknown client request, $\mathsf{I_c}$, is mathematically linked to the known indices as: $\mathsf{I_c} = \mathsf{I_a} + \mathsf{I_b}$. Now, the adversary may attempt to determine if the tag $\mathsf{T_c}$ exhibits any mathematical relationship with the previously encountered tags. In this case, there is indeed a mathematical relationship, and $\mathsf{g}^{\rho^{\mathsf{I_c}}} = \mathsf{g}^{\rho^{\mathsf{I_a}+\mathsf{I_b}}} = \mathsf{g}^{\rho^{\mathsf{I_a}}.\rho^{\mathsf{I_b}}} = \mathsf{g^{AB}}$.

Notice, this is exactly the DDH problem, and due to the established difficulty of this problem, the adversary will be unable to discern the mathematical relationships among the indices from the observed tags. Furthermore, because of the hardness of the CDH problem, the adversary will not be able to compute $\mathsf{g}^{\rho^{\mathsf{I_c}}}$ from $\mathsf{g^A}$ and $\mathsf{g^B}$. As a result, our tag generation function remains unforgeable as well.

Shelter Tags:

In addition to being generated in a collision-resistant manner, shelter tags must also be updated after processing each request. Otherwise, S_β might be able to detect repeated requests (during Phase 1 of Figure: 3.8). Therefore, our scheme computes the shelter tag of block I as $\widehat{\mathsf{T_I}} = (\mathsf{T_I.a.b.c}) \bmod \mathsf{p}$, where a, b & c are random elements of $\mathbb{G}$, with $\mathsf{S}_\alpha, \mathsf{S}_\beta$ & S_γ changing each of them, respectively, during each request. Because $\widehat{\mathsf{T_I}} \in \mathbb{G}$ has a size of $\|\mathsf{p}\|$-bits, the efficiency of DPF evaluations might be affected. To address this issue, our scheme attaches an additional shorter tag, $\widehat{\mathsf{t_I}}$, with each shelter item. This $\|\mathsf{r}\|$-bit long tag is computed as $\widehat{\mathsf{t_I}} = \widehat{\mathsf{T_I}} \bmod \mathsf{r}$.

3.4.5 Per Epoch Operations

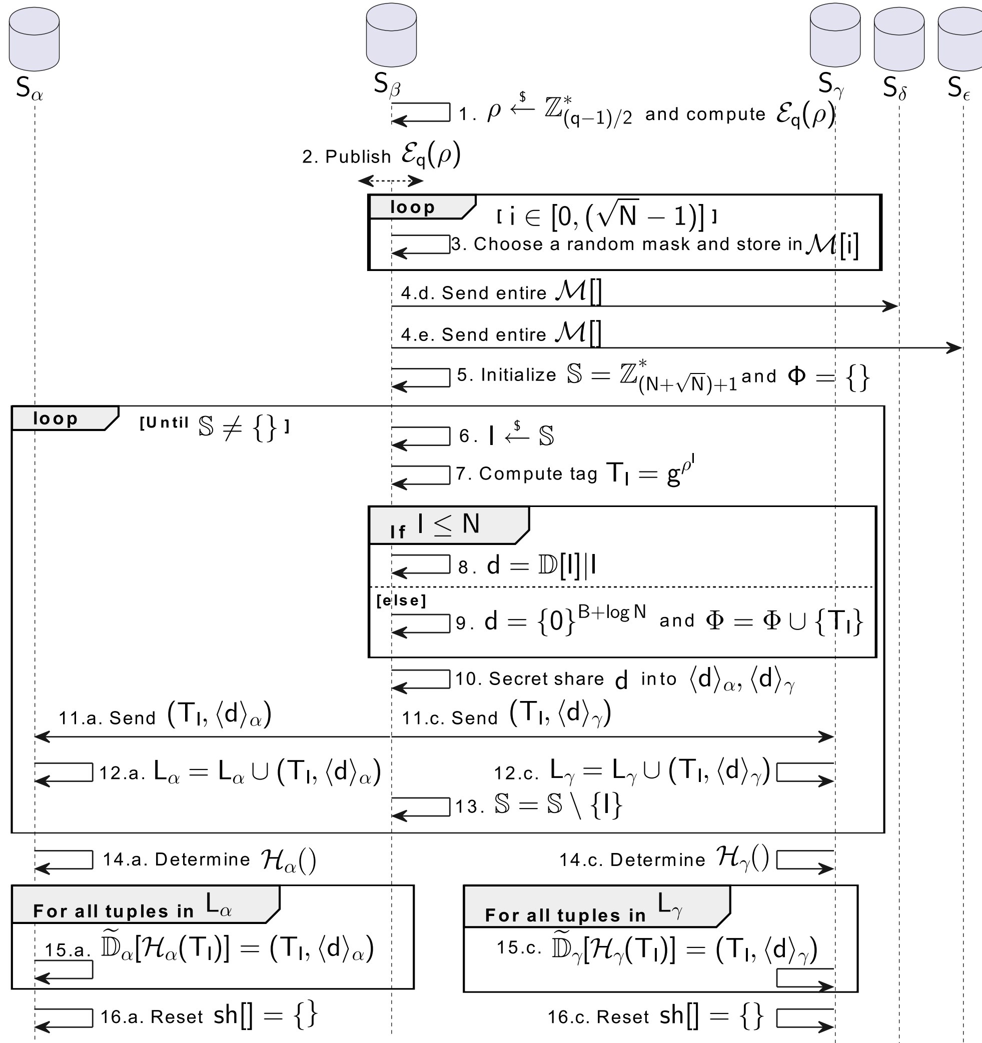


Figure 3.9: Per Epoch Operations

At the beginning of each epoch, $\widetilde{\mathbb{D}}_{i\in\{\alpha,\gamma\}}$ are re-initialized and the shelter replicas are emptied as shown in Figure 3.9. First, S_β randomly chooses a new secret parameter for the one-way function, $\rho \in \mathbb{Z}^*_{(q-1)/2}$ and publishes its El-Gamal ciphertext, $\mathcal{E}_q(\rho)$ (Figure 3.9, Step 1-2). While El-Gamal encryption in a non-prime order group is typically not considered secure, this concern is not applicable in our context. First, q is selected as a safe

prime, rendering the Pohlig-Hellman attack [74] irrelevant. Additionally, because ρ is chosen from the quadratic residue group, the adversary (the PIR client in this context) cannot extract even a single bit of ρ from $\mathcal{E}_{\mathsf{q}}(\rho)$.

Next, S_β generates $\sqrt{\mathsf{N}}$-number of random $(\mathsf{B} + \log \mathsf{N})$-bit masks and shares them with S_δ and S_ϵ. Subsequently, the tags and additive secret shares of all the blocks (either real or dummy) are created and sent to $\mathsf{S}_{\mathsf{i} \in \{\alpha,\gamma\}}$, in a random order. Dummy blocks also contain distinct tags. Index values, $\mathsf{N} < \mathsf{I} \leq (\mathsf{N} + \sqrt{\mathsf{N}})$ are used to compute the dummy tags. S_β keeps track of these dummy tags in a special set, Φ (Step 9).

Later, during the request processing phase, $\mathsf{S}_{\mathsf{i} \in \{\alpha,\gamma\}}$ searches for the requested tag values in $\widetilde{\mathbb{D}}_{\mathsf{i} \in \{\alpha,\gamma\}}$ and returns the corresponding secret shares. To make the searching process faster, $\mathsf{S}_{\mathsf{i} \in \{\alpha,\gamma\}}$ utilizes cuckoo hashing with stash (see appendix: D), to map specific tags to specific locations within $\widetilde{\mathbb{D}}_{\mathsf{i} \in \{\alpha,\gamma\}}$. $\mathsf{S}_{\mathsf{i} \in \{\alpha,\gamma\}}$ initially collects all the received tuples in a temporary list, $\mathsf{L}_{\mathsf{i} \in \{\alpha,\gamma\}}$ and then builds the hash-tables based on all the tag values (Step 14.a and 14.c). Received tuples are then stored at locations determined by computing the cuckoo hash on their tag part. Finally, both $\mathsf{S}_{\mathsf{i} \in \{\alpha,\gamma\}}$ clear their existing shelter contents.

After performing the per-epoch operations, our scheme can process PIR requests, as outlined in section 3.4.2. All phases of Figure 3.8 are now described in detail.

3.4.6 Determination of the shelter-tag to be searched (Phase 1 of Figure 3.8):

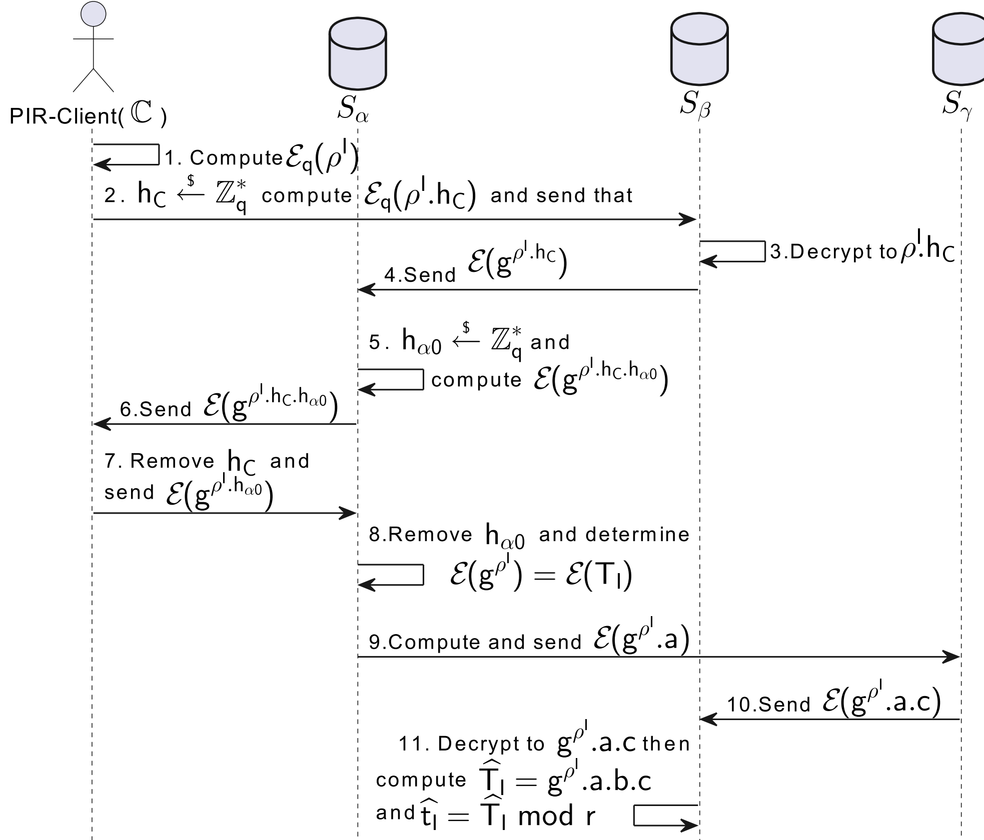


Figure 3.10: Shelter-tag determination

During this phase, $\mathbb{C}, \mathsf{S}_\alpha, \mathsf{S}_\beta$ & S_γ collaborate (Figure 3.10), without requiring anyone to reveal their secret to others. Using El-Gamal encryption's homomorphic properties, $\mathbb{C}$ first computes $\mathcal{E}_\mathsf{q}(\rho^\mathsf{l})$ from its secret request, l, and the publicly available ciphertext $\mathcal{E}_\mathsf{q}(\rho)$. Since ρ^l will be used as an exponent of g and the order of g in $\mathbb{G}$ is q, ρ^l can only be wrapped around using $\bmod \mathsf{q}$. Hence, $\mathcal{E}_\mathsf{q}$ is used in this case, instead of $\mathcal{E}$.

Previously, we observed that the insecurity of $\mathcal{E}_\mathsf{q}(\rho)$ can be mitigated by selecting ρ from the quadratic residue group. However, this does not hold true in the current scenario because ρ^l can take on any value, depending on l. Nevertheless, this also does not pose a

problem in our scheme, as only S_β has access to the ciphertext. Since S_β already possesses the decryption key, it will not gain any additional advantage from this insecurity.

However, transmitting $\mathcal{E}_\mathsf{q}(\rho^\mathsf{I})$ to S_β may pose challenges, as this would enable S_β to decrypt and access the plaintext exponent (ρ^I) associated with the request. In turn, this, would allow S_β to trace back to the requested index, I, through a simple lookup. To mitigate this risk, $\mathbb{C}$ homomorphically blinds ρ^I using a random h_C (Step 2). After receiving $\mathcal{E}_\mathsf{q}(\rho^\mathsf{I} \cdot \mathsf{h}_\mathsf{C})$, S_β proceeds to decrypt the value, raises the plaintext to the generator g, and subsequently re-encrypts the resulting value, this time employing $\mathcal{E}$ (Step 3 and Step 4).

An immediate return of the resulting ciphertext directly to $\mathbb{C}$ will enable a client to compute $\mathsf{I} \longleftrightarrow \mathcal{E}(\rho^\mathsf{I})$ pairs. A fake client could exploit this capability. Hence, S_β sends $\mathsf{g}^{\rho^\mathsf{I} \cdot \mathsf{h}_\mathsf{C}}$ to S_α, so that S_α can add its own blinding exponent, $\mathsf{h}_{\alpha 0}$, before returning the blinded result to $\mathbb{C}$ (Step 5). Upon reception, $\mathbb{C}$ removes h_C from the exponent by homomorphically raising the ciphertext to $\mathsf{h}_\mathsf{C}^{-1}$ and returns the result to S_α. Subsequently, S_α homomorphically removes $\mathsf{h}_{\alpha 0}$ from the exponent and obtains the El-Gamal ciphertext of the shuffled database tag.

Regarding the corresponding shelter-tag, $\mathsf{S}_\alpha, \mathsf{S}_\beta$ and S_γ respectively manage $\mathsf{a}, \mathsf{b},$ and c, which are their own per-request randomness. After computing $\mathcal{E}(\mathsf{T}_\mathsf{I})$, S_α performs a homomorphic multiplication with its own per-request randomness, a (Step 9). Subsequently, S_γ homomorphically multiplies its own random value, c, and sends the resulting ciphertext to S_β. At this point, S_β decrypts the message and multiplies it with the randomness, b, in plaintext space to determine $\widehat{\mathsf{T}_\mathsf{I}}$. Finally $\widehat{\mathsf{t}_\mathsf{I}}$ is computed from $\widehat{\mathsf{T}_\mathsf{I}}$ (Step 11).

3.4.7 Oblivious Shelter Search (Phase 2):

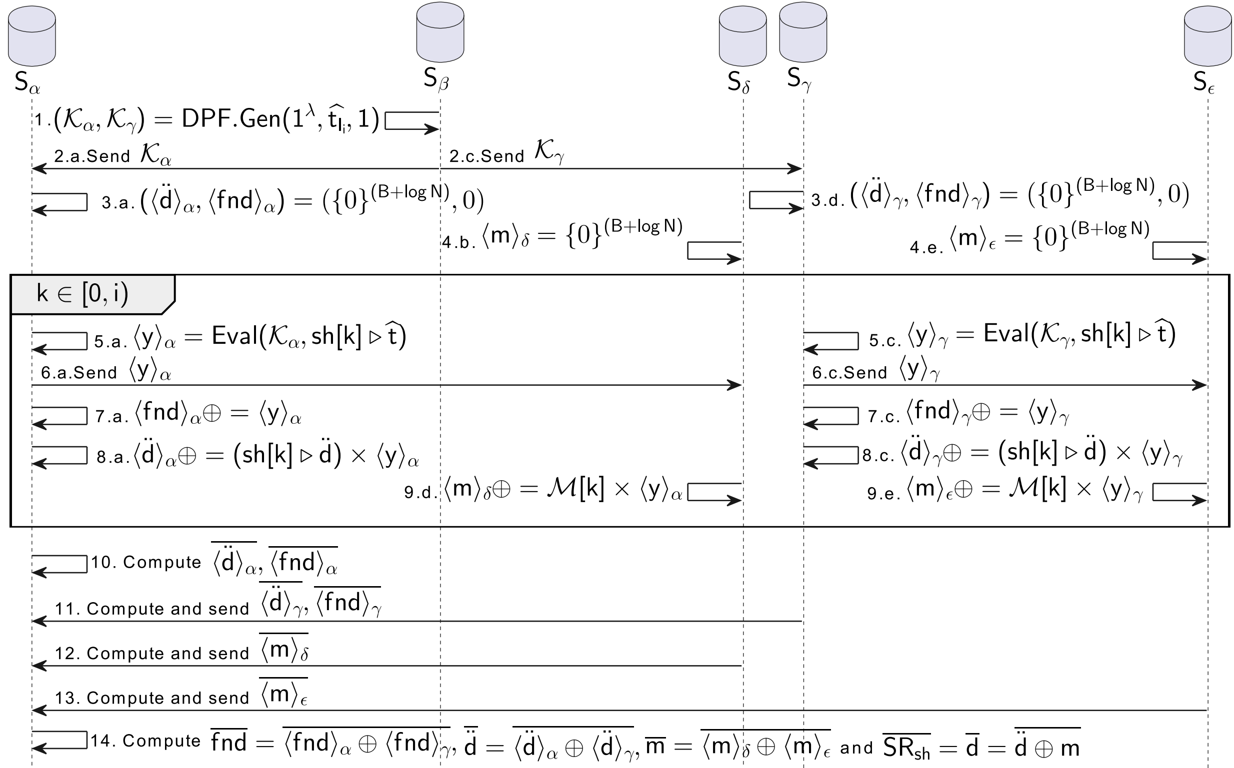


Figure 3.11: Obliviously search shelter

Next (refer to Figure 3.11), a DPF-based private keyword search [68] is performed, to check whether $\widehat{t_l}$ is present within $sh[]$. For this, S_β generates two DPF evalution keys $\mathcal{K}_{i \in \{\alpha,\gamma\}}$ from the search tag $\widehat{t_l}$ (Step 1). Then $S_{i \in \{\alpha,\gamma\}}$ uses $\mathcal{K}_{i \in \{\alpha,\gamma\}}$ and performs DPF-evaluations on the tag portion $(sh[k] \triangleright \widehat{t})$ of all existing shelter elements (Step 4). We choose the range of the $\mathsf{Eval}()$ algorithm to be $\mathbb{Z}_2$. Hence after each evaluation (Step 5), $S_{i \in \{\alpha,\gamma\}}$ observes a single bit. The security of the DPF scheme ensures that this evaluation output bit is indistinguishable from a random coin toss.

S_α and S_γ also share all the evaluation output bits with S_δ and S_ϵ, respectively (Step 6). By the end of processing all the shelter elements, $S_{i \in \{\alpha,\gamma\}}$ locally generate $\langle fnd \rangle_{i \in \{\alpha,\gamma\}}$ & $\langle \ddot{d} \rangle_{i \in \{\alpha,\gamma\}}$. Where, $\langle fnd \rangle_{i \in \{\alpha,\gamma\}}$ are the secret shares of the search result. When these shares are combined, they produce the bit fnd, which takes on the value of 1 only if $\widehat{t_l}$ is found within $sh[]$.

$\langle \ddot{\mathsf{d}} \rangle_{i \in \{\alpha,\gamma\}}$ are the secret shares of $\ddot{\mathsf{d}}$, which is found item from the shelter (i.e., the masked response). By utilizing the received DPF evaluation output bits, $\mathsf{S}_{i \in \{\delta,\epsilon\}}$ generate $\langle \mathsf{m} \rangle_{i \in \{\delta,\epsilon\}}$ (Step 8), which are the secret shares of the mask, m, corresponding to the found item from the shelter.

Next, $\mathsf{S}_\gamma, \mathsf{S}_\delta$ and S_ϵ send their own shares to S_α (Step 11-13) after encrypting them with the FHE public-key. S_α combines them homomorphically (Step 14) with its own shares and determines the ciphertexts of the shelter search result, $\overline{\mathsf{fnd}}$ & $\overline{\mathsf{SR_{sh}}} = \overline{\mathsf{d}}$. If the item is not found in the shelter, $\overline{\mathsf{fnd}}$ & $\overline{\mathsf{SR_{sh}}}$ holds the ciphertexts of zeros (i.e., $\overline{0}$ & $\overline{0^{\mathsf{B}+\log \mathsf{N}}}$). The security of DPF scheme and the IND-CPA security of FHE ensures that, irrespective of the outcome of the shelter search operation, none of the servers learn anything.

3.4.8 Oblivious tag selection for the shuffled databases (Phase 3):

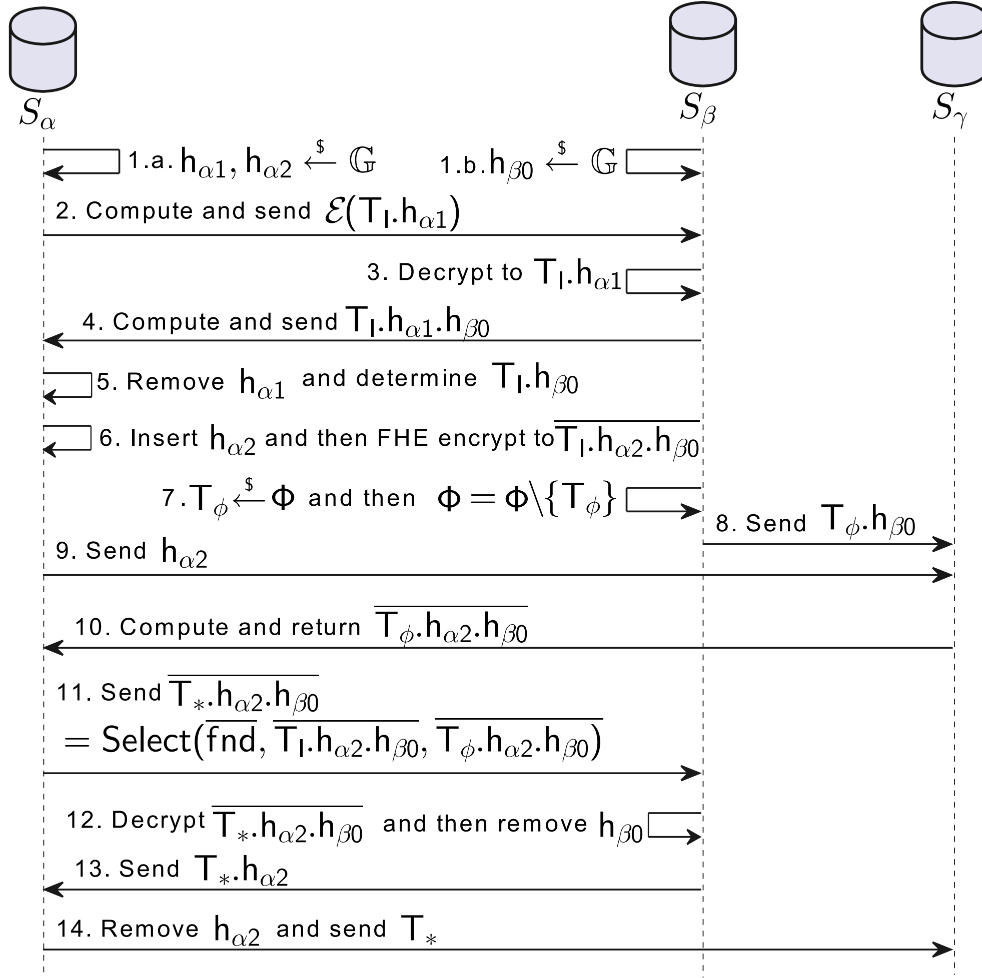


Figure 3.12: Select shuffled database fetch tag

In this phase, depending on the shelter search result, S_α obliviously selects between the real tag, T_l, and the dummy tag, T_ϕ. Initially (Steps 1-5, in Figure 3.12), S_β transforms the S_α possessed $\mathcal{E}(T_l)$ to a blinded plaintext version, $T_l.h_{\beta0}$. Homomorphic property of El-Gamal encryption is used here to ensure that T_l remains hidden from S_β, in this process.

Next (Steps 6-10), both T_l and T_ϕ are blinded using the same blinding factor, before

encrypting them with $\mathsf{pk_F}$. Subsequently, S_α applies the homomorphic selection function on the resulting ciphertexts and the encrypted shelter search result bit, $\overline{\mathsf{fnd}}$. After executing the selection, S_α obtains the FHE ciphertext of the blinded T_*. Finally, S_α and S_γ learn the plaintext T_*, after the decryption and unblinding operations (Steps 12-14). This T_* is used next to identify the shares to be fetched from $\widetilde{\mathbb{D}}_{\mathsf{i}\in\{\alpha,\gamma\}}$.

3.4.9 Fetch-combine shares and then select response (Phase 4-6):

After obtaining T_*, $\mathsf{S}_{\mathsf{i}\in\{\alpha,\gamma\}}$ examines their respective cuckoo hash table to locate the corresponding content shares (Figure 3.13, Step 1). They individually fetch the shares from the identified locations and encrypt them using FHE. Because our scheme guarantees that T_* is unique and never reused, no location in $\widetilde{\mathbb{D}}_{\mathsf{i}\in\{\alpha,\gamma\}}$ is accessed more than once.

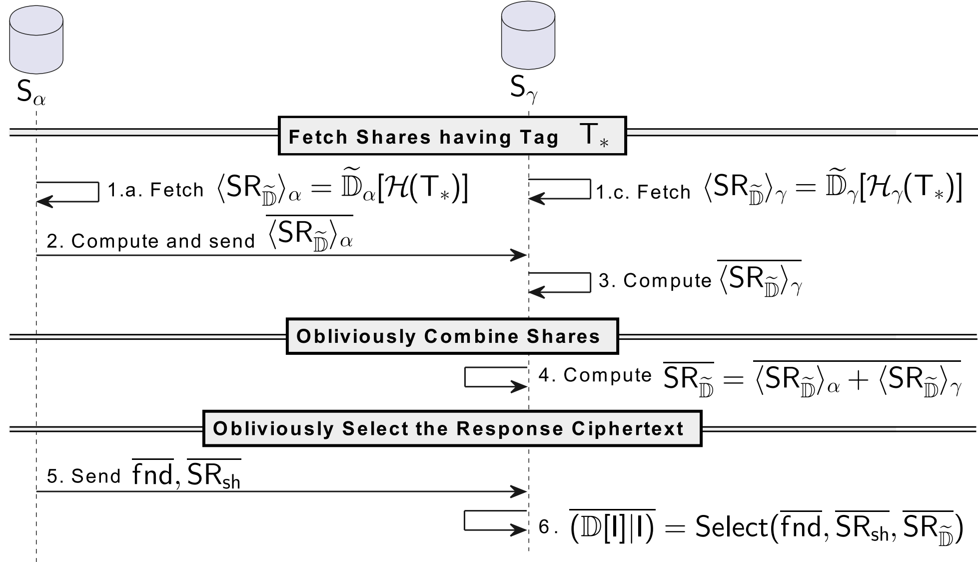


Figure 3.13: Fetch-combine and select

Next, S_γ obliviously combines the ciphertexts of the shares with homomorphic addition (Step 4). Finally, S_γ makes an oblivious choice between this newly-formed ciphertext and the previously-determined encrypted shelter-search result, $\overline{\mathsf{SR_{sh}}}$, utilizing the homomorphic select function (Step 6).

3.4.10 Update shelter tags (Phase 7):

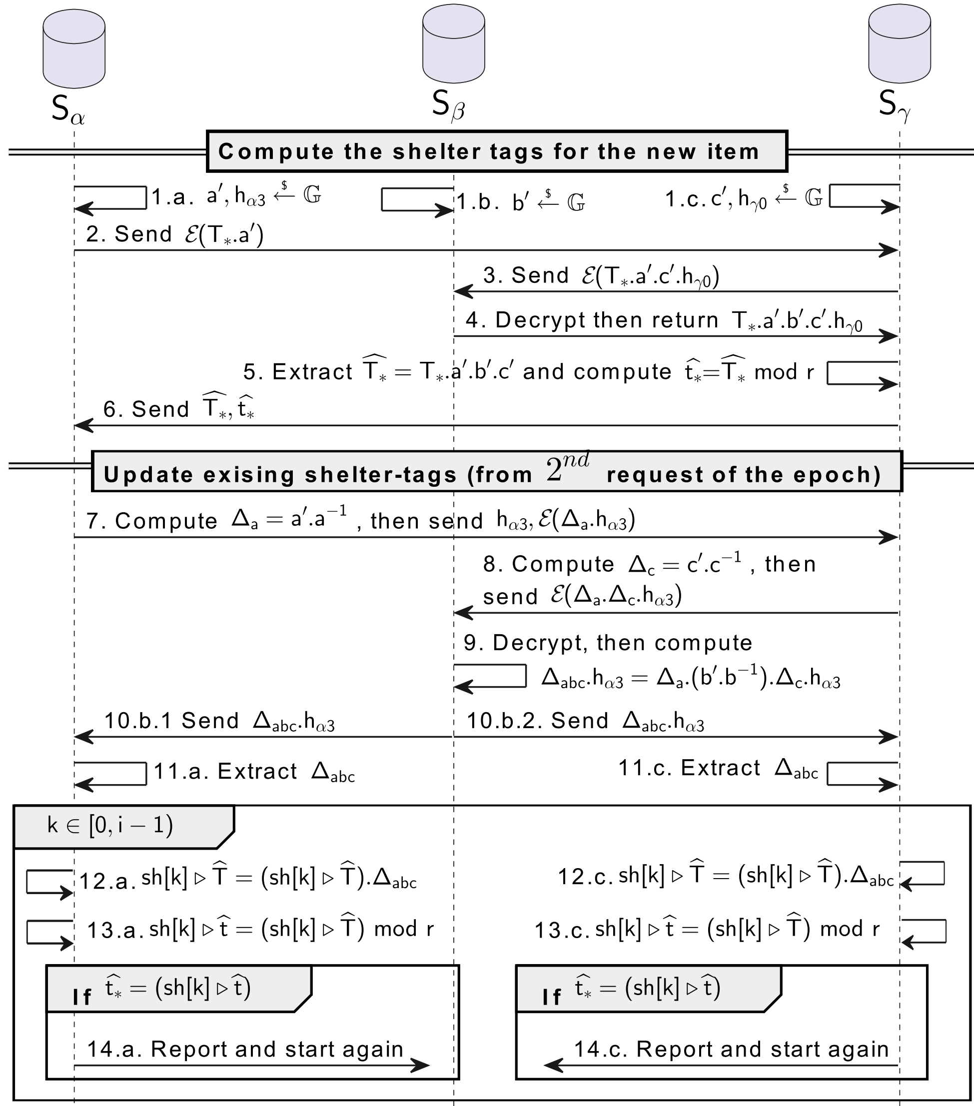


Figure 3.14: Shelter update

Before adding the masked response at $\mathsf{sh}[i]$, its corresponding shelter tags are calculated. To remove any possibility of repetition in the plaintext tag portion of the shelter elements, T_* is utilized (guaranteed to be unique) to determine the shelter tag ($\widehat{T_*}$ & $\widehat{t_*}$) of the newly

added element. By leveraging the blinding and homomorphic properties of the El-Gamal encryption scheme, S_α& S_γ learn the tags of the new element, without revealing anything to S_β (Figure 3.14 steps 1-6).

Following this, all existing shelter tags are revised, incorporating newly selected randoms: $\mathsf{a}', \mathsf{b}', \mathsf{c}' \in \mathbb{G}$ (Steps 7-14). To achieve this, $\mathsf{S}_{\mathsf{i}\in\{\alpha,\gamma\}}$ calculate the *ratio* between the current and previous joint randomization, applying this ratio to adjust all existing shelter tags. In the context of $\mathbb{G}$, this ratio is: $\Delta_{\mathsf{abc}} = (\mathsf{a}'.\mathsf{b}'.\mathsf{c}').(\mathsf{a}.\mathsf{b}.\mathsf{c})^{-1} \bmod \mathsf{p}$. To compute Δ_{abc}, the homomorphic properties of El-Gamal encryption and blinding are once again employed to safeguard the confidentiality of each individual's secret random values.

Because existing shelter tags are unique, multiplying them all by the same value will not introduce repetitions. However, due to the wrap-around nature of the $\bmod \mathsf{r}$ operation, the newly computed shorter tag, $\widehat{\mathsf{t}_*}$, could potentially collide. While such collisions are unlikely, with a probability estimated to be $< 10^{-10}$ [75], they can still occur. If they do, all three servers will restart this phase using different random numbers.

3.4.11 Delivering Response and Append to the Shelter(Phase 8):

At the end of the earlier Phase 6 (See step 6 of Figure 3.13), S_γ obtains the FHE ciphertext of the actual response, $\overline{(\mathbb{D}[\mathsf{I}]||\mathsf{I})}$. In this Phase 8 (Details in Figure 3.15), our protocol delivers the corresponding plaintext response to the client and appends the masked version of that response to both replicas of the shelter. It is crucial to note that a different mask is used every time when appending a new response in the shelter, and it essentially works as a one-time pad encryption. Thus, $\mathsf{S}_{\mathsf{i}\in\{\alpha,\gamma\}}$ learn nothing from the shelter content, not even whether multiple replicas of the same response are present in the shelter or not.

Primarily, we use the oblivious decryption (discussed earlier in Section 3.2.6) for this purpose. $\mathbb{C}$ and S_γ each select a random mask individually. S_γ homomorphically combines those masks with the response ciphertext $(\mathbb{D}[\mathsf{I}]||\mathsf{I})$ and then sends those resulting ciphertexts to S_β for a decryption (Step 3). S_β decrypts $((\mathbb{D}[\mathsf{I}]||\mathsf{I}) \oplus \mathsf{m}_\gamma)$. Before returning the plaintext to S_γ, it applies the ith mask of $\mathcal{M}[]$ to it. Subsequently, $\mathsf{S}_{\mathsf{i}\in\{\alpha,\gamma\}}$ appends that masked plaintext response to its shelter replica (Step 6-8). On the otherhand, S_β returns $((\mathbb{D}[\mathsf{I}]||\mathsf{I}) \oplus \mathsf{m}_\mathsf{C})$ to $\mathbb{C}$. The client unmasks that using m_C and obtains the desired plaintext response.

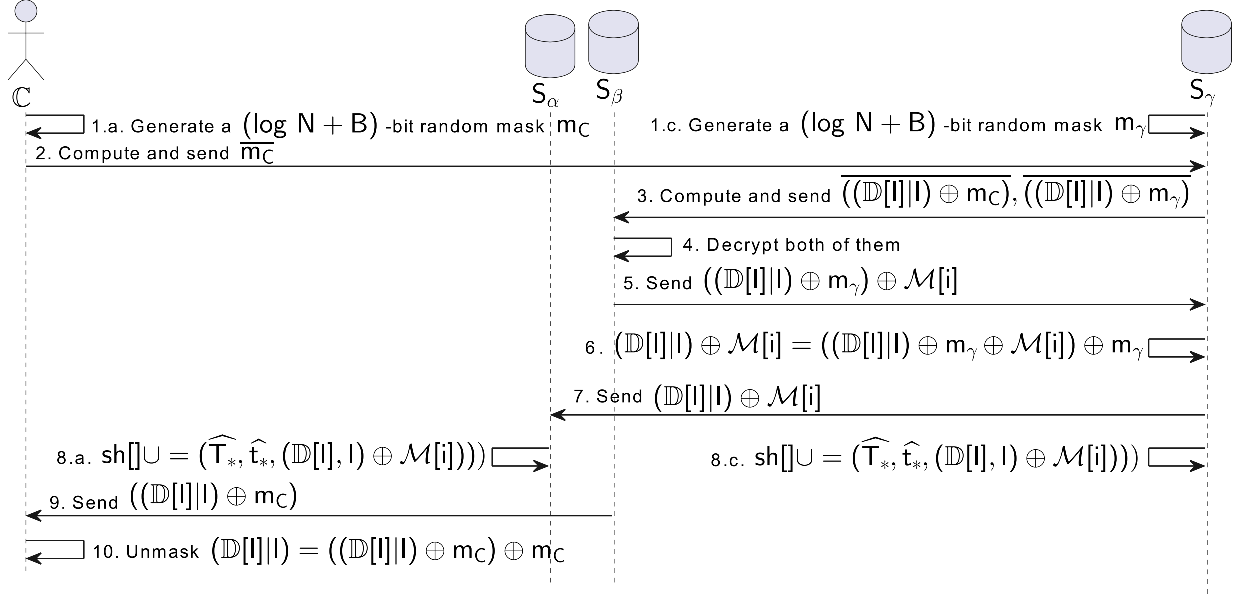


Figure 3.15: Returning Plaintext Response

3.4.12 Privacy Analysis

As mentioned earlier, even if the servers do not collude with one another, they are individually honest-but-curious. We employ a simulation-based proof technique [76, 77] to demonstrate that our protocol leaks no information to any server about the client's request or response. Yet, servers have the capability to generate fake clients to obtain adversarial advantages. Therefore, we also demonstrate that generating a fake client does not provide any additional information to the server, which guarantees active security [55].

Since, the initialization and per-epoch operations do not depend on the client's requests, these steps cannot reveal anything about the client's access patterns. Only the execution trace of our protocol, while processing each PIR request, is influenced by the client's request and response. Therefore, we will focus our analysis on the privacy of that execution trace.

Processing each PIR request can be considered as a functionality, $\mathcal{F}^{\mathsf{PIR}}$, which involves six parties: $\mathbb{C}, S_\alpha, S_\beta, S_\gamma, S_\delta$ and S_ϵ. $\mathcal{F}^{\mathsf{PIR}}$ receives input(s) from each party and produces output(s) for some of them. Notably, the output(s) for each party is influenced by the "view" of the party. View of a party while executing a protocol consists of its own input, internal random tape and messages obtained from the other parties [77]. Hence,

$\forall_{p \in \{\mathbb{C}, S_\alpha, S_\beta, S_\gamma, S_\delta, S_\epsilon\}} \mathsf{View}_p(\mathsf{Input}) = \{\mathsf{Input}_p; R_p; \mathsf{Msgs}_p\}$. Where R_p represents the party p's internal random tape.

By inspecting the detailed flow of our protocol (Figure 3.10- 3.15), the inputs, outputs, and observed incoming messages for all the parties can be determined, and Table 3.5 summarizes them. For simplicity, we have not shown the public inputs in this table (e.g., the encryption schemes' public keys). It is implicit that all the parties have access to them. By the end, our protocol outputs the requested data block, concatenated with the requested index, to the client.

Table 3.5: The inputs and outputs of all parties while processing the ith PIR-request

Party	**Input**	**Output**	**Incoming messages**
S_α	$\mathsf{Input}_{S\alpha} = \{\ \widetilde{\mathbb{D}}_\alpha[],\ \mathcal{H}_\alpha,\ a,\ sh^{\{i-1\}}[]\}$	$\mathsf{Output}_{S\alpha} = \{\ a',\ sh^i[]\}$	$\mathsf{Msgs}_{S\alpha} = \{\ \mathcal{E}(g^{\rho^I . h_C}),\ \mathcal{E}(g^{\rho^I . h_{\alpha 0}}),\ \mathcal{K}_\alpha,\ \overline{\langle \ddot{d} \rangle_\gamma},\ \overline{\langle fnd \rangle_\gamma},\ \overline{\langle m \rangle_\beta},\ \overline{\langle m \rangle_\delta},\ T_I.h_{\alpha 1}.h_{\beta 0},\ \overline{T_\phi.h_{\alpha 2}.h_{\beta 0}},\ T_*.h_{\alpha 2},\ \widehat{T_*},\ \widehat{t_*},\ \Delta_{abc}.h_{\alpha 3},\ ((\mathbb{D}[I] \Vert I) \oplus \mathcal{M}[i])\}$
S_β	$\mathsf{Input}_{S\beta} = \{\ \mathcal{M}[],\ sk_F,\ sk_{\mathcal{E}},\ sk_{\mathcal{E}q},\ \rho,\ b,\ \pi_\Phi\}$	$\mathsf{Output}_{S\beta} = \{\ b'\}$	$\mathsf{Msgs}_{S\beta} = \{\ \mathcal{E}_q(\rho^I.h_C),\ \mathcal{E}(g^{\rho^I}.a.c),\ \mathcal{E}(T_I.h_{\alpha 1}),\ \overline{T_*.h_{\alpha 2}.h_{\beta 0}},\ \mathcal{E}(T_*.a'.c'.h_{\gamma 0}),\ \mathcal{E}(\Delta_a.\Delta_c.h_{\alpha 3}),\ \overline{((\mathbb{D}[I] \Vert I) \oplus m_C)},\ \overline{((\mathbb{D}[I] \Vert I) \oplus m_\gamma)}\}$
S_γ	$\mathsf{Input}_{S\gamma} = \{\ \widetilde{\mathbb{D}}_\gamma[],\ \mathcal{H}_\gamma,\ c,\ sh^{\{i-1\}}[]\}$	$\mathsf{Output}_{S\gamma} = \{\ c',\ sh^i[]\}$	$\mathsf{Msgs}_{S\gamma} = \{\ \mathcal{E}(g^{\rho^I}.a),\ \mathcal{K}_\gamma,\ T_\phi.h_{\beta 0},\ h_{\alpha 2},\ T_*,\ \overline{\langle SR_{\widetilde{\mathbb{D}}} \rangle_\alpha},\ \overline{fnd},\ \overline{SR_{sh}},\ \mathcal{E}(T_*.a'),\ T_*.a'.b'.c'.h_{\gamma 0},\ h_{\alpha 3},\ \mathcal{E}(\Delta_a.h_{\alpha 3}),\ \Delta_{abc}.h_{\alpha 3},\ \overline{m_C},\ ((\mathbb{D}[I] \Vert I) \oplus m_\gamma \oplus \mathcal{M}[i])\}$
S_δ	$\mathsf{Input}_{S\delta} = \{\mathcal{M}[]\}$	$\mathsf{Output}_{S\delta} = -$	$\mathsf{Msgs}_{S\delta} = \{\forall_{k \in [0,i)} :\ \langle y \rangle_\alpha = \mathsf{Eval}(\mathcal{K}_\alpha, sh[k] \triangleright \widehat{t})\}$
S_ϵ	$\mathsf{Input}_{S\epsilon} = \{\mathcal{M}[]\}$	$\mathsf{Output}_{S\epsilon} = -$	$\mathsf{Msgs}_{S\epsilon} = \{\forall_{k \in [0,i)} :\ \langle y \rangle_\gamma = \mathsf{Eval}(\mathcal{K}_\gamma, sh[k] \triangleright \widehat{t})\}$
$\mathbb{C}$	$\mathsf{Input}_{\mathbb{C}} = \{I\}$	$\mathsf{Output}_{\mathbb{C}} = (\mathbb{D}[I] \Vert I)$	$\mathsf{Msgs}_{\mathbb{C}} = \{\ \mathcal{E}(g^{\rho^I . h_C . h_{\alpha 0}}),\ ((\mathbb{D}[I] \Vert I) \oplus m_C)\}$

By definition, PIR protocol does not reveal anything to the servers. However, after processing each request, the protocol must update the servers' state information to ensure

correct processing of the next request. Hence, those updates are treated as the servers' outputs. It is worth noting that the servers may generate several other intermediate values during protocol execution (e.g., Steps 3, 5, 8, 11, etc., in Figure 3.10). These do not influence their state and are not their outputs.

Simulation-based security requires a probabilistic polynomial-time (PPT) simulator for each party that can generate a view for that party. The generated view must be computationally indistinguishable from the view produced during the protocol's actual execution. The simulator for each party must be able to generate that indistinguishable view, solely using that party's input(s) and output(s). Furthermore, simulation-based security ensures that outputs produced by the party utilizing that simulated view remain consistent with the party's own input(s) and indistinguishable from the output(s) of a real execution of the protocol [77, Definition 4.1].

Thus, our first goal is to design six PPT simulators: $\mathsf{Sim}_{\mathbb{C}}, \mathsf{Sim}_{\mathsf{S}\alpha}, \mathsf{Sim}_{\mathsf{S}\beta}, \mathsf{Sim}_{\mathsf{S}\gamma}, \mathsf{Sim}_{\mathsf{S}\delta}$ and $\mathsf{Sim}_{\mathsf{S}\epsilon}$ and prove the following condition:

$$\forall_{\mathsf{p}\in\{\mathbb{C},\mathsf{S}_\alpha,\mathsf{S}_\beta,\mathsf{S}_\gamma,\mathsf{S}_\delta,\mathsf{S}_\epsilon\}} \left\{\left(\mathsf{Sim}_{\mathsf{p}}(\mathsf{Input}_{\mathsf{p}}, \mathsf{Output}_{\mathsf{p}}(\mathsf{Input})), \mathcal{F}^{\mathsf{PIR}}(\mathsf{Input})\right)\right\} \stackrel{\mathsf{c}}{\equiv} \{(\mathsf{View}_{\mathsf{p}}(\mathsf{Input}), \mathsf{Output}(\mathsf{Input}))\}$$

We use $\stackrel{\mathsf{c}}{\equiv}$ to denote computational indistinguishability, while $\stackrel{\mathsf{s}}{\equiv}$ indicates statistical indistinguishability. Notably, since $\stackrel{\mathsf{s}}{\equiv}$ ensures indistinguishability even against a computationally unbounded adversary, any claim of $\stackrel{\mathsf{s}}{\equiv}$ also implies $\stackrel{\mathsf{c}}{\equiv}$. We will discuss further details regarding our simulator's design and privacy claims in the following text. Following that, our second goal is to show that even if each individual server can launch and control fake clients, none of them will get any adversarial advantages.

Lemma 3.1 (S_α's view is simulatable)**.** *If* $(\mathsf{Gen}, \mathsf{Eval})$ *is a secure DPF scheme and* FHE & $\mathcal{E}$ *are IND-CPA secure encryption schemes, then for the semi-honest (i.e., honest-but-curious)* S_α*, there exists a PPT simulator,* $\mathsf{Sim}_{\mathsf{S}\alpha}$*, such that:*

$$\left\{\left(\mathsf{Sim}_{\mathsf{S}\alpha}(\mathsf{Input}_{\mathsf{S}\alpha}, \mathsf{Output}_{\mathsf{S}\alpha}(\mathsf{Input})), \mathcal{F}^{\mathsf{PIR}}(\mathsf{Input})\right)\right\} \stackrel{\mathsf{c}}{\equiv} \{(\mathsf{View}_{\mathsf{S}\alpha}(\mathsf{Input}), \mathsf{Output}(\mathsf{Input}))\}$$

Proof. Since $(\mathsf{Gen}, \mathsf{Eval})$ is a secure DPF scheme, then there must exist some simulator, $\mathsf{Sim}^{\mathsf{DPF}}$, for DPF keys. In other words, the simulator can produce an evaluation key (without knowing the point function) that is indistinguishable from the real evaluation key. Our designed $\mathsf{Sim}_{\mathsf{S}\alpha}$ utilizes $\mathsf{Sim}^{\mathsf{DPF}}$. The complete design of $\mathsf{Sim}_{\mathsf{S}\alpha}$ is as follows:

1. $\mathsf{Sim}_{\mathsf{S}\alpha}$ chooses a uniform random tape, $\mathsf{R}^{(\mathsf{Sim})}$ for the party.
2. $\mathsf{Sim}_{\mathsf{S}\alpha}$ then utilizes $\mathsf{R}^{(\mathsf{Sim})}$ to choose, $\mathsf{h}_{\alpha 2}^{(\mathsf{Sim})}$ and $\mathsf{a}'^{(\mathsf{Sim})}$ from $\mathbb{G}$.

3. While simulating, a simulator already knows the inputs and outputs of the party and utilizes them to simulate the incoming messages. Therefore, $\mathsf{Sim}_{\mathsf{S}_\alpha}$ extracts $\mathsf{sh}[\mathsf{i}] \triangleright \widehat{\mathsf{T}}, \mathsf{sh}[\mathsf{i}] \triangleright \widehat{\mathsf{t}}$ and $\mathsf{sh}[\mathsf{i}] \triangleright \ddot{\mathsf{d}}$ from the party's output, $\mathsf{sh}^{\mathsf{i}}[]$.

4. $\mathsf{Sim}_{\mathsf{S}_\alpha}$ utilizes $\mathsf{Sim}^{\mathsf{DPF}}$ to generate $\mathcal{K}^{(\mathsf{Sim})}$.

5. $\mathsf{Sim}_{\mathsf{S}_\alpha}$ chooses two random group elements $\mathsf{r}_0, \mathsf{r}_1$ from $\mathbb{G}$ and then produces their El-Gamal Encryptions: $\mathcal{E}(\mathsf{r}_0)$ and $\mathcal{E}(\mathsf{r}_1)$.

6. $\mathsf{Sim}_{\mathsf{S}_\alpha}$ generates three $(\mathsf{m} + \log \mathsf{N})$-bit randoms $\mathsf{r}_2, \mathsf{r}_4, \mathsf{r}_5$, one single bit random r_3 and one p bit random (r_7). Then it produces FHE-ciphertexts of them.

7. $\mathsf{Sim}_{\mathsf{S}_\alpha}$ also chooses two group elements $\mathsf{r}_6, \mathsf{r}_9$ from $\mathbb{G}$, but utilizing a different randomness. This ensures S_α cannot observe any link between these elements and $\mathsf{R}^{(\mathsf{Sim})}$.

8. With that different randomness, $\mathsf{Sim}_{\mathsf{S}_\alpha}$ also chooses $\mathsf{r}_8 \in \mathbb{G}$, but without replacement.

9. Finally, $\mathsf{Sim}_{\mathsf{S}_\alpha}$ outputs the following:

$$\left\{ \left\{ \widetilde{\mathbb{D}}_\alpha[], \mathcal{H}_\alpha, \mathsf{a}, \mathsf{sh}^{\{\mathsf{i}-1\}}[] \right\}; \mathsf{R}^{(\mathsf{Sim})}; \left\{ \mathcal{E}(\mathsf{r}_0), \mathcal{E}(\mathsf{r}_1), \mathcal{K}^{(\mathsf{Sim})}, \overline{\mathsf{r}_2}, \overline{\mathsf{r}_3}, \overline{\mathsf{r}_4}, \overline{\mathsf{r}_5}, \mathsf{r}_6, \overline{\mathsf{r}_7}, \mathsf{r}_8.\mathsf{h}_{\alpha 2}^{(\mathsf{Sim})}, \mathsf{sh}[\mathsf{i}] \triangleright \widehat{\mathsf{T}}, \mathsf{sh}[\mathsf{i}] \triangleright \widehat{\mathsf{t}}, \mathsf{r}_9, \mathsf{sh}[\mathsf{i}] \triangleright \ddot{\mathsf{d}} \right\} \right\}$$

It is to be noted that, $\mathsf{sh}[\mathsf{i}] \triangleright \widehat{\mathsf{T}}, \mathsf{sh}[\mathsf{i}] \triangleright \widehat{\mathsf{t}}$ and $\mathsf{sh}[\mathsf{i}] \triangleright \ddot{\mathsf{d}}$ match exactly with the actual incoming messages, $\widehat{\mathsf{T}_*}, \widehat{\mathsf{t}_*}$ and $((\mathbb{D}[\mathsf{I}]|\mathsf{I}) \oplus \mathcal{M}[\mathsf{i}])$, respectively. Due to the IND-CPA security of the El-Gamal encryption scheme, $\mathcal{E}(\mathsf{r}_0), \mathcal{E}(\mathsf{r}_1)$ are indistinguishable from $\mathcal{E}(\mathsf{g}^{\rho^{\mathsf{I}}.\mathsf{h}_{\mathsf{C}}}), \mathcal{E}(\mathsf{g}^{\rho^{\mathsf{I}}.\mathsf{h}_{\alpha 0}})$. Similarly, due to the IND-CPA security of FHE, $\overline{\mathsf{r}_2}, \overline{\mathsf{r}_3}, \overline{\mathsf{r}_4}, \overline{\mathsf{r}_5}$ and $\overline{\mathsf{r}_7}$ are indistinguishable from $\overline{\langle \ddot{\mathsf{d}} \rangle_\gamma}$, $\overline{\langle \mathsf{fnd} \rangle_\gamma}$, $\overline{\langle \mathsf{m} \rangle_\beta}$, $\overline{\langle \mathsf{m} \rangle_\delta}$, and $\overline{\mathsf{T}_\phi.\mathsf{h}_{\alpha 2}.\mathsf{h}_{\beta 0}}$, respectively.

The security of the DPF scheme, ensures $\mathcal{K}^{(\mathsf{Sim})}$ and $\mathcal{K}_\alpha$ are indistinguishable. Since in real protocol execution, the incoming messages, $\mathsf{T}_{\mathsf{I}}.\mathsf{h}_{\alpha 1}.\mathsf{h}_{\beta 0}$ and $\Delta_{\mathsf{abc}}.\mathsf{h}_{\alpha 3}$ are formed by multiplying group elements, chosen randomly by the other parties, they are indistinguishable from the independently chosen random group elements r_6 and r_9. Additionally, since r_8 is a random choice without replacement, it remains indistinguishable from T_*.

Furthermore, it can be verified that, by utilizing these simulated messages with the honest protocol execution, S_α's output becomes $\{\mathsf{a}'^{(\mathsf{Sim})}, \mathsf{sh}^{\mathsf{i}}[]\}$. In other words, the first

component remains indistinguishable from the real output and consistent with $\mathsf{R}^{(\mathsf{Sim})}$, and the second component of the output matches exactly. Thus, we can conclude that for the designed PPT simulator,

$$\left\{\left(\mathsf{Sim}_{\mathsf{S}\alpha}(\mathsf{Input}_{\mathsf{S}\alpha}, \mathsf{Output}_{\mathsf{S}\alpha}(\mathsf{Input})), \mathcal{F}^{\mathsf{PIR}}(\mathsf{Input})\right)\right\} \stackrel{\mathsf{c}}{\equiv} \{(\mathsf{View}_{\mathsf{S}\alpha}(\mathsf{Input}), \mathsf{Output}(\mathsf{Input}))\}$$

□

Lemma 3.2 (S_β's view is simulatable)**.** *For the semi-honest* S_β*, there exists a PPT simulator,* $\mathsf{Sim}_{\mathsf{S}\beta}$*, such that:*

$$\left\{\left(\mathsf{Sim}_{\mathsf{S}\beta}(\mathsf{Input}_{\mathsf{S}\beta}, \mathsf{Output}_{\mathsf{S}\beta}(\mathsf{Input})), \mathcal{F}^{\mathsf{PIR}}(\mathsf{Input})\right)\right\} \stackrel{\mathsf{s}}{\equiv} \{(\mathsf{View}_{\mathsf{S}\beta}(\mathsf{Input}), \mathsf{Output}(\mathsf{Input}))\}$$

Proof. All the components of Msgs_β are either El-Gamal or FHE-ciphertexts. However, since S_β already knows the decryption keys, it can actually learn the corresponding plaintexts. Hence, the simulator's task in this case would be to ensure that the plaintext corresponding to the incoming ciphertexts remains indistinguishable. Therefore:

1. $\mathsf{Sim}_{\mathsf{S}\beta}$ chooses a uniform random tape, $\mathsf{R}^{(\mathsf{Sim})}$.
2. $\mathsf{Sim}_{\mathsf{S}\beta}$ then utilizes $\mathsf{R}^{(\mathsf{Sim})}$ to choose, $\mathsf{b}'^{(\mathsf{Sim})}$ from $\mathbb{G}$.
3. $\mathsf{Sim}_{\mathsf{S}\beta}$ chooses a couple of random values, all by using a randomness other than $\mathsf{R}^{(\mathsf{Sim})}$.
   (a) It chooses r_0 from $\mathbb{Z}_\mathsf{q}^*$.
   (b) It chooses $\mathsf{r}_1, \mathsf{r}_2, \mathsf{r}_3, \mathsf{r}_4$ and r_5 from $\mathbb{G}$.
   (c) It also chooses two $(\mathsf{m} + \log \mathsf{N})$-bit randoms r_6 and r_7.
4. Finally, $\mathsf{Sim}_{\mathsf{S}\beta}$ outputs the following:

$$\left\{ \left\{\mathcal{M}[], \mathsf{sk}_\mathsf{F}, \mathsf{sk}_\mathcal{E}, \mathsf{sk}_{\mathcal{E}_\mathsf{q}}, \rho, \mathsf{b}, \pi_\Phi\right\}; \quad \mathsf{R}^{(\mathsf{Sim})}; \quad \left\{\mathcal{E}_\mathsf{q}(\mathsf{r}_0), \mathcal{E}(\mathsf{r}_1), \mathcal{E}(\mathsf{r}_2), \overline{\mathsf{r}_3}, \mathcal{E}(\mathsf{r}_4), \mathcal{E}(\mathsf{r}_5), \overline{\mathsf{r}_6}, \overline{\mathsf{r}_7}\right\} \right\}$$

The plaintexts corresponding to all the incoming messages are formed by utilising random(s) chosen independently by others (e.g., $\rho^{\mathsf{I}}.\mathsf{h_C}$ is formed by using the random $\mathsf{h_C}$, chosen by $\mathbb{C}$). Hence, plaintexts of all the real incoming messages remain statistically indistinguishable from the randoms chosen by Sim_β (i.e., $\mathsf{r}_0, \mathsf{r}_1..\mathsf{r}_7$). Moreover, $\mathsf{b}'^{(\mathsf{Sim})}$ and $\mathsf{R}^{(\mathsf{Sim})}$ are also consistent with each other. Thus, for the designed PPT simulator,

$$\left\{\left(\mathsf{Sim}_{\mathsf{S}\beta}(\mathsf{Input}_{\mathsf{S}\beta}, \mathsf{Output}_{\mathsf{S}\beta}(\mathsf{Input})), \mathcal{F}^{\mathsf{PIR}}(\mathsf{Input})\right)\right\} \stackrel{\mathsf{s}}{\equiv} \{(\mathsf{View}_{\mathsf{S}\beta}(\mathsf{Input}), \mathsf{Output}(\mathsf{Input}))\}$$

□

Lemma 3.3 (S_γ's view is simulatable)**.** *If* $(\mathsf{Gen}, \mathsf{Eval})$ *is a secure DPF scheme and* FHE & $\mathcal{E}$ *are IND-CPA secure encryption schemes, then for the semi-honest* S_γ*, there exists a PPT simulator,* $\mathsf{Sim}_{\mathsf{S}\gamma}$*, such that:*

$$\left\{\left(\mathsf{Sim}_{\mathsf{S}\gamma}(\mathsf{Input}_{\mathsf{S}\gamma}, \mathsf{Output}_{\mathsf{S}\gamma}(\mathsf{Input})), \mathcal{F}^{\mathsf{PIR}}(\mathsf{Input})\right)\right\} \stackrel{\mathsf{c}}{\equiv} \{(\mathsf{View}_{\mathsf{S}\gamma}(\mathsf{Input}), \mathsf{Output}(\mathsf{Input}))\}$$

Proof. Our designed $\mathsf{Sim}_{\mathsf{S}\gamma}$ utilizes $\mathsf{Sim}^{\mathsf{DPF}}$. The complete design of $\mathsf{Sim}_{\mathsf{S}\gamma}$ is as follows:

1. $\mathsf{Sim}_{\mathsf{S}\gamma}$ chooses a uniform random tape, $\mathsf{R}^{(\mathsf{Sim})}$.
2. Then utilizes $\mathsf{R}^{(\mathsf{Sim})}$ to choose, $\mathsf{c}'^{(\mathsf{Sim})}$ from $\mathbb{G}$ and $\mathsf{m}_\gamma^{(\mathsf{Sim})}$ of size $(\mathsf{B} + \log \mathsf{N})$-bits.
3. From S_γ's output, $\mathsf{Sim}_{\mathsf{S}\gamma}$ extracts $\mathsf{sh}[\mathsf{i}] \triangleright \ddot{\mathsf{d}}$.
4. $\mathsf{Sim}_{\mathsf{S}\gamma}$ utilizes $\mathsf{Sim}^{\mathsf{DPF}}$ to generate $\mathcal{K}^{(\mathsf{Sim})}$.
5. $\mathsf{Sim}_{\mathsf{S}\gamma}$ then chooses a few more random values using a different randomness.
   (a) It chooses $\mathsf{r}_0, \mathsf{r}_1, \mathsf{r}_2, \mathsf{r}_7, \mathsf{r}_8, \mathsf{r}_9, \mathsf{r}_{10},$ and r_{11} from $\mathbb{G}$.
   (b) It also chooses r_3 from $\mathbb{G}$, but without replacement.
   (c) It chooses three $(\mathsf{m} + \log \mathsf{N})$-bit, $\mathsf{r}_4, \mathsf{r}_6$ and r_{12}.
   (d) Finally, it chooses one single-bit value, r_5.
6. Finally, $\mathsf{Sim}_{\mathsf{S}\gamma}$ outputs the following:

$$\left\{ \left\{\widetilde{\mathbb{D}}_\gamma[], \mathcal{H}_\gamma, \mathsf{c}, \mathsf{sh}^{\{\mathsf{i}-1\}}[]\right\}; \quad \mathsf{R}^{(\mathsf{Sim})}; \quad \left\{\mathcal{E}(\mathsf{r}_0), \mathcal{K}^{(\mathsf{Sim})}, \mathsf{r}_1, \mathsf{r}_2, \mathsf{r}_3, \overline{\mathsf{r}_4}, \overline{\mathsf{r}_5}, \overline{\mathsf{r}_6}, \mathcal{E}(\mathsf{r}_7), \mathsf{r}_8, \mathsf{r}_9, \mathcal{E}(\mathsf{r}_{10}), \mathsf{r}_{11}, \overline{\mathsf{r}_{12}}, (\mathsf{sh}[\mathsf{i}] \triangleright \ddot{\mathsf{d}} \oplus \mathsf{m}_\gamma^{(\mathsf{Sim})})\right\} \right\}$$

Among the simulated messages, due to IND-CPA security, the El-Gamal and FHE ciphertext parts remain indistinguishable. The security of the DPF scheme ensures that $\mathcal{K}^{(\mathsf{Sim})}$ is indistinguishable from $\mathcal{K}_\alpha$. Randomly chosen $(\mathsf{r}_2, \mathsf{r}_9)$ are indistinguishable from the randomly chosen $(\mathsf{h}_{\alpha 2}, \mathsf{h}_{\alpha 3})$. Moreover, r_3 is a random choice without replacement. Therefore, it also remains indistinguishable from T_*.

Since $\mathsf{T}_\phi.\mathsf{h}_{\beta 0}, \mathsf{T}_*.\mathsf{a}'.\mathsf{b}'.\mathsf{c}'.\mathsf{h}_{\gamma 0}$ and $\Delta_{\mathsf{abc}}.\mathsf{h}_{\alpha 3}$ are formed by multiplying group elements, chosen randomly by the other parties, they are indistinguishable from the independently simulated group elements $\mathsf{r}_1, \mathsf{r}_8$ and r_{11}. $(\mathsf{sh}[\mathsf{i}] \triangleright \ddot{\mathsf{d}} \oplus \mathsf{m}_\gamma^{(\mathsf{Sim})})$ is also indistinguishable from the corresponding real message (i.e., $\mathsf{sh}[\mathsf{i}] \triangleright \ddot{\mathsf{d}} \oplus \mathsf{m}_\gamma = ((\mathbb{D}[\mathsf{I}]||\mathsf{I}) \oplus \mathcal{M}[\mathsf{i}] \oplus \mathsf{m}_\gamma))$. Furthermore, by utilizing these simulated messages with the honest protocol execution, S_γ's output becomes $\{\mathsf{c}'^{(\mathsf{Sim})}, \mathsf{sh}^{\mathsf{i}}[]\}$. In other words, the first component remains indistinguishable from the real output and consistent with $\mathsf{R}^{(\mathsf{Sim})}$ and the second component of the output matches exactly. Thus, we can conclude that for the designed PPT simulator,

$$\left\{\left(\mathsf{Sim}_{\mathsf{S}\gamma}(\mathsf{Input}_{\mathsf{S}\gamma}, \mathsf{Output}_{\mathsf{S}\gamma}(\mathsf{Input})), \mathcal{F}^{\mathsf{PIR}}(\mathsf{Input})\right)\right\} \stackrel{\mathsf{c}}{\equiv} \{(\mathsf{View}_{\mathsf{S}\gamma}(\mathsf{Input}), \mathsf{Output}(\mathsf{Input}))\}$$

□

Lemma 3.4 (S_δ's view is simulatable)**.** *If* $(\mathsf{Gen}, \mathsf{Eval})$ *is a secure DPF scheme, then for the semi-honest* S_δ*, there exists a PPT simulator,* $\mathsf{Sim}_{\mathsf{S}\delta}$*, such that:*

$$\left\{\left(\mathsf{Sim}_{\mathsf{S}\delta}(\mathsf{Input}_{\mathsf{S}\delta}, \mathsf{Output}_{\mathsf{S}\delta}(\mathsf{Input})), \mathcal{F}^{\mathsf{PIR}}(\mathsf{Input})\right)\right\} \stackrel{\mathsf{c}}{\equiv} \{(\mathsf{View}_{\mathsf{S}\delta}(\mathsf{Input}), \mathsf{Output}(\mathsf{Input}))\}$$

Proof. S_δ does not require any randomness while executing its steps of the protocol. Hence, $\mathsf{Sim}_{\mathsf{S}\delta}$ does not generate any random tape for the party. During processing the ith PIR request, S_δ just receives $(\mathsf{i}-1)$ DPF evaluation results. Following is the design of $\mathsf{Sim}_{\mathsf{S}\delta}$:

1. $\mathsf{Sim}_{\mathsf{S}\delta}$ chooses $(\mathsf{i}-1)$-random elements, $\forall_{\mathsf{k}\in[0,\mathsf{i})} : \mathsf{r}_\mathsf{k}$, from the range of $\mathsf{Eval}()$-algorithm.
2. Then, $\mathsf{Sim}_{\mathsf{S}\delta}$ outputs the following:

$$\left\{ \left\{\mathcal{M}[]\right\}; \quad -; \quad \left\{\forall_{\mathsf{k}\in[0,\mathsf{i})} : \mathsf{r}_\mathsf{k}\right\} \right\}$$

Any secure DPF scheme uses pseudorandomness while performing the $\mathsf{Eval}()$ algorithm. The security of the DPF scheme ensures that the output distribution of the $\mathsf{Eval}()$ algorithm remains indistinguishable from a random permutation of the range of the $\mathsf{Eval}()$. Hence, $\forall_{k \in [0,i)} : r_k$ remains computationally indistinguishable from $\forall_{k \in [0,i)} : \langle y \rangle_\alpha = \mathsf{Eval}(\mathcal{K}_\alpha, \mathsf{sh}[k] \triangleright \widehat{t})$. Moreover, S_δ does not produce any output. Thus, we can conclude that,

$$\left\{ \left(\mathsf{Sim}_{\mathsf{S}\delta}(\mathsf{Input}_{\mathsf{S}\delta}, \mathsf{Output}_{\mathsf{S}\delta}(\mathsf{Input})), \mathcal{F}^{\mathsf{PIR}}(\mathsf{Input}) \right) \right\} \stackrel{c}{\equiv} \{(\mathsf{View}_{\mathsf{S}\delta}(\mathsf{Input}), \mathsf{Output}(\mathsf{Input}))\}$$

□

Lemma 3.5 (S_ϵ's view is simulatable)**.** *If* $(\mathsf{Gen}, \mathsf{Eval})$ *is a secure DPF scheme, then for the semi-honest* S_ϵ*, there exists a PPT simulator,* $\mathsf{Sim}_{\mathsf{S}\epsilon}$*, such that:*

$$\left\{ \left(\mathsf{Sim}_{\mathsf{S}\epsilon}(\mathsf{Input}_{\mathsf{S}\epsilon}, \mathsf{Output}_{\mathsf{S}\epsilon}(\mathsf{Input})), \mathcal{F}^{\mathsf{PIR}}(\mathsf{Input}) \right) \right\} \stackrel{c}{\equiv} \{(\mathsf{View}_{\mathsf{S}\epsilon}(\mathsf{Input}), \mathsf{Output}(\mathsf{Input}))\}$$

Proof. The design of $\mathsf{Sim}_{\mathsf{S}\epsilon}$ is same as the design of $\mathsf{Sim}_{\mathsf{S}\delta}$ and the proof remains similar to Lemma 3.4. □

Lemma 3.6 ($\mathbb{C}$'s view is simulatable)**.** *If* $\mathcal{E}$ *is IND-CPA secure, then for the semi-honest* $\mathbb{C}$*, there exists a PPT simulator,* $\mathsf{Sim}_{\mathbb{C}}$*, such that:*

$$\left\{ \left(\mathsf{Sim}_{\mathbb{C}}(\mathsf{Input}_{\mathbb{C}}, \mathsf{Output}_{\mathbb{C}}(\mathsf{Input})), \mathcal{F}^{\mathsf{PIR}}(\mathsf{Input}) \right) \right\} \stackrel{c}{\equiv} \{(\mathsf{View}_{\mathbb{C}}(\mathsf{Input}), \mathsf{Output}(\mathsf{Input}))\}$$

Proof. The design of $\mathsf{Sim}_{\mathbb{C}}$ is as follows:

1. $\mathsf{Sim}_{\mathbb{C}}$ chooses a uniform random tape, $\mathsf{R}^{(\mathsf{Sim})}$ for $\mathbb{C}$ and utilizes that to produce a mask, $\mathsf{m}_{\mathsf{C}}^{(\mathsf{Sim})}$, of size $(\mathsf{B} + \log \mathsf{N})$-bits.
2. $\mathsf{Sim}_{\mathbb{C}}$ also chooses a random $\mathsf{r}_0 \in \mathbb{G}$ (the used randomness does not matter).
3. Finally, $\mathsf{Sim}_{\mathbb{C}}$ outputs the following:

$$\left\{ \begin{array}{r} \left\{ \mathsf{I} \right\}; \\ \mathsf{R}^{(\mathsf{Sim})}; \\ \left\{ \mathcal{E}(\mathsf{r}_0), ((\mathbb{D}[\mathsf{I}] || \mathsf{I}) \oplus \mathsf{m}_{\mathsf{C}}^{(\mathsf{Sim})}) \right\} \end{array} \right\}$$

Since El-Gamal encryption scheme is IND-CPA secure, $\mathcal{E}(\mathsf{r}_0)$ and $\mathcal{E}(\mathsf{g}^{\rho^{\mathsf{l}}.\mathsf{h}_{\mathsf{c}}.\mathsf{h}_{\alpha 0}})$ remain indistinguishable. The simulator generates the mask in the same way as the real execution. Hence, this simulation allows $\mathbb{C}$ to obtain the expected output and the output remains consistent with $\mathbb{C}$'s own view. Thus, we can conclude that for the designed PPT simulator,

$$\left\{\left(\mathsf{Sim}_{\mathbb{C}}(\mathsf{Input}_{\mathbb{C}}, \mathsf{Output}_{\mathbb{C}}(\mathsf{Input})), \mathcal{F}^{\mathsf{PIR}}(\mathsf{Input})\right)\right\} \stackrel{c}{\equiv} \{(\mathsf{View}_{\mathbb{C}}(\mathsf{Input}), \mathsf{Output}(\mathsf{Input}))\}$$

□

Corollary 3.1. *If* (Gen, Eval) *is a secure DPF scheme and* FHE & $\mathcal{E}$ *are IND-CPA secure encryption schemes, then our protocol leaks nothing except the intended output(s) to individual parties.*

Now we proceed towards our next goal. Any of the servers can launch fake clients and control any number of queries, independently, attempting to determine access patterns of a targeted PIR client. Therefore, active security [15, 55] is required. Hence, we show that even if the client is a fake client, launched by one of the servers, our protocol leaks no information to the adversary.

When launching a fake client, the server will have access to the client's input and output, as well as the messages it receives. Hence, the aim is to show that, even then, the server can learn nothing extra. As a result, launching a fake client does not help break the access pattern privacy of legitimate clients. Notice that, since the servers do not collude, each fake client can be controlled by only one server.

Thus, now we design five PPT simulators: $\mathsf{Sim}_{(\mathbb{C},\mathsf{S}\alpha)}, \mathsf{Sim}_{(\mathbb{C},\mathsf{S}\beta)}, \mathsf{Sim}_{(\mathbb{C},\mathsf{S}\gamma)}, \mathsf{Sim}_{(\mathbb{C},\mathsf{S}\delta)}$ and $\mathsf{Sim}_{(\mathbb{C},\mathsf{S}\epsilon)}$ and prove the following condition:

$$\forall_{\mathsf{p}\in\{\mathbb{C}\}\times\{\mathsf{S}_\alpha,\mathsf{S}_\beta,\mathsf{S}_\gamma,\mathsf{S}_\delta,\mathsf{S}_\epsilon\}} \left\{\left(\mathsf{Sim}_{\mathsf{p}}(\mathsf{Input}_{\mathsf{p}}, \mathsf{Output}_{\mathsf{p}}(\mathsf{Input})), \mathcal{F}^{\mathsf{PIR}}(\mathsf{Input})\right)\right\} \stackrel{c}{\equiv} \{(\mathsf{View}_{\mathsf{p}}(\mathsf{Input}), \mathsf{Output}(\mathsf{Input}))\}$$

We start with the design of $\mathsf{Sim}_{(\mathbb{C},\mathsf{S}\beta)}$. Since S_β holds all the decryption keys, $\mathsf{Sim}_{(\mathbb{C},\mathsf{S}\beta)}$'s task is not only to generate indistinguishable ciphertext messages, but also to ensure that their underlying plaintexts remain consistent with the input and output. Therefore:

1. $\mathsf{Sim}_{(\mathbb{C},\mathsf{S}\beta)}$ chooses a uniform random tape, $\mathsf{R}^{(\mathsf{Sim})}$.
2. $\mathsf{Sim}_{(\mathbb{C},\mathsf{S}\beta)}$ then utilizes $\mathsf{R}^{(\mathsf{Sim})}$ to choose:
   (a) $\mathsf{b}'^{(\mathsf{Sim})}$ from $\mathbb{G}$.
   (b) $\mathsf{h}_{\mathsf{C}}^{(\mathsf{Sim})}$ from $\mathbb{Z}_{\mathsf{q}}^*$.

(c) $(\mathsf{m}+\log\mathsf{N})$-bit random $\mathsf{m}_{\mathsf{C}}^{(\mathsf{Sim})}$.

3. $\mathsf{Sim}_{(\mathbb{C},\mathsf{S}\beta)}$ then chooses a couple of other values, using a different randomness.
   (a) It chooses $\mathsf{r}_2,\mathsf{r}_3,\mathsf{r}_4,\mathsf{r}_5$ and r_6 from $\mathbb{G}$.
   (b) It also chooses one $(\mathsf{m}+\log\mathsf{N})$-bit random r_6.
4. Finally, $\mathsf{Sim}_{(\mathbb{C},\mathsf{S}\beta)}$ outputs the following:

$$\Big\{\Big\{\mathsf{Input}_{\mathbb{C}\cup}\mathsf{Input}_{\mathsf{S}\beta}\Big\}=\Big\{\mathsf{I},\mathcal{M}[],\mathsf{sk}_{\mathsf{F}},\mathsf{sk}_{\mathcal{E}},\mathsf{sk}_{\mathcal{E}_{\mathsf{q}}},\rho,\mathsf{b},\pi_{\Phi}\Big\};$$

$$\mathsf{R}^{(\mathsf{Sim})};$$

$$\Big\{\mathsf{Msgs}_{\mathbb{C}}\cup\mathsf{Msgs}_{\mathsf{S}\beta}\Big\}=\Big\{\mathcal{E}(\mathsf{r}_0),((\mathbb{D}[\mathsf{I}]||\mathsf{I})\oplus\mathsf{m}_{\mathsf{C}}^{(\mathsf{Sim})}),\mathcal{E}_{\mathsf{q}}(\rho^{\mathsf{I}}.\mathsf{h}_{\mathsf{C}}^{(\mathsf{Sim})}),\mathcal{E}(\mathsf{r}_1),\mathcal{E}(\mathsf{r}_2),\overline{\mathsf{r}_3},\mathcal{E}(\mathsf{r}_4),\mathcal{E}(\mathsf{r}_5),\overline{(\mathbb{D}[\mathsf{I}]||\mathsf{I})\oplus\mathsf{m}_{\mathsf{C}}^{(\mathsf{Sim})}},\overline{\mathsf{r}_6}\Big\}\Big\}$$

First, we analyze the plaintext corresponding to the first component of the incoming messages, $\mathsf{g}^{\rho^{\mathsf{I}}.\mathsf{h}_{\mathsf{C}}.\mathsf{h}_{\alpha 0}}$. Since, now S_β controls $\mathbb{C}$ it already knows ρ,I and h_{C}. However, $\mathsf{h}_{\alpha 0}$ is randomly chosen by S_α. Hence, r_0 remains computationally indistinguishable from $\mathsf{g}^{\rho^{\mathsf{I}}.\mathsf{h}_{\mathsf{C}}.\mathsf{h}_{\alpha 0}}$, due to the known hardness of the DDH Problem.

Except $\mathcal{E}_{\mathsf{q}}(\rho^{\mathsf{I}}.\mathsf{h}_{\mathsf{C}})$ and $\overline{(\mathbb{D}[\mathsf{I}]||\mathsf{I})\oplus\mathsf{m}_{\mathsf{C}}}$, plaintexts corresponding to the all other ciphertexts are formed by utilising random(s) chosen by a different server (e.g., $\mathsf{g}^{\rho^{\mathsf{I}}}.\mathsf{a}.\mathsf{c}$ is formed by using the random elements a,c, chosen by S_α & S_γ respectively). Hence, plaintexts of all those incoming messages remain statistically indistinguishable from the randoms chosen by $\mathsf{Sim}_{(\mathbb{C},\beta)}$ (i.e., $\mathsf{r}_1,\mathsf{r}_2..\mathsf{r}_6$).

On the other hand, the messages $\mathcal{E}_{\mathsf{q}}(\rho^{\mathsf{I}}.\mathsf{h}_{\mathsf{C}}^{(\mathsf{Sim})})$ and $\overline{(\mathbb{D}[\mathsf{I}]||\mathsf{I})\oplus\mathsf{m}_{\mathsf{C}}^{(\mathsf{Sim})}}$ and the output $\mathsf{b}'^{(\mathsf{Sim})}$ remains consistent with the randomness of the party, $\mathsf{R}^{(\mathsf{Sim})}$. They are also indistinguishable from the real execution of the protocol. Thus, we can conclude that for the designed PPT simulator,

$$\big\{\big(\mathsf{Sim}_{\mathsf{S}\beta}(\mathsf{Input}_{\mathsf{S}\beta},\mathsf{Output}_{\mathsf{S}\beta}(\mathsf{Input})),\mathcal{F}^{\mathsf{PIR}}(\mathsf{Input})\big)\big\}\stackrel{s}{\equiv}\{(\mathsf{View}_{\mathsf{S}\beta}(\mathsf{Input}),\mathsf{Output}(\mathsf{Input}))\}$$

Apart from S_β, no other server can decrypt the incoming messages. Hence, launching a fake client by them will give them only the combined view. Therefore, the design of all other simulators is straightforward. Specifically, $\forall_{\mathsf{p}\in\{\mathbb{C}\}\times\{\mathsf{S}_\alpha,\mathsf{S}_\gamma,\mathsf{S}_\delta,\mathsf{S}_\epsilon\}}\mathsf{Sim}_{\mathsf{p}}$ utilizes $\forall_{\mathsf{s}\in\{\mathsf{S}_\alpha,\mathsf{S}_\gamma,\mathsf{S}_\delta,\mathsf{S}_\epsilon\}}\mathsf{Sim}_{\mathsf{s}}$ and $\mathsf{Sim}_{\mathbb{C}}$ together and returns the combined output of them.

Therefore, for all the designed simulators, we can conclude:

$$\forall_{\mathsf{p}\in\{\mathbb{C}\}\times\{\mathsf{S}_\alpha,\mathsf{S}_\beta,\mathsf{S}_\gamma,\mathsf{S}_\delta,\mathsf{S}_\epsilon\}}\big\{\big(\mathsf{Sim}_{\mathsf{p}}(\mathsf{Input}_{\mathsf{p}},\mathsf{Output}_{\mathsf{p}}(\mathsf{Input})),\mathcal{F}^{\mathsf{PIR}}(\mathsf{Input})\big)\big\}\stackrel{c}{\equiv}\{(\mathsf{View}_{\mathsf{p}}(\mathsf{Input}),\mathsf{Output}(\mathsf{Input}))\}$$

Corollary 3.2. *If* $(\mathsf{Gen}, \mathsf{Eval})$ *is a secure DPF scheme and* FHE & $\mathcal{E}$ *are IND-CPA secure encryption schemes, then our protocol leaks nothing to the participating servers, even if each of them individually launches and controls fake clients.*

3.4.13 Overhead Analysis

We will begin by conducting an asymptotic analysis of this scheme, specifically examining its differences with *Basic Scheme*. More importantly, we will evaluate the performance parameters of this scheme, in practical settings. To achieve this, we have implemented the scheme and carried out an empirical analysis of the associated overhead.

Asymptotic Analysis:

This scheme uses two new servers, S_δ & S_ϵ, each storing $\mathcal{M}[]$, which has $\sqrt{\mathsf{N}}$ elements. S_δ & S_ϵ also process $O(\sqrt{\mathsf{N}})$ elements of $\mathcal{M}[]$ during each request. Hence, both the computation and storage requirements of S_δ & S_ϵ are $O(\sqrt{\mathsf{N}})$. Unlike our first scheme, this scheme requires both S_α and S_γ to maintain and search the shelter. As a result, compared to our *Basic Scheme*, the computational overhead for S_α increased from $O(1)$ to $O(\sqrt{\mathsf{N}})$ when processing each request. Similarly, the storage requirement for S_α also rose from $O(1)$ to $O(\sqrt{\mathsf{N}})$.

Moreover, during the per-epoch operations, this scheme stores the secret shares of the shuffled database in both S_α and S_γ. Consequently, the amortized communication and computation for S_γ increased from $O(1)$ to $O(\sqrt{\mathsf{N}})$. The remaining details regarding the asymptotic overheads of this scheme are the same as those shown in Table 3.2. This scheme still achieves the same overall overhead of $O(\sqrt{\mathsf{N}})$.

Importantly, this scheme significantly reduces the number of costly homomorphic operations required for processing each request, decreasing it from $O(\sqrt{\mathsf{N}})$ to a fixed number. A detailed examination of the sequence diagrams (Figure 3.10- 3.15) shows that each request involves twelve FHE encryptions, three FHE decryptions, and nine FHE evaluations. It is important to note that none of the homomorphic evaluations are prohibitively expensive in our context, as we only need to evaluate homomorphic circuits with a maximum multiplicative depth of two. Consequently, we can effectively employ leveled FHE in our scheme, which enhances overall performance. Furthermore, this scheme does not store FHE ciphertexts within the shuffled databases or within the shelter. As a result, we not only reduce the effective server storage requirements but also achieve significantly faster per-epoch operations.

Empirical Analysis:

We developed an initial prototype of our scheme [78]. In our implementation, we utilized the GMP library [79] for large number arithmetic. For DPF operations, we modified and repurposed the library created by Wang et al. [80]. Our protocol employs BGV [81] as the underlying FHE scheme, leveraging the available implementation from the OpenFHE library [58]. Additionally, for cuckoo hashing, we used the Kuku library [82].

Our PIR scheme is a computational PIR system, which we configured for 128-bit security during our experimentation. Consequently, we have selected $\|\mathsf{p}\|$ to be 3072 bits and $\|\mathsf{q}\|$ to be (256 + 2), which totals 258 bits. Additionally, we have chosen $\|\mathsf{r}\|$ to be 64 bits, resulting in a collision probability of approximately 4.6×10^{-11} [75] during the shelter update operation. For our experiment, we utilized servers with 16-core CPUs (Intel Xeon @2494 MHz) and 120GB of RAM. For simulating a low-power client, we used a single-core machine with 4GB of RAM, although actual RAM utilization was just a few megabytes.

We evaluated the performance of our protocol and compared it with two of the leading PIR schemes, PIANO [53] and SimplePIR [83]. Although both PIANO and SimplePIR are based on pre-processing techniques, they have some similarities with our approach. Particularly, like they require pre-processing and associated cost, we also require re-shuffling activities after each epoch. To ensure a fair comparison, we adopted the same storage parameters as PIANO and SimplePIR during our experiments, utilizing a database size of 100GB divided into 512-bit blocks, i.e., $\mathsf{N} = 1677721600$.

Table: 3.6 presents the results of the comparison. In our scheme, the PIR client is responsible for executing only a limited number of computations. In total, the client needs to remain active for 27 milliseconds for each request. In terms of computation, the client transfers FHE ciphertext (approximately 257 KB), El-Gamal ciphertexts, and receives the masked response. The client is only required to store the plaintext mask during the request processing, which is just $\{\mathsf{B} + \log \mathsf{N}\}$-bit in size.

During the processing of each request, around eleven FHE ciphertexts (irrespective of N) - each sized at 257KB - are exchanged among the servers, constituting the bulk of the online communication. As expected from asymptotic analysis, the amortized offline communication between the servers is $O(\sqrt{\mathsf{N}})$. This communication amount exceeds that of other PIR schemes. This is reflected in the measured amortized value of 36MB.

In our scheme, the average time taken by the server to process each request is 423 ms. Although this is a relatively very low server processing time given the size of the database, we acknowledge that it is not the absolute minimum value. While this online server time is 28× faster than Simple-PIR, it falls short of the speed achieved by PIANO (which has the

Table 3.6: Empirical analysis ($\mathsf{N} = 1677721600$)

		Simple-PIR (100GB)	PIANO (100GB)	Basic Var.* (100GB)	DP Var.* (100GB)	DPF Var. (100GB)	DPF Var.* (1 TB)
Client	Time	∼7 hr	∼3 hr	27 ms	27 ms	27 ms	28ms
	Comm.	1.2 GB	100 GB	260 KB	260 KB	260 KB	267 KB
	Stor.	1.2 GB	839 MB	543 bits	543 bits	543 bits	∼6.26 KB
Server†	Online time	10.9 s	11.9 ms	∼ 8 s	∼ 2.2 s	423 ms⋈	453ms
	Online com.	2.3 MB	100 KB	2.5 MB	260 MB	4.5 MB	4.5MB
	Offline time	29.6 ms	13.2 ms	4 min	12 min	281ms	285ms
	Offline com.	1.4 KB	120 KB	10 GB	30 GB	36 MB	360MB
Total‡	Time	∼7 hr	∼3 hr	∼4 min	∼12 min	∼731ms	∼766ms
	Com.	∼1.2 GB	∼100 GB	∼10 GB	∼30 GB	∼41 MB	∼41 MB

† Server cost means the total cost incurred by all three servers together
⋈ Dominated by DPF-based shelter search. In our experiment $\sqrt{\mathsf{N}} = 40960$. However, on average, the shelter will remain half-full and will contain 20480 elements.
‡ Total includes both client and server overhead (considering both online and offline phase)
* These columns represent extrapolated values.

lowest empirical online-time, to date) for the experimented database size. However, since our solution asymptotically reduces server overhead, it will be a more efficient solution than others for future large databases.

Furthermore, to ensure comparability, we selected a very small block size of 512 bits, which is not optimal for the FHE-encryption scheme. In fact, the FHE-encryption scheme can securely manage a block of 5120 bits in a single ciphertext. Consequently, as illustrated in the last column of the table, our scheme exhibits nearly identical performance, even for a database that is 10 times larger.

Most importantly, when we take into account the overhead incurred by the client is included in the analysis (please refer to Table: 3.6), our scheme demonstrates considerable practicality overall. For now, while we acknowledge that further enhancements are possible in the future, our scheme can still be regarded as a viable and practical PIR scheme.

3.5 Benefits of Our PIR Schemes

Our PIR schemes not only offer essential privacy guarantees but also provide additional advantages for real-world applications. In terms of overhead, they reduce the burden on the server without imposing any additional stress on the clients. Furthermore, our schemes

enable the server to maintain aggregated statistics and allow clients to detect any malicious behavior.

3.5.1 Practicality: Considering both Server and Client Overhead

A comparison of the asymptotic overhead of our PIR schemes with existing options is presented in Table: 3.7. Our schemes exhibit significantly lower server overhead, in asymptotic terms, compared with any other currently available options. Most importantly, our schemes do not put any burden on the clients, while reducing the server overheads. Specifically, the computational, communication, and storage requirements remain at $O(1)$ for clients. This characteristic makes our schemes highly appealing, especially for resource-constrained clients, such as IoT devices.

We have a full implementation of our most practical scheme, the *DPF Variant* and the detailed experimental results can be found in Table: 3.6. Although we have not been able to implement the complete versions of our *Basic Scheme* and *DP-variant* due to time constraints and limited experimental hardware resources, we have conducted some preliminary experiments with them and extrapolated those findings in the table. Moreover, we also inferred the performance of our scheme with larger database sizes. All these demonstrate the practicality of our schemes, especially when considering the client and server effort together.

While our schemes may result in increased online inter-server communication overhead, data centres frequently utilize specialized low-latency communication channels to connect with one another, even in a multi-cloud provider context [86], effectively circumventing standard internet routing. This approach helps mitigate additional traffic on public communication networks. Consequently, we do not view the added online inter-server communication costs as a significant obstacle, particularly when privacy is of higher priority. Our schemes require additional server storage, but we believe that the substantial recent reduction in server storage costs [39] makes this less of an issue. This consideration is especially important as servers must address both the growing concerns regarding privacy and the need for low-latency data delivery.

3.5.2 Maintaining Aggregated Statistics

While PETs can offer significant privacy benefits for clients, they often diminish the capabilities of servers, which can deter their adoption in real-world applications. For instance, since PIR does not expose any access patterns to servers, they are unable to analyze how

Table 3.7: Asymptotic Overhead Comparisons⁎

Solution	Server Computation	Client Comp.	# of Costly Operations‡	Server Comm.	Client Comm.⋈	Server Storage	Client Storage⋈
Chor et al. [47]	$O(\mathsf{N})$	$O(1)$	$O(1)$	$O(\mathsf{N}^{\frac{1}{3}})$	$O(\mathsf{N}^{\frac{1}{3}})$	$O(1)$	$O(1)$
Beimel et al. [48]	$O(\mathsf{N})$	$O(1)$	$O(1)$	$O(\mathsf{N}^{\frac{1}{3}})$	$O(\mathsf{N}^{\frac{1}{3}})$	$O(1)$	$O(1)$
Zhou et al. [53]	$\widetilde{O}(\sqrt{\mathsf{N}})$	$\widetilde{O}(\sqrt{\mathsf{N}})$	$O(1)$	$O(\sqrt{\mathsf{N}})$	$O(\sqrt{\mathsf{N}})$	$O(\mathsf{N})$	$\widetilde{O}(\sqrt{\mathsf{N}})$
Ostrovsky-Shoup [55]†	$O(\mathsf{plog}(\mathsf{N}))$	$O(\mathsf{plog}(\mathsf{N}))$	$O(\mathsf{plog}(\mathsf{N}))$	$O(\mathsf{plog}(\mathsf{N}))$	$O(\mathsf{plog}(\mathsf{N}))$	$O(\mathsf{N})$	$O(1)$
Lu & Ostrovsky [84]†	$O(\log \mathsf{N})$	$O(\log \mathsf{N})$	$O(\log \mathsf{N})$	$O(\log \mathsf{N})$	$O(\log \mathsf{N})$	$O(\mathsf{N})$	$O(1)$
Falk et al. [85]†	$O(\mathsf{plog}(\mathsf{N}))$	O(1)	$O(\log \mathsf{N})$	$O(\mathsf{plog}(\mathsf{N}))$	$O(\log \mathsf{N})$	$O(\mathsf{N})$	$O(1)$
Boyle et al. [68]	$O(\mathsf{N})$	$O(\log \mathsf{N})$	$O(1)$	$O(1)$	$O(1)$	$O(\mathsf{N})$	$O(1)$
Doerner & shelat [69]†	$O(\mathsf{N})$	$O(\log \mathsf{N})$	$O(\log \mathsf{N})$	$O(\log \mathsf{N})$	$O(1)$	$O(\mathsf{N})$	$O(1)$
DUORAM [70]†	$O(\mathsf{N})$	$O(\log \mathsf{N})$	$O(1)$	$O(1)$	$O(1)$	$O(\mathsf{N})$	$O(1)$
Ramen [71]†	$O(\sqrt{\mathsf{N}}\log\left(\mathsf{N}+\sqrt{\mathsf{N}}\right))$	$O(\log \mathsf{N})$	$O(1)$	$O(1)$	$O(1)$	$O(\mathsf{N})$	$O(1)$
Our *Basic Scheme*	$O(\sqrt{\mathsf{N}})$	$O(1)$	$O(\sqrt{\mathsf{N}})$	$O(\sqrt{\mathsf{N}})$	$O(1)$	$O(\mathsf{N})$	$O(1)$
Our *DP-Variant Scheme*	$< O(\sqrt{\mathsf{N}})$	$O(1)$	$< O(\sqrt{\mathsf{N}})$	$< O(\sqrt{\mathsf{N}})$	$O(1)$	$O(\mathsf{N})$	$O(1)$
Our *DPF-Variant Scheme*	$O(\sqrt{\mathsf{N}})$	$O(1)$	$O(1)$	$O(\sqrt{\mathsf{N}})$	$O(1)$	$O(\mathsf{N})$	$O(1)$

⁎ Security parameters are not considered in this comparison, since they will affect all the schemes similarly.
‡ This is the amount of expensive operations per query that can be performed by the client, the server, or both.
⋈ To identify one item out of N, the client must first locally generate (and therefore store) and then send a request that is at least $\log \mathsf{N}$ bits long. However, this is not classified as $O(\mathsf{logN})$ complexity; rather, complexity is assessed based on the number of blocks required to be transferred or stored by the client.
† These are all primarily intended for an MPC context but can likely be adapted to a PIR scheme. All values are based on this assumption.

their data is consumed. This functionality is essential in certain scenarios. Take YouTube, for example: if PIR were implemented, the server would be unable to track how many times a video is viewed, ultimately hindering its ability to compensate content creators. This could jeopardize the sustainability of the creator economy platform. On the other hand, it also remains essential to protect individual clients' personal viewing habits from the server's scrutiny.

Our PIR schemes can be advantageous in such scenarios. Optionally, all our schemes allow S_β to maintain statistics on retrieved blocks with only a few straightforward modifications. At the conclusion of processing each request, in all three of our schemes, S_γ receives the FHE-encrypted response, $\overline{(\mathbb{D}[\mathsf{I}]\|\mathsf{I})}$ (refer to Figures 3.3, 3.6, 3.13). Although S_γ cannot decrypt these ciphertexts, it is able to store them. After a sufficient number of requests have been processed - potentially aggregated over multiple epochs -these ciphertexts can be sent to S_β in a random order.

Once S_β receives the ciphertexts, S_β can decrypt them. In our schemes, the actual block index is also included in the response, allowing S_β to examine the index portion of all the plaintext responses. From this analysis, it can easily determine how many times each block has been accessed. However, S_β remains unaware of which specific client accessed which particular block, as the ciphertexts are received in a random order.

It is not surprising that enabling this feature will reduce the level of privacy associated with a pure PIR scheme. Depending on the context, this feature may either be activated or remain deactivated. In fact, it is possible to achieve a balance between client privacy and the accuracy of the analytical results by implementing additional strategies. For example, S_γ may choose not to send a few randomly selected ciphertext responses to S_β or may opt to duplicate some of these responses (also requires to add $\overline{\{0\}^{\mathsf{B}+\log \mathsf{N}}}$ homomorphically to conceal the fact that these are duplicated ciphertexts) before transmission. This approach yields a *noisy* histogram of aggregated requests sent to S_β, thereby enhancing client privacy.

3.5.3 Detection of Malicious Behavior

Like many other PIR-researchers, we assume the existence of multiple non-colluding servers, each of which is *honest-but-curious*. Becoming malicious or non-compliant with the established protocol might not provide the servers with the clients' secrets. However, servers may still have motivations to deviate from the established protocol steps. Especially because PIR is fundamentally a resource-consuming protocol for a server's point of view. Therefore, by deviating from the protocol and skipping some steps, the amount of consumed resources may be reduced.

Our schemes have some advantages in this aspect. Our schemes allow the clients to detect malicious behavior. Our scheme returns $(\mathbb{D}[\mathsf{I}]||\mathsf{I})$ to the client, and the client can verify the I part from the response. If any server fails to execute the protocol correctly, the client does not receive the requested index within the response.

This verifiability property of our scheme enables the client to identify malicious behavior. Although due to the involvement of three servers, the client cannot pinpoint the specific non-compliant server, the client can still report overall non-compliance. As a result, each server is impelled to adhere to the established protocol.

Chapter 4

Protecting Personal Data in a single location

When using a web service, individuals often need to share personal information with the service provider. In these situations, two conflicting interests come into play. On the one hand, the service provider aims to utilize that data for various computations. On the other hand, the service user seeks reassurance that their privacy is safeguarded and that their personal information remains protected.

4.1 Motivation: Enforcing Privacy Policies and a Data Expiry Mechanism

While privacy regulations require service providers to undergo privacy audits, successfully passing such an audit doesn't guarantee that these protections will be enforced consistently over time [87]. Typically, service users have no control over the remote environments where service providers store and process their information. Since the data is shared digitally, it is all too easy for a dishonest service provider to keep a hidden copy of that and misuse it. Controlling, even tracing, such covert usage can be difficult.

For example, during the pandemic, it became a common practice for many web service providers, such as hotel booking and airline reservation websites, to request vaccination certificates from their users. Typically, these websites included privacy policies stating that the vaccination certificates would only be used to verify users' eligibility for travel and accommodations, and that they would be deleted from the system after a specified

period. Users provided their vaccination certificates with the expectation that these privacy policy terms would be respected, especially given the strict legal regulations surrounding the data-privacy and the websites' compliance with these regulations.

Nevertheless, a dishonest provider can pretend to behave honestly in front of auditors and obtain the compliance certificate from them. Afterwards, in the auditors' absence, they could covertly retain copies of the vaccination certificates and later misuse them. For instance, imagine a future scenario where medical research uncovers a correlation between a specific vaccine and an increased risk of certain diseases. In this case, the provider might sell this information to unscrupulous entities, such as amoral insurance companies, which could then leverage it to deny insurance applications for individuals who received the vaccine.

In a digital world, once private data is exposed in plaintext, regaining control is nearly impossible because covert copies are so easy to create. A potential solution is to avoid disclosing private data in plaintext entirely, while still enabling the recipient to perform computations on it. Significant research effort has been made to address this challenge. Homomorphic encryption [28] enables computations on ciphertexts, but it cannot be applied naively to enforce privacy policies, as the recipient of the ciphertexts can perform any computation on them.

Conversely, functional encryption [88] can tie ciphertexts to specific computations; however, it remains prohibitively expensive, hindering its practical application in real-world scenarios. We investigate different ways to implement a *data-rotting* function, which would make the shared data - even in ciphertext form - unusable once the expiry time has elapsed.

4.1.1 Trusted Execution Environment: A Practical Alternative

Regarding privacy-preserving computation, an increasing number of processor manufacturers and cloud service providers are now offering Trusted Execution Environments (TEEs), a promising and practical hardware-based technology. Unlike homomorphic encryption, TEEs do not require performing computations on ciphertexts, making computations much more efficient. A TEE obtains a ciphertext, decrypts it only within its trusted and tamper-proof hardware registers while performing computations on it. However, the security properties of the TEE environment ensure that no one, including the manufacturers of the TEE hardware, can ever learn the plaintext content. Thus, it effectively achieves the same guarantee of an FHE scheme.

Furthermore, TEE can provide an unforgeable proof of the computing software being utilized. Various research efforts have combined these features to develop several PETs

that are significantly more efficient than traditional cryptographic implementations. For example, a functional encryption scheme [89] utilizing TEE has achieved performance levels that are 750,000 times faster than conventional pure cryptographic techniques.

While TEEs can provide efficient alternatives for several cryptographic primitives, tackling the data-rotting issue using TEEs is quite challenging. An adversary might cause a power interruption and cause a rollback attack. In this situation, the attacker can retain a copy of an older encrypted state and replay it as the most recent version during the next launch of the TEE. Therefore, in addition to developing a practical PET that leverages TEEs to ensure privacy policy enforcement, we now investigate ways to achieve the data expiry (or data-rotting property).

4.2 Related Work

In this research, we primarily concentrate on two aspects of privacy: the enforcement of privacy policies and data expiration. We examine the relevant literature related to these topics. Furthermore, since our emphasis is on a TEE-based solution, we also explore the significance of rollback attacks in this context and examine relevant work in these areas.

4.2.1 Privacy Policy Enforcement

XACML [90] was developed for the purpose of ensuring policy enforcement and ease of auditing. In this model, the data access rules to be followed by the data user are specified in a machine-readable XML format. Instead of directly accessing the data, the data user submits the data access request to a policy enforcement point (PEP). Then, the PEP forwards the request to another model called a policy decision point (PDP). The PDP checks the details of the data access request and consults the XACML policy to determine whether the request can be granted or not.

If the request is granted, then the PEP allows the data user to access the data item, while respecting the privacy policy. Policy enforcement can now be ensured by verifying whether the stated privacy policy on the website and the corresponding XACML policy are in alignment or not. However, a concern arises regarding the potential consequences if a malicious data user interferes with the execution flow of the PEP or PDP, or if they gain access to the personal data by disabling the PEP and PDP.

Instead of storing the private data in a single location, Liu et al. propose distributing

it amongst multiple brokers [91] using Shamir's secret sharing technique [92]. Each broker has their own PEP and PDP. The data user requests data fragments from multiple brokers.

On each data fragment request, each broker checks the privacy policy and enforces it with the combination of PEP and PDP. If enough data fragments are granted, then the data user is allowed access to the data. Multiple data brokers reduce the probability of cheating during policy enforcement. However, this mechanism works well only if the majority of the brokers are honest. Furthermore, a malicious data broker can not only allow illegitimate data access requests from the data user but also launch a DoS attack, by denying a legitimate request from the data user.

Smart contracts are programs stored within a blockchain [93]. These contracts execute reliably at a predefined time or upon the occurrence of specific events. Due to the transparent and unalterable nature of blockchains, smart contracts execute transparently and reliably. Alansari proposed [94] enforcing attribute-based access control policies during data access on the data user side, leveraging smart contracts. However, smart contracts are slow and costly. Costs are higher for large-sized smart contract code [95].

TEEs in Policy Enforcement

Birrell et al. introduced the concept of delegated monitoring in their work [96]. This mechanism involves a TEE on the data user side verifying the credentials of applicants seeking access to the personal data. In another study [97], an XACML-based access control approach is adapted, utilizing TEE. In this framework, a data owner signs the agreed-upon policy along with the requested data and submits it to the data user-side TEE. The TEE assesses the integrity of the PEP software. Only if it finds no signs of tampering, it shares the data with the PEP.

Since the actual data processing code operates outside of the TEE, personal data remains in plaintext format in RAM during processing. This poses a risk, as an adversary could potentially snoop the RAM contents and access the personal data. To mitigate this issue, Mazumdar et al. proposed Mitigator [98]. Their approach begins with the TEE verifying that the data processing software adheres to the specified privacy policy. If compliance is confirmed, the data processing code is then executed within the TEE itself by utilizing a local attestation mechanism [99]. As a result, both policy enforcement and the confidentiality of personal data can be effectively maintained.

In all of these proposed solutions, it is essential to recognize that the data owner could also be a malicious party, capable of exploiting these circumstances. For instance, they

might use another individual's vaccination certificate fraudulently to make hotel reservations. Consequently, while maintaining data confidentiality and enforcing policies is important for the data owner, ensuring the authenticity of personal data is crucial for the data users.

4.2.2 Data Expiry

Restricting malicious users from accessing personal information after the retention period poses significant challenges. Even with established deletion procedures, there remains a risk that data may be copied and exploited later. In EphPub [100], information is stored in an encrypted format, with bits of the encryption key represented by the states of various public DNS resolver caches. These DNS cache states are volatile and regularly updated, meaning that after a certain period, the encoded key material becomes unavailable, rendering the data unusable.

Vanish [101] used the inherent volatile nature of the distributed hash table (DHT) [102]. It encrypts the data with a random encryption key and then uses Shamir secret sharing technique on the encryption key. It distributes the parts of the key across random nodes in the DHT. Due to the constantly changing nature of the DHT, after some time, there are no longer shares of the keys to reconstruct the data.

Comet [103] uses a special type of hardware storage called active storage media [104]. In this storage, the data is stored as an active storage object. A special handler is registered to delete the object, and these handlers are automatically invoked after the expiry time. One major issue with all of these approaches is that data users may maintain their own copy of the plaintext data, which allows for continued usage after the expiration period has passed. Thus, the data must not be revealed in plaintext form at all.

Enforcing Data Expiry with TEE

SGX, the TEE developed by Intel, had a notable feature: it can maintain certain monotonic counters within its internal non-volatile storage. TEERASE [105] utilized this specific property of Intel SGX to implement a time-limited data access mechanism, which is secure against the rollback attack. However, subsequent research has revealed that accessing these counters is time-intensive (taking approximately 140ms per read) and is susceptible to various attacks [106]. Additionally, these counters exhibit instability after a specified number of accesses, leading to their removal from the current Intel SGX feature set.

Recently, Gao et al. proposed TEEKAP [107]. It does not use any special feature of any particular TEE technology but protects against rollback attacks by distributing the state information among multiple trusted parties, known as access committee members. One downside is that, it requires active involvement of those additional parties. As a result, TEEKAP requires additional resources and assumptions, such as a majority-uncorrupted, sufficient availability, etc. The majority-uncorrupted parties assumption can be relaxed, but in that case, all the access committee members need access to the TEE.

4.2.3 Protection against Rollback attack on TEEs

Ariadne [108], much like TEERASE, utilizes SGX counters to protect against rollback attacks. To provide resilience against crashes, it implements a store-then-increment mechanism for updating the counter. To minimize the likelihood of race conditions, Ariadne ingeniously designs the counter increment by flipping only a single bit. Memoir [109] employs Trusted Platform Module (TPM) technology [110], another different trusted hardware module, to safeguard against rollback attacks targeting TEEs. Additionally, ICE [111] offers rollback protection through modifications at the hardware level. It enhances the CPU with protected volatile memory, a power supply, and a capacitor that ensures the most recent state is flushed to non-volatile memory during system shutdown.

One alternative approach [112, 113] is to use a separate trusted server to manage the latest state information. However, this approach carries a notable drawback: the centralized trusted state server may easily become a target for attacks. It is conceivable that an attacker could launch a denial-of-service (DoS) attack or disrupt the transmission of updated state messages to the trusted server. To mitigate the risk of this single point of failure, the trusted state server can be replicated through a Byzantine consensus mechanism. However, standard consensus protocols such as PBFT [114] involve multiple rounds of communication, exhibit high message complexity, and require multiple replicas.

LCM [115] suggests safeguarding against rollback attacks, by retaining certain state information on the trusted client (data owner) side. However, this requires the client to actively participate each time the program runs within the TEE. In the realm of web services, this implies that the service users must be involved whenever the service provider intends to perform computations with the data. Furthermore, there is a risk that the service users could be malicious and exploit this situation.

On the other hand, Matetic et al. introduce ROTE [116], a solution that implements rollback protection through a distributed system approach. Their key insight is that, in many practical scenarios, there exist multiple TEEs, who can support one another. When

a TEE updates its state, it saves a counter to a group of other TEEs. Later, when the TEE needs to recover its state, it retrieves counter values from the assisting TEEs to confirm that the recovered state data reflects the latest version. However, this mechanism is ineffective if only a single TEE is available.

4.3 *ROT*: A Retention and Operation Limitation Framework utilizing TEE

In the context of web-services, a *Service Provider* acts as a *Data User* and a *Service User* is the *Data Owner*. A data user (*DU*) seeks to process the private data (D) of a data owner (*DO*) in order to offer a service. D could be issued by a trusted data issuer (*DI*) (for instance, a vaccination certificate issued by the government). *DU* intends to carry out a particular action based on the outcome of a specific computation ($\mathcal{C}$) performed on D. For example, the *DU* might choose to either grant or deny the requested service to the *DO* (such as permitting or refusing flight ticket booking) or provide tailored services (like discounts for senior citizens).

DO shares D with *DU* after agreeing to the description of $\mathcal{C}$, which is outlined in the privacy policy of *DU*. Once *DU* acquires D, they may choose to store it and process it whenever required. Meanwhile, *DO* seeks to ensure that *DU* is only able to perform the agreed-upon computation, $\mathcal{C}$, on D, while also expecting that D remains usable only for a limited duration of time ($\mathbb{T}_{\mathsf{L}}$). We propose a solution, *ROT*, that effectively addresses both the **R**etention and **O**perational limitation issues by leveraging **T**EE on the *DU*'s side.

It is to be noted that, in certain countries and privacy regulations, instead of *Data Owner* and *Data User*, they use the terms *Data Controller* and *Data Processor*, respectively.

4.3.1 Background: Trusted Execution Environment

A Trusted Execution Environment facilitates the creation of secure and isolated software execution spaces known as *enclave*s. Multiple enclaves can exist within a TEE-enabled platform. When establishing an enclave, it can be configured to execute a designated code. A TEE provides two key properties: (a) **Confidentiality**, ensuring that after its creation, all content within an enclave - encompassing both code and data - remains encrypted and is accessible solely by the trusted hardware CPU core linked to the specific TEE, and (b)

Attested execution, supplying a computation output and a proof of which specific code was executed to generate that output.

Enclaves reside within RAM, making their contents inherently volatile. To preserve data across reboots, TEEs offer a feature known as **Sealing**. This capability enables an enclave to securely store information on permanent media, such as a disk. However, the sealed data remains encrypted, with the encryption key known solely to the owning enclave. This means that only the enclave, executing the same code within the same TEE, can later recover the sealed data.

Further details about TEEs can be found in Appendix E.

4.3.2 Overview: Process data within TEE and Leverage Blockchain Against Rollback Attack

Figure 4.1 gives a top-level overview of our solution.

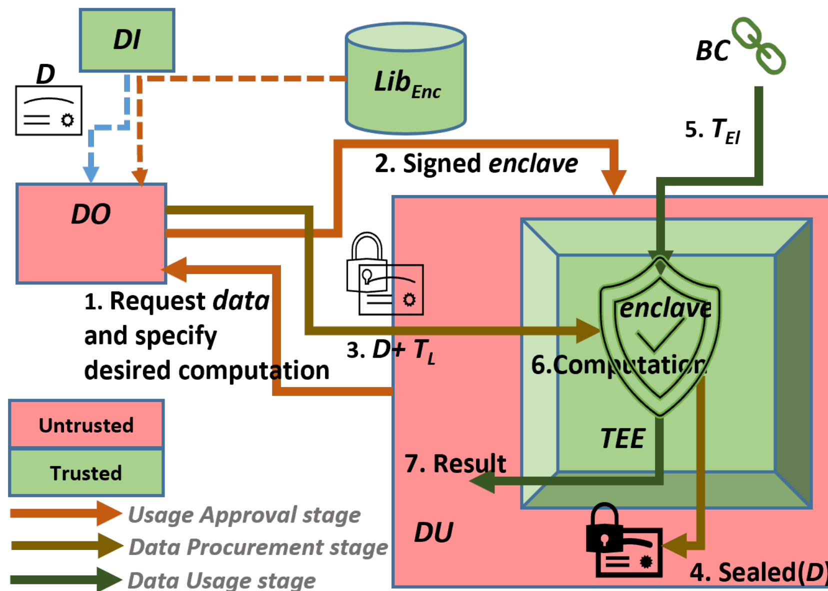


Figure 4.1: Top-Level Overview of ROT

DO already holds the private data, D, issued by DI. In addition to the actual data component, D contains the identity of DO and the signature of DI. This signature ensures

the authenticity of D. Prior to obtaining D from DO, the DU launches a new enclave within its TEE-enabled platform. However, even when operating within a TEE, a malicious enclave can still compromise private information (for instance, a malicious enclave might disclose the decryption of incoming inputs). To address this risk, ROT requires that DU executes enclaves solely from a trusted library, $\mathsf{Lib_{Enc}}$.

To ensure the security of the executing software, $\mathsf{Lib_{Enc}}$ can be made publicly accessible and maintained by a trusted entity. For instance, vaccination certificates can serve two specific purposes: The first purpose might be to confirm that the traveller has received all necessary vaccines within a designated timeframe prior to travel (without disclosing the exact dates). The second purpose might be to verify that the vaccines administered are exclusively from an approved list (without revealing the specific names of the vaccines). In this context, a trusted entity, such as a government body, can act as the maintainer of $\mathsf{Lib_{Enc}}$, ensuring that the associated stored enclave complies with established privacy rules and regulations. Furthermore, $\mathsf{Lib_{Enc}}$ maintains metadata for each of its stored enclaves, including information about the computations performed by the enclave, the hash value of that enclave.

Upon receiving a service request from DO, the DU requests the necessary data, from DO and specifies the computation that needs to be performed (Figure 4.1, Step 1). Subsequently, DO fetches the specific enclave from $\mathsf{Lib_{Enc}}$, signs and sends the signed enclave to DU (Step 2). Next, DU launches the signed enclave. Following this, DO securely transmits the data D to that newly launched enclave, along with a specified time limit ($\mathbb{T}_{\mathsf{L}}$). $\mathbb{T}_{\mathsf{L}}$ is represented in terms of the number of seconds.

The enclave stores both D and $\mathbb{T}_{\mathsf{L}}$ in non-volatile media in a sealed format for future use (Step 4). Whenever necessary, the DU requests the enclave to perform computations on D and return the results (Step 6 and 7). Since D is never exposed outside the enclave in plaintext, DU cannot retain a plaintext copy of D for uncontrolled use in the future. Although DU may attempt to keep a copy of the sealed format of D, only the specific enclave can recover its contents. Thus, in such cases, only the results of the DO approved computations can be obtained.

However, DU may still attempt to use the same enclave to perform the approved computation indefinitely. To address this risk and ensure data expiration, our solution leverages the inherent properties of blockchain technology. Specifically, a blockchain (BC) continuously updates its state at regular intervals. In our approach, the enclave tracks the blockchain state before performing the computation on the sealed data. Specifically, by monitoring the current blockchain state, it verifies that the elapsed time ($\mathbb{T}_{\mathsf{El}}$) has not surpassed the DO-specified $\mathbb{T}_{\mathsf{L}}$ (Figure 4.1, Step 5). Given that the blockchain is maintained in

a distributed manner and the underlying consensus mechanism (e.g., PoW [117]) ensures the reliability of the blockchain state, our solution effectively implements a *data-rotting* mechanism.

The notation used for ROT is summarized in Table: A.3.

4.4 ROT: Details

In this section, we discuss the details of our solution. First, we delve into the targeted threat model. Then, we will explore different data structures required during our protocol execution and the intricacies of the protocols.

4.4.1 Threat Model and Assumptions

Our threat model assumes the presence of a Byzantine adversary. In the context of ROT, this implies that both data owner and data user may engage in malicious behavior. The sole requirement for the data user in ROT is that they must possess a TEE-enabled platform, whether locally or in the cloud. Our protocol can detect whether a data user is using a genuine TEE; if not, it excludes the data user from the protocol.

We assume that DI is both honest and trustworthy. When it issues personal data (e.g., such as vaccination certificates) to DO, it does not disclose this information to any third parties. DU possesses a public key certificate and does not share its private key with anyone else. Furthermore, we presume that the enclaves within $\mathsf{Lib}_{\mathsf{Enc}}$ are designed to uphold privacy and execute computations as described. $\mathsf{Lib}_{\mathsf{Enc}}$ is accessible to both DO and DU and is assumed to provide all possible computations for all possible personal data types that a DU might require.

In designing our protocol, we took into account that support for TEEs on client-side platforms, such as desktop processors, has either slowed or stalled [118]. Our solution only necessitates TEE functionality on the server side. We believe this requirement is reasonable, considering the industry is poised to invest heavily in this technology [119]. There are efforts to standardize TEE-based infrastructures [120] and protocols.

We recognize that side-channel attacks have been identified as vulnerabilities for TEEs in certain scenarios. However, significant improvements have been implemented to bolster the security of TEE [40]. Furthermore, as previously mentioned in Section 2.7.3, the integration of $RouterORAM$ with a TEE can effectively mitigate any side-channel attacks

related to access pattern leakage. However, in the development of ROT, we operate under the premise that TEE hardware reliably upholds its expected security properties.

Specifically, once execution begins, the contents of the enclave become inaccessible, and the internal code remains unchanged. Beyond this point, we do not make any assumptions about security. For example, an adversary retains the ability to control or modify everything outside the enclave, including the operating system, communication channels, and the contents of RAM. They may also inject malicious inputs when invoking the enclave's entry points. In developing our protocol, we took into account the possibility that an adversary might attempt to disable or bypass the TEE, and we designed our protocol to effectively address these potential challenges.

4.4.2 Utilization of Blockchain Properties

Blockchains are distributed ledgers that maintain a sequential list of transactions organized into blocks. Each block is assigned a unique, incrementing block number. The underlying consensus mechanism guarantees the integrity of the global state of the blockchain as well as the contents of each individual block. In many widely used blockchains, such as Ethereum [121], this consensus mechanism also ensures that the rate of block generation remains constant, which can be utilized as a reliable reference for time measurement.

Imagine that right before the time measurement, the block number of the last added block to the blockchain was $\mathsf{BN_S}$. If the new block number immediately following the time measurement is $\mathsf{BN_E}$ and the average block generation time of the blockchain is $\mathbb{T}_\mathsf{B}$, then the estimated elapsed time ($\mathbb{T}_\mathsf{El}$) can be expressed as:

$$\mathbb{T}_\mathsf{El} = (\mathsf{BN_E} - \mathsf{BN_S}) \times \mathbb{T}_\mathsf{B} \tag{4.1}$$

A natural question may arise: why cannot existing trusted time sources, such as NTP servers, be utilised to track elapsed time? The answer lies in the fact that, an enclave cannot interact with the outside world independently. Therefore, for any time information to reach the enclave, it must pass through the host computer's network stack and operating system, both of which are vulnerable to adversarial control. This opens the possibility of manipulating that time information. In contrast, blockchain technology employs consensus mechanisms to ensure the integrity of individual block contents, including block numbers. As a result, it is not feasible to provide a manipulated block number to the enclave.

However, there is a possibility for a replay attack, wherein a malicious host could present a past (but valid) block during the enclave's time calculations, ultimately misleading it

and manipulating the elapsed time measurement. When the enclave needs to retrieve any block number ($\mathsf{BN_S}$ or $\mathsf{BN_E}$), it first generates a random value internally and writes it to the blockchain (with the help of host). Subsequently, the enclave performs a read operation on the blockchain to identify the block number that contains the newly written random value. This combination of newly generated randomness and the blockchain's consensus mechanism provides protection against replay attacks.

Our approach for estimating elapsed time utilizing blockchain offers some advantages. First, existing blockchains can be utilized for this purpose, eliminating the need to establish any new entities and corresponding new trust assumptions. Furthermore, a solution dependent on a trusted time server is vulnerable to a single point of failure [116]. In such scenarios, an external adversary can also easily disrupt access to the time server, effectively halting the operation of the enclave and resulting in a denial-of-service attack on DU. In contrast, the blockchain is inherently distributed, ensuring there will always be enough miners to maintain its global state.

In developing our solution, we utilize the Ethereum blockchain, which is among the most widely used blockchains today. It is important to note that Ethereum includes a timestamp in each block; however, this is not a standard feature across all blockchains. To ensure our solution remains as generic as possible, we chose not to use this feature, opting instead to leverage the consistent block generation rate.

4.4.3 Structure of Data Owner's Personal Data

Our solution restricts DO from transmitting arbitrary data to DU. Instead, it permits only certified private data to be shared. A trusted data issuer, DI, is responsible for certifying the personal data of DO. Consequently, in addition to the actual private information, other essential details such as the identity of the data owner, signature of DI must also be transmitted through the personal data structure, D.

Moreover, actual private information often encompasses multiple attributes. For example, the government issues a driver's license for the individual, which includes essential details such as date of birth, address, license number, and other pertinent information, all of which are certified by the government. To achieve these details, without designing a new structure, we utilize the *x509 Certificate* along with the *V3 extension* [122] while transmitting personal data.

With the specialized *V3 extension*, DI can issue a certificate to DO, detailing a customizable number of attributes about DO in the *X509v3 extensions* section of the certificate. Since personal data is encoded as X509v3 certificates, we can effectively utilize

existing PKI infrastructure while ensuring the authenticity of this data. In the context of *ROT*, it is expected that *DO* will consider this certificate as a confidential electronic document and provide it to the *DU*-side enclave upon request.

It is worth mentioning that, for the purpose of transmitting the personal data, an X.509 attribute certificate [123] would have been a better choice. However, X.509 attribute certificates are now obsolete, and current cryptographic libraries (e.g., OpenSSL) do not support them. Hence, considering the ease of implementation, we decided to choose X509v3 certificates over the X.509 attribute certificates.

4.4.4 Structure of the Computing Enclaves

$\mathsf{Lib}_{\mathsf{Enc}}$ contains numerous enclaves. While each enclave is designed to perform distinct actions, they also share several similarities and adhere to a common structure. Much like an executable binary, an enclave comprises various components, including the *code-section*, *data-section*, *enclave-stack*, *enclave-heap*, among others. An enclave additionally has a *control-structure*. Figure 4.2 illustrates the general structure of any enclave within $\mathsf{Lib}_{\mathsf{Enc}}$.

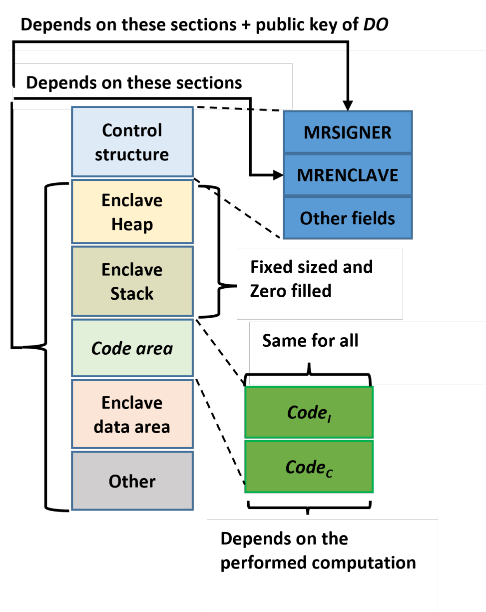


Figure 4.2: Structure of the enclave

Code Section:

The code section has two parts: $Code_I$ and $Code_{\mathcal{C}}$. The $Code_{\mathcal{C}}$ is responsible for performing the desired computation $\mathcal{C}$ that the DU intends to perform on D. This part, $Code_{\mathcal{C}}$, varies from enclave to enclave. In contrast, the $Code_I$ is independent of the specific computation $\mathcal{C}$ and remains consistent across all enclaves within $\mathsf{Lib}_{\mathsf{Enc}}$. The $Code_I$ carries out tasks such as verifying data authenticity and managing data expiry, which are common to all enclaves in $\mathsf{Lib}_{\mathsf{Enc}}$.

Stack, heap and data sections:

Unlike typical executables, an enclave is restricted from accessing any other area of RAM beyond its designated memory boundary. Consequently, it is crucial to define the contents of the stack, heap, data sections, and other components within the enclave's binary executable during its creation. The data section of each enclave varies according to its specific functionality.

Additionally, we incorporate certain pre-initialized information into the data section, such as the certificate of a trusted certification authority. In our solution, we store pre-built enclaves in $\mathsf{Lib}_{\mathsf{Enc}}$, each containing a fixed-size, zero-initialized stack and heap region. This design ensures that the enclave's hash, which performs a specific computation, remains constant.

Control-structure:

This section contains several pieces of important information, including *MRSIGNER*, *MRENCLAVE*, and the size of the enclave, among other details. Notably, *MRENCLAVE* holds the hash value representing the entire content of the enclave, except the control structure. *MRSIGNER* represents the *DO*'s signature on the enclave. This signature is calculated on the entire enclave content, with the public-key of the enclave owner. *MRSIGNER* remains unpopulated while the enclave is located in $\mathsf{Lib}_{\mathsf{Enc}}$. However, before launching the enclave on the *DU* side, the *DO* fills in this value. Before any enclave starts, the TEE hardware verifies this signature, along with the other contents of the control section. Then, only the enclave is granted execution permission.

4.4.5 Details of Protocol Stages

Our solution consists of three distinct stages:

- **Usage Approval** (Figure 4.3): During this initial stage, *DO* verifies the data usage request and then transmits a signed enclave to *DU*. Later, *DU* uses this enclave to perform computations on the personal data of *DO*.
- **Data Procurement** (Figure 4.4): Then, *DO* securely delivers personal data and pertinent information to the signed enclave, executing on the *DU* side.
- **Data Usage** (Figure 4.5): *DU* requests the enclave to perform the necessary computations and produce the result.

Usage Approval Stage:

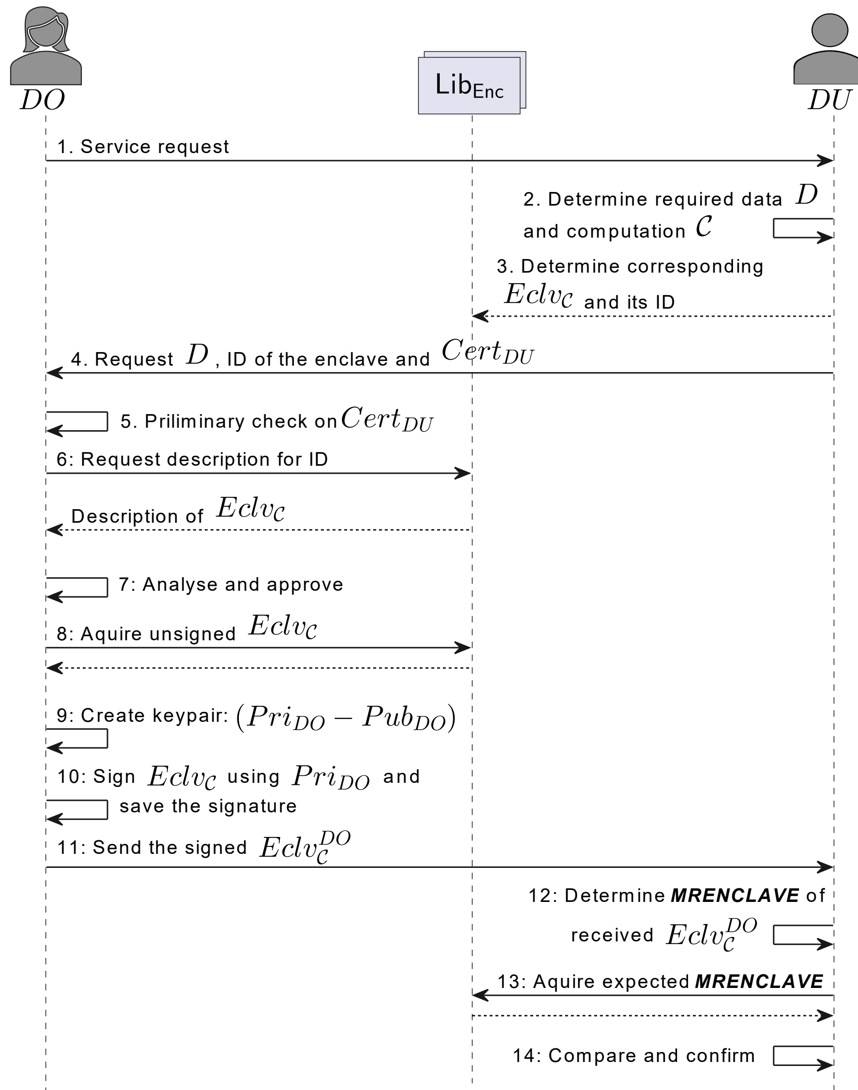


Figure 4.3: Sequence of Usage Approval Stage

DO initiates the process by sending a service request to DU. Subsequently, DU assesses which private data, D, about DO is necessary to access before providing the requested service. After that, DU determines, what computation, $\mathcal{C}$, is required on D and subsequently identifies the corresponding enclave, $Eclv_{\mathcal{C}}$, from $\mathsf{Lib}_{\mathsf{Enc}}$, along with its ID. Once these are

identified, DU sends a request for D and the ID of the $Eclv_{\mathcal{C}}$ back to DO.

Along with this, it provides its own certificate $Cert_{DU}$, which contains DU's public key, Pub_{DU}, essential for a later stage of the process. At this stage, DO only verifies that the received $Cert_{DU}$ was issued by a trusted certificate authority and not an expired or revoked certificate. To enhance the performance of our protocol, the verification of DU's ownership of $Cert_{DU}$ occurs during the data usage stage.

After getting the ID of the $Eclv_{\mathcal{C}}$, DO acquires the detailed description about the performed computation done by $Eclv_{\mathcal{C}}$ directly from the trusted $\mathsf{Lib}_{\mathsf{Enc}}$. Next DO decides whether to proceed further, by analyzing the detailed description. If there is agreement, DO retrieves the unsigned version of $Eclv_{\mathcal{C}}$ directly from $\mathsf{Lib}_{\mathsf{Enc}}$.

After this, DO generates an ephemeral key pair (Pri_{DO}-Pub_{DO}) and signs the acquired $Eclv_{\mathcal{C}}$ and produces $Eclv_{\mathcal{C}}^{DO}$. Specifically, DO populates the *SIGSTRUCT, MRSIGNER* and some other fields in the control structure of the enclave. DO locally stores the signature which helps DO to ensure privacy later. DO sends the entire signed enclave to DU.

During this usage approval stage, DO can act maliciously. DO might modify the original $Eclv_{\mathcal{C}}$ obtained from $\mathsf{Lib}_{\mathsf{Enc}}$ and subsequently provide a signed copy of the altered $Eclv_{\mathcal{C}}$ to DU. Therefore, prior to accepting the signed $Eclv_{\mathcal{C}}^{DO}$, DU determines its *MRENCLAVE* value, which does not change with the signature, to verify the integrity of the enclave's content (refer to Appendix E.1). DU then retrieves the expected $MRENCLAVE$ value of $Eclv_{\mathcal{C}}$ directly from $\mathsf{Lib}_{\mathsf{Enc}}$ and compares both $MRENCLAVE$ values. Only if they match, DU accepts the received enclave and deploys it on its TEE-enabled platform.

Data Procurement Stage:

DU deploys the DO-signed $Eclv_{\mathcal{C}}^{DO}$ on a TEE-enabled platform and directs it to retrieve data from the designated data owner identified by a specified identity (Step 1-2). The deployed $Eclv_{\mathcal{C}}^{DO}$ leverages this identity to ensure that only the intended data owner, DO, can send private data to the enclave. Subsequently, the enclave generates a key pair and discloses the public key, which is utilized to encrypt all communications between the enclave and DO.

Subsequently, DO establishes a specialized TLS channel with the deployed enclave, referred to as an attested-TLS channel [124, 125] via DU. During this process, the TLS client (in this case, DO) acquires not only the public key, epk, but also a cryptographic proof. This proof certifies to the TLS client that the owner of the keypair is indeed the designated enclave, identified by its unique *MRENCLAVE* and *MRSIGNER* values, which

DO recorded during the *Usage Approval Stage*. Consequently, any message transmitted through this channel can only be decrypted by $Eclv_C^{DO}$.

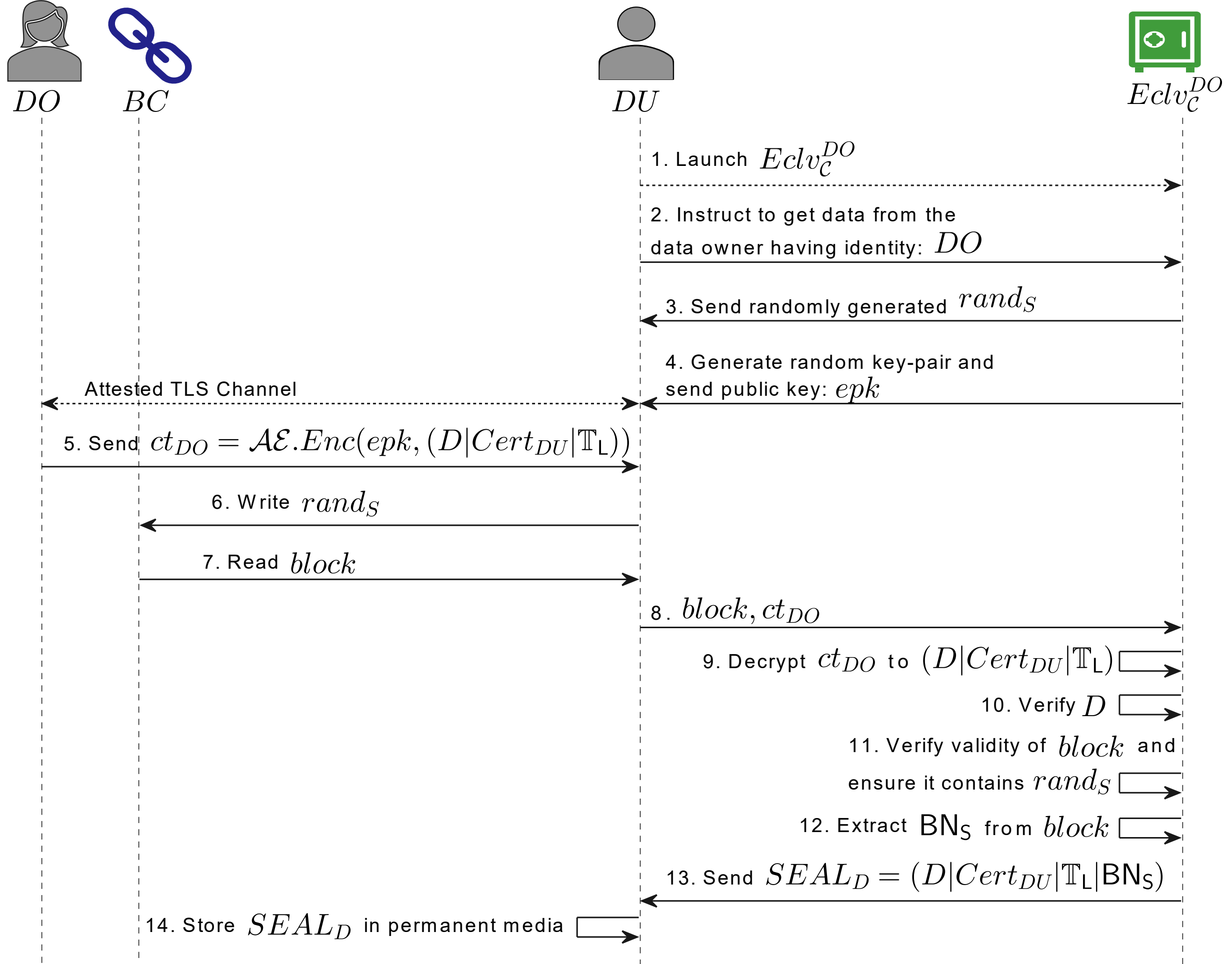


Figure 4.4: Sequence of Data Procurement Stage

Following this through the attested TLS-channel, the DO sends the private data D along with its usage time limit, $\mathbb{T}_\mathsf{L}$ (Step 5). To ensure that only DU can access and recover the computation results, DO also transmits the intended public key certificate, $Cert_{DU}$, which it received earlier from DU during the *Usage Approval Stage*. Upon receiving the ciphertexts via DU(Step 8), the enclave decrypts and verifies the authenticity of the personal data by examining the DI's signature. Additionally, it confirms that the

identity field within D corresponds to the identity specified by the DU during the enclave deployment.

During this stage, $Eclv_{\mathcal{C}}^{DO}$ also records the current block number in the blockchain, $\mathsf{BN_S}$, for future time calculations. To achieve this, as previously discussed in Section 4.4.2, $Eclv_{\mathcal{C}}^{DO}$ generates a fresh random value and, with the assistance of DU, writes this value to the blockchain. Following this, the enclave conducts a read operation on the blockchain to confirm that the newly generated random value has been recorded, noting the block number that contains this value as $\mathsf{BN_S}$. The blockchain's inherent immutability feature, combined with the freshly generated randomness, ensure protection against any replay attacks by DU.

Finally, the enclave stores D, $\mathbb{T}_\mathsf{L}$, $\mathsf{BN_S}$, and $Cert_{DU}$ into the permanent media by consolidating them into a single sealed format. The sealing key is determined based on the sealing identity [126]. As a result, only the enclave specifically signed by DO can recover this information. This process not only binds D to the computation agreed upon by DO but also ensures that only the $\mathbb{T}_\mathsf{L}$ specified by DO can be utilized for time-limit verification during data usage.

It is worth noting that the sealed data is simply a ciphertext. DU might have a plan of performing a *harvest-now, decrypt-later attack* [127] on that. However, that is not a concern in our scheme, since the sealed data remains encrypted under a secure symmetric-key encryption scheme, and a quantum computer is not a real threat to that. In particular, our experimental hardware version uses AES-128, which provides at least 64-bit security, even in the era of quantum computers. If an even stronger level of security is required, we just need to use the updated TEE hardware version, which increases the key length of the used AES scheme.

Data Usage Stage:

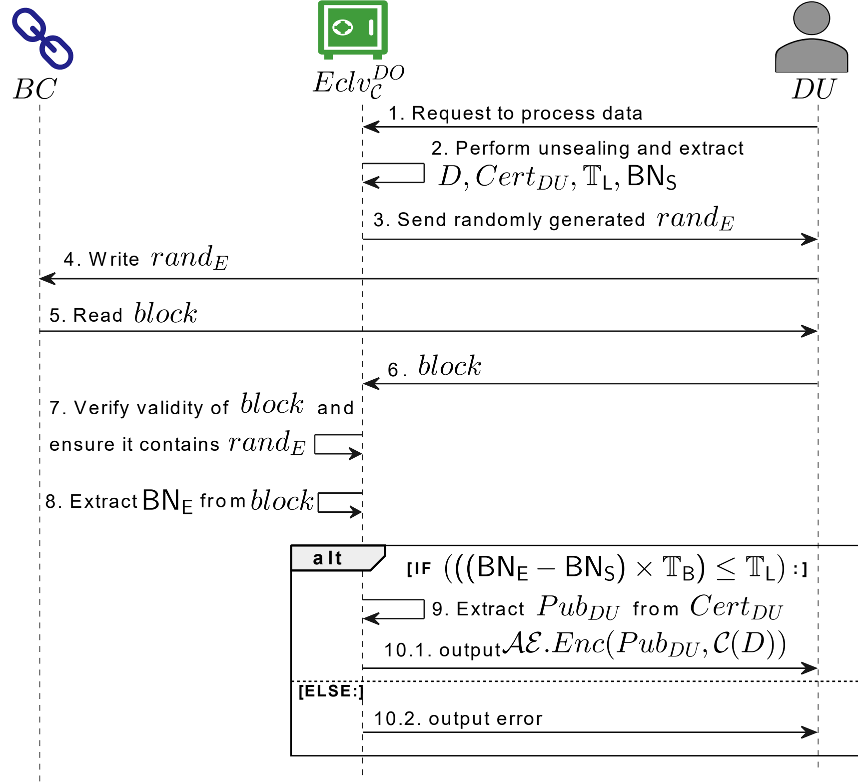


Figure 4.5: Sequence of Data Usage Stage

Once the personal data has been procured, DU may request the launched enclave to perform computation on it and deliver the result. Upon receiving the request, $Eclv_{\mathcal{C}}^{DO}$ first unseals the data and, using a similar mechanism to that employed in the earlier stage, verifies the current state of the blockchain to identify the latest block number, $\mathsf{BN_E}$. Next, by utilizing the unsealed $\mathsf{BN_S}$ value, it calculates the elapsed time as per Equation 4.1. If the data is still valid, $Eclv_{\mathcal{C}}^{DO}$ then proceeds to perform the computation $\mathcal{C}$ on D.

When returning the result, $Eclv_{\mathcal{C}}^{DO}$ encrypts the computation outcome using Pub_{DU}. The DU can then recover the plaintext result by utilizing its private key, Pri_{DU}. As Pri_{DU} remains confidential, the result is accessible solely to the intended DU. On the other hand, if the expiry time has lapsed, $Eclv_{\mathcal{C}}^{DO}$ will generate an error message, ensuring that the DU receives no information and that the data becomes impossible to use.

4.5 Privacy Analysis

We analyse the privacy of ROT in UC-Framework (please refer to Appendix : F for details of UC-Framework) and our analysis unfolds in multiple stages. Initially, a comprehensive formal representation of the entire protocol of ROT is established. Then, the expected privacy goals of our solution are determined. Subsequently, the ideal functionality for ROT, $\mathcal{F}_{ROT}$, is defined. Finally, $\mathcal{F}_{ROT}$ is utilized to demonstrate ROT's privacy and security within a UC-framework.

4.5.1 Formal Representation of ROT

To demonstrate security and privacy within a UC-framework, the protocol sequences already outlined in section 4.4.5 have been unified and formally represented in this section. Figure 4.6 illustrates the formal representation of the ROT protocol ($Prot_{ROT}$), while Figure 4.7 provides a formal description of the enclave program ($Prog^{ROT}_{Eclv}$). These representations rely on the existing ideal functionality of attested execution, G_{att} [1] and ideal functionality of blockchain, $\mathcal{F}_B$ [128, Figure 7].

$Prot_{ROT}$ interacts with the enclave program ($Prog^{ROT}_{Eclv}$) at specific entry-points, by issuing a "resume" command to G_{att} (i.e., lines 26, 30, 33, 36 in Figure 4.6). In $Prot_{ROT}$, DO, DU, and the $\mathsf{Lib}_{\mathsf{Enc}}$ each have distinct roles. DI is not considered in this formal analysis, since it does not take part during the online execution of the ROT protocol.

In the terminology of the UC-framework, the DU's GetDetails and SaveData entry points (refer to Figure 4.6, line 19, 24) and all the entry points of $\mathsf{Lib}_{\mathsf{Enc}}$ function as internal machines in $Prot_{ROT}$ and are not exposed to the external environment, $\mathcal{Z}$. As a result, these entry-points can only be controlled indirectly. In contrast, all other entry points of $Prot_{ROT}$ are directly accessible from $\mathcal{Z}$. The entry points are coded using colors: green entry points may only be invoked once, with any subsequent attempts resulting in a failure ($\perp$), while cyan entry points can be invoked multiple times.

The formal representation of the protocol and the enclave program closely follows the previously specified sequence flows, with one notable exception. For the purpose of facilitating simulation, we have introduced a backdoor in the "DataUsage" entry point of the enclave program (at line 29). If the input is ($y \neq \perp$), the enclave will simply return y along with its attestation. Conversely, if the input is ($y = \perp$), the enclave will provide the actual outcome of the evaluation.

$Prot_{ROT}[\Sigma, \Sigma_{TEE}, \mathcal{AE}, Enc[], Cert_{DU}, DO, DU, \mathsf{Lib_{Enc}}]$

<u>*DO*</u>**:**

1: On receive ("UsageApproval", SR) from $\mathcal{Z}$:
2: send("GetDetails", SR) to DU wait for $(Req(D), ID_{Enc}, Cert_{DU})$
3: assert $Cert_{DU}$ is valid
4: send("GetDesc", ID_{Enc}) to $\mathsf{Lib_{Enc}}$ wait for $desc$
5: send("GetEnc", ID_{Enc}) to $\mathsf{Lib_{Enc}}$ wait for $Eclv_{\mathcal{C}}$
6: $(Pri_{DO} - Pub_{DO}) \leftarrow \mathcal{AE}.KeyGen(1^{\lambda})$
7: $\sigma_{DO} := \Sigma.Sig(Pri_{DO}, Eclv_{\mathcal{C}})$ and update $Eclv_{\mathcal{C}}^{DO} = (Eclv_{\mathcal{C}}|\sigma_{DO})$
8: store σ_{DO} and send($Eclv_{\mathcal{C}}^{DO}$) to DU
9: On receive ("DataProcurement", D, $Cert_{DU}$, $\mathbb{T}_{\mathsf{L}}$) from $\mathcal{Z}$:
10: send("SaveData") to DU wait for $(eid, epk, \sigma_{TEE}(epk))$
11: assert $\Sigma_{TEE}.Vf(G_{att}.getpk(), (eid, Eclv_{\mathcal{C}}^{DO}, epk), \sigma_{TEE}(epk))$
12: send $\mathcal{AE}.Enc(epk, (D|Cert_{DU}|\mathbb{T}_{\mathsf{L}}))$ to DU

<u>$\mathsf{Lib_{Enc}}$</u>**:** ▷ All invokable entry points of $\mathsf{Lib_{Enc}}$, act as internal machine

13: On initialize ($Enc[]$): store $Enc[]$ for future use ▷ $Enc[]$ holds details of all the enclaves
14: On receive ("GetID", $desc$):
15: output ID for which $Enc[ID].desc = desc$
16: On receive ("GetDesc", ID): output $Enc[ID].desc$
17: On receive ("GetEnc", ID): output $Enc[ID].enc$
18: On receive ("GetMRENCLAVE", ID): output $Enc[ID].MRENCLAVE$

<u>*DU*</u>**:**

19: On initialize ($Cert_{DU}, Pri_{DU}$): store $Cert_{DU}, Pri_{DU}$ for future use
20: On receive ("GetDetails", SR) from DO: ▷ This entry point, acts as an internal machine
21: determine $Req(D)$ and $desc$ ▷ Determine data requirement and the description of the usage
22: send("GetID", $desc$) to $\mathsf{Lib_{Enc}}$ wait for ID_{Enc}
23: store DO and send $(Req(D), ID_{Enc}, Cert_{DU})$ to DO
24: On receive ("SaveData") from DO: ▷ This entry point, acts as an internal machine
25: send("install", $Eclv_{\mathcal{C}}^{DO}$) to G_{att} wait for eid
26: send("resume", eid, ("EncGetData", DO)) to G_{att} wait for $(rand_S, (epk, \sigma_{TEE}(eid, Eclv_{\mathcal{C}}^{DO}, epk)))$
▷ Henceforth $\sigma_{TEE}(eid, Eclv_{\mathcal{C}}^{DO}, .)$ is denoted as $\sigma_{TEE}(.)$
27: send($(epk, \sigma_{TEE}(epk))$) to DO wait for ct_{DO}
28: send("write", $rand_S$) to $\mathcal{F}_B$
29: send("read") to $\mathcal{F}_B$ wait for $block$
30: send("resume", eid, ("EncSaveData", $ct_{DO}, block$)) to G_{att} wait for $SEAL_D$
31: store $SEAL_D$ in parmanent storage
32: On receive ("DataUsage", y) from $\mathcal{Z}$: ▷ y is used only during simulation, by $\mathcal{F}_{ROT}$
33: send("resume", eid, ("EncDataUsage", $\perp$, $\perp$)) to G_{att} wait for $rand_E$
34: send("write", $rand_E$) to $\mathcal{F}_B$
35: send("read") to $\mathcal{F}_B$ wait for $block$
36: send("resume", eid, ("EncDataUsage", $block$, $\perp$)) to G_{att} wait for $\mathcal{AE}.Enc(Pub_{DU}, \mathcal{C}(\mathcal{D}))$
37: obtain the computation result, $\mathcal{C}(\mathcal{D})$, by decrypting the ciphertext with Pri_{DU}

Figure 4.6: Formal representation of the ROT protocol

$Prog^{ROT}_{Eclv}$

1: On receive ("EncGetData", DO):
2: assert ($state = \varnothing$)
3: store DO
4: $(epk - esk) \leftarrow \mathcal{AE}.KeyGen(1^\lambda)$
5: generate a random: $rand_S$
6: $state := GetData$
7: **output** $((rand_S, \sigma_{TEE}(rand_S)), (epk, \sigma_{TEE}(epk)))$ ▷ Along with each actual output (.), enclave always attaches an attestation, $\sigma_{TEE}(.)$. For brevity, it is not shown for the subsequent outputs.
8: On receive ("EncSaveData", $ct_{DO}, block$)):
9: assert ($state = GetData$)
10: $\mathcal{AE}.Dec(esk, ct_{DO})$ and parse as $(D|Cert_{DU}|\mathbb{T}_\mathsf{L})$
11: assert ($D.id = DO$)
12: assert $\Sigma_{DI}.Vf(D.sig)$ ▷ Verify DI's signature on the data. DI's public-key is already installed within the enclave
13: assert validity of $block$ ▷ Details depend on the consensus mechanism (e.g., PoW)
14: assert $rand_S \in block$
15: extract $\mathsf{BN_S}$ from $block$
16: $state := ProcessData$ and **output** $SEAL_D = SealedVersion(D|Cert_{DU}|\mathbb{T}_\mathsf{L}|\mathsf{BN_S})$
17: On receive ("EncDataUsage", $block$, y):
18: assert ($state = ProcessData$)
19: **if** ($block = \perp$):
20: generate a random: $rand_E$
21: **output** $rand_E$
22: **else**:
23: assert $rand_E \in block$
24: extract $\mathsf{BN_E}$ from $block$
25: unseal $SEAL_D$ as $(D|Cert_{DU}|\mathbb{T}_\mathsf{L}|\mathsf{BN_S})$
26: extract Pub_{DU} from $Cert_{DU}$
27: assert $(((\mathsf{BN_E} - \mathsf{BN_S}) \times \mathbb{T}_\mathsf{B}) \leq \mathbb{T}_\mathsf{L})$
28: **if** ($y = \perp$): **output** $\mathcal{AE}.Enc(Pub_{DU}, \mathcal{C}(D))$ ▷ Actual computation result
29: **else**: **output** $\mathcal{AE}.Enc(Pub_{DU}, y)$ ▷ This is a backdoor, implanted to achieve equivocation

Figure 4.7: Enclave program of ROT

The implementation of this specific backdoor allows the simulator to achieve **equivocation**, which is a known trick, utilized during the simulation process. In a real-world protocol, an honest DU will always submit a value of ($y = \perp$). If a malicious DU submits ($y \neq \perp$), it does not gain any additional information beyond the input y and its attestation. As a result, this backdoor does not provide any advantage to the adversary. Rather,

it simply aids the simulator during the UC-proof. Additionally, it is important to note a common phenomenon within the UC-proofs: both the adversary and the simulator may possess a TEE and launch their own enclaves alongside real-world parties [1]. However, none of these entities can compromise the security properties of the TEE.

Two different EUF-CMA-secure digital signature schemes are used in ROT, one is, $\Sigma(KGen, Sig, Vf)$, which is used for verifying the authenticity of D. $\Sigma_{TEE}(KGen, Sig, Vf)$ is another signature scheme, used by TEEs to attest to legitimacy of the output of the TEEs in question. An IND-CPA-secure asymmetric encryption scheme $\mathcal{AE}(KeyGen, Enc, Dec)$ is used to communicate securely with the enclave.

4.5.2 Privacy and Security Goals

The following are the specific privacy and security goals of our solution.

- DU should never be able to access the plaintext version of the received data, D and can only learn the computation result on it.
- Still, DU should be able to verify the authenticity of D and get the assurance of the fact that the received result is obtained after performing the intended computation, on the intended data.
- DU should not be able to perform any other computation on D, except the DO-agreed computation, $\mathcal{C}$.
- DU should not be able to perform $\mathcal{C}$ on D, after DO-specified expiry time, $\mathbb{T}_{\mathsf{L}}$.
- Computation result on D, should only be available to the intended recipient, DU.

4.5.3 Ideal Functionality

Based on the desired privacy and security goals (section: 4.5.2) and the specified threat model (section: 4.4.1), we define the ideal functionality for our protocol, $\mathcal{F}_{ROT}$ (Figure 4.8). In $Prot_{ROT}$, there are three entry points callable by $\mathcal{Z}$. Consequently, $\mathcal{F}_{ROT}$ also has only those entry points. The following sections describe the design of each entry point of $\mathcal{F}_{ROT}$.

As a fundamental construct of a UC-Framework [129], the caller of $\mathcal{F}_{ROT}$ is either an honest party or the simulator, Sim. If $\mathcal{F}_{ROT}$ sends a message, it first goes to the adversary, $\mathcal{A}$. If $\mathcal{A}$ allows, then the message is forwarded on to the proper recipient. In the ideal world,

Sim functions by executing $\mathcal{A}$ as an internal subroutine. Consequently, any communication between $\mathcal{F}_{ROT}$ and $\mathcal{A}$ is routed through Sim.

$\mathcal{F}_{ROT}[\Sigma, \Sigma_{TEE}, \mathcal{AE}, Enc[], Cert_{DU}, DO, DU]$

1: On receive ("UsageApproval", SR) from DO:
2: assert ($state = \varnothing$)
3: faithfully determine corresponding $Req(D), \mathcal{C}$
4: findout $desc, ID_{Enc}, Eclv_{\mathcal{C}}$ from $Enc[]$
5: notify $\mathcal{A}$ about ("UsageApproval", DO, DU, $Req(D), ID_{Enc}, Eclv_{\mathcal{C}}, Cert_{DU}$) and block until $\mathcal{A}$ replies
6: store $\mathcal{C}, DO$ and **output** "OK" to DO, DU
7: $state := DataProcurement$
8: On receive ("DataProcurement", D, $Cert_{DU}$, $\mathbb{T}_{\mathsf{L}}$) from DO:
9: assert ($state = DataProcurement$)
10: notify $\mathcal{A}$ about ("DataProcurement", DO, DU, $|(D|Cert_{DU}|\mathbb{T}_{\mathsf{L}})|$) and block until $\mathcal{A}$ replies
11: determine current time: τ_s
12: store $(D, Cert_{DU}, \mathbb{T}_{\mathsf{L}}, \tau_s)$ and **output** "OK" to DO, DU
13: $state = DataUsage$
14: On receive ("DataUsage", y) from DU: ▷ Honest DU supplies $y = \perp$, only Sim may call with $y \neq \perp$
15: assert ($state = DataUsage$)
16: **if** $y = \perp$:
17: notify $\mathcal{A}$ about ("DataUsage") and block until $\mathcal{A}$ replies
18: determine current time: τ_{cur}
19: assert $\mathbb{T}_{\mathsf{L}} \geq (\tau_{cur} - \tau_s)$
20: **output** $\mathcal{AE}.Enc(Pub_{DU}, \mathcal{C}(\mathcal{D}))$ to DU
21: **else: output** y to DU

Figure 4.8: $\mathcal{F}_{ROT}$: Ideal Functionality of ROT

UsageApproval entry point of $\mathcal{F}_{ROT}$:

In the UC-framework, upon receiving the inputs, the ideal functionality must execute actions that align with the honest execution of the protocol. Consequently, after receiving the service request, SR, $\mathcal{F}_{ROT}$ determines the data requirements $Req(D)$ and the necessary computation $\mathcal{C}$, just as an honest DU would. $\mathcal{F}_{ROT}$ has access to the public $Enc[]$ and it can accurately identify the details of the required computation $desc$, the ID of the necessary enclave, and the enclave itself $Eclv_{\mathcal{C}}$.

Since nothing remains encrypted during the network exchanges of the UsageApproval stage, a network observer may monitor all the messages between DO and DU [129,

Chapter 7.3.1]. Furthermore, the pattern of these network exchanges indicates that DO and DU are in the process of executing UsageApproval. To capture this, $\mathcal{F}_{ROT}$ notifies all this information to the adversary, $\mathcal{A}$.

Since $\mathcal{F}_{ROT}$ operates in an ideal world, preparing a signature on the specified enclave is unnecessary. As a result, $\mathcal{F}_{ROT}$ cannot also notify the signed version of the enclave to the adversary. $\mathcal{A}$ has the authority to block or tamper network communication. As a result, $\mathcal{F}_{ROT}$ must wait for $\mathcal{A}$ to grant permission for the transmission of all network messages before notifying success. $\mathcal{F}_{ROT}$ also records the determined computation, $\mathcal{C}$, along with the identity of the data owner for future reference.

DataProcurement entry point of $\mathcal{F}_{ROT}$:

Any subsequent communication between DO and DU after UsageApproval stage, serves as evidence that both parties are now engaged in the DataProcurement stage. During this stage, DO transmits all information over TLS channel, rendering the content of the messages imperceptible to the adversary. As a result, the ideal functionality only reports the size of the messages [129, Chapter 7.3.2], reflecting the effects of the encrypted communication channel. Provided the adversary does not interfere with this communication, $\mathcal{F}_{ROT}$ will log the current time and retain all relevant information for future reference. It is important to note that, since $\mathcal{F}_{ROT}$ is considered trustworthy, it can reliably track time, without requiring interaction with the blockchain.

DataUsage entry point of $\mathcal{F}_{ROT}$:

After invoking this entry point, $\mathcal{F}_{ROT}$ faithfully determines the current time and ensures that the data has not yet expired. It then performs computations on the stored data and returns the computation result. Even though this call involves only one party, DU, an external adversary still receives a notification about it due to the interaction with the blockchain, which can be observed by an outside entity.

For the sake of simulability, $\mathcal{F}_{ROT}$ may convey the computational result to DU, obtained through the input parameter y. It is crucial to emphasize that this arrangement is solely for the purpose of simulability and does not compromise security in any way. Given that $\mathcal{F}_{ROT}$ cannot be accessed by an adversary, there is no risk of it being misused to transmit arbitrary computation results to DU.

4.5.4 UC-proof

An adversary, $\mathcal{A}$, can undertake various actions to compromise security, the specifics of which are unknown to Sim. However, in the UC framework, Sim executes $\mathcal{A}$ as an internal, opaque subroutine. Consequently, the external interactions between $\mathcal{A}$ and other parties become visible to Sim, which also possesses the ability to modify these interactions. The objective of Sim is to ensure that for every action $\mathcal{A}$ takes to breach security, Sim has an effective countermeasure in place.

Within the UC-framework, we can say that the security is compromized, if the environment, $\mathcal{Z}$ can distinguish between the real-world execution of the protocol and the execution involving Sim, which represents the ideal world. Therefore, we establish both the existence and design of Sim. Furthermore, we demonstrate that the ideal world remains indistinguishable from the real world across the three fundamental design stages previously outlined: UsageApproval, DataProcurement, and DataUsage. Details regarding the design of Sim and the indistinguishability arguments can be found in Appendix G.

4.6 Implementation and Performance Analysis

We have implemented our solution using C and C++. We used the Intel SGX SDK [130] to prepare the enclave-side code. The full implementation of *ROT* can be found here [6]. We use Intel SGX as the underlying TEE-technology. For development and testing, Microsoft Azure's SGX-enabled VM, DC4SV3. Ubuntu 20.04 is used as the operating system. This DC4SV3, instance consists of a quad-core processor (Intel Xeon Platinum @2.8 GHz) and is equipped with 32GB of RAM. We use Ethereum 2.0 as the blockchain and access its state via Infura's JSON-RPC API [131]. However, *ROT* is not reliant on any particular TEE technology or specific blockchain. OpenSSL 1.1.1f is utilised for all cryptographic functions.

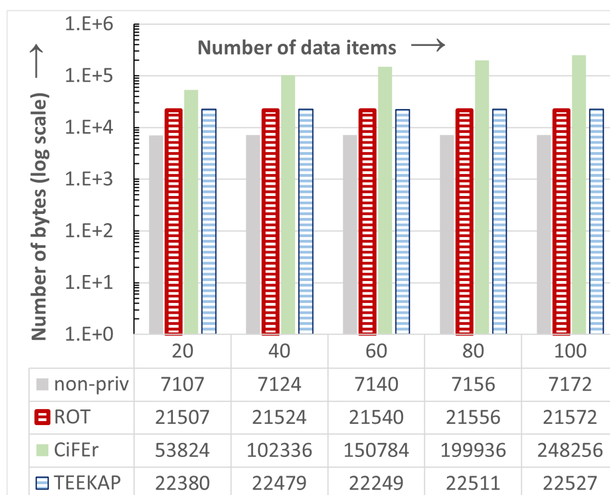
To our knowledge, no existing solution encompasses all the features offered by *ROT*. Nonetheless, we compare our solution with other cryptographic alternatives that offer a partial set of these features. Regarding data privacy, computation on encrypted data (COED) is a newly defined umbrella term [132]. It comprises three cryptographic primitives: homomorphic encryption (HE), multi-party computation (MPC), and functional encryption (FE).

Hence, we selected FE and MPC as comparison candidates because they are partially applicable to web service scenarios for privacy-preserving computation. We chose to ex-

clude HE from the comparison because it operates on ciphertexts and yields results in ciphertext form, making direct comparison challenging. Therefore, we compare *ROT* with a representative MPC-implementation, ABY [133], and an FE-implementation, CiFEr [134]. Despite some differences, TEEKAP [107] is the nearest TEE-based alternative. So, we compared our performance matrices with theirs as well.

We also examine the overhead that *ROT* introduces compared to a standard, non-privacy-preserving setup. To illustrate this, we define a typical secure yet non-privacy-preserving web service, referred to as *non-priv*. In this scenario, after the service user agrees to the privacy policy, they send their personal information to the service provider through a server-authenticated TLS connection. The service provider then verifies the authenticity of the received data and securely stores it in an encrypted format within a database. When the provider needs to perform computations on the data, it retrieves the encrypted data, decrypts it, and performs the necessary computations.

For our experiments, we executed all the operations of all the involved parties (e.g., *DO*, *DU* and $\mathsf{Lib}_{\mathsf{Enc}}$), within the same computing environment. To facilitate their interaction, we utilized a local loopback interface. While this method of communication does not accurately reflect the realistic transfer times found in actual networks, we focused on measuring and comparing the number of bytes transferred. Typically, around 20 to 30 items of personal data are collected and processed during a standard web service [135]. Consequently, we varied the quantity of data attributes from 20 to 100 throughout our experimentation. In our analysis, we assume that, on average, each personal data attribute (e.g., age or passport number) is represented by a 64-bit unsigned integer.



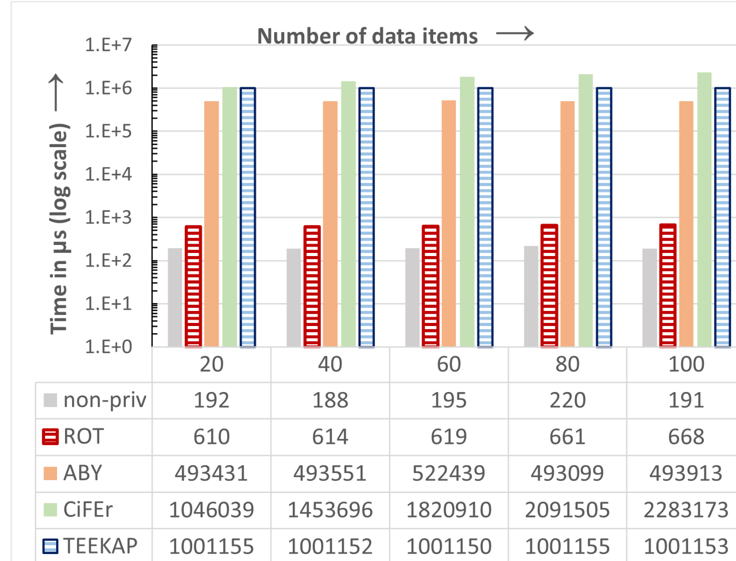
	20	40	60	80	100
non-priv	7107	7124	7140	7156	7172
ROT	21507	21524	21540	21556	21572
CiFEr	53824	102336	150784	199936	248256
TEEKAP	22380	22479	22249	22511	22527

Figure 4.9: Number of required bytes for sending data

To ensure a fair comparison, we executed all the comparison candidates (i.e., CiFEr, ABY, and TEEKAP) on the same computer. We opted for the vector inner product as the performed computation ($\mathcal{C}$) on the private data attributes, as all comparison candidates support this operation. Figures 4.9, 4.10, and 4.11 present the details of our measurements and comparisons.

In *ROT*, *DU* requests *DO*'s private data once and can make use of that data multiple times until the specified expiry time. This one-time transfer of private data involves a constant increase in network communication, approximately 14.5KB compared to a *non-priv* scenario. The primary contributor to this additional size is the remote attestation process between *DO* and $\mathsf{Lib}_{\mathsf{Enc}}$, which relies on TEE technology, and for SGX, it consumes around 7KB. The remaining 7.5KB are required to communicate with the blockchain.

Since it involves remote attestation, TEEKAP also uses a comparable amount of network bandwidth. However, *ROT* has a slight advantage because it does not need to communicate with multiple parties during the data transfer stage. In COED-based solutions, MPC doesn't transfer any data prior to computation; all data exchanges occur during the computation phase. While FE includes a data sending phase, it generates a significantly large ciphertext. Consequently, TEE-based solutions are generally much more efficient. Figure 4.9 shows the detailed comparison.



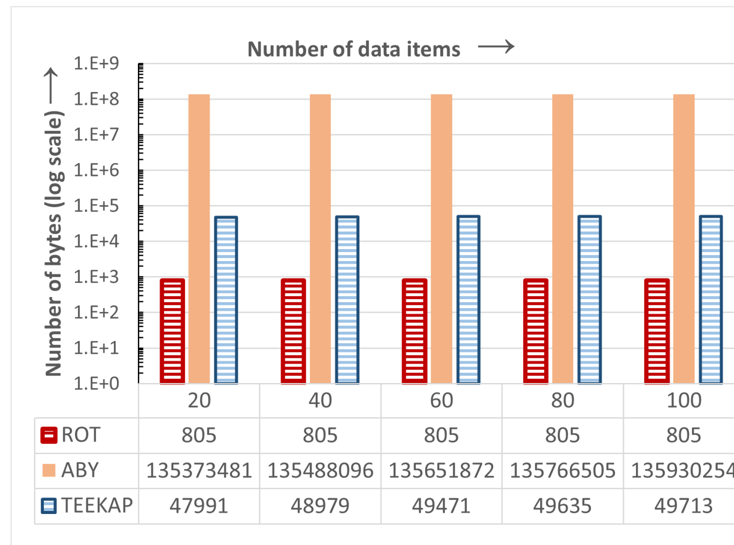
	20	40	60	80	100
non-priv	192	188	195	220	191
ROT	610	614	619	661	668
ABY	493431	493551	522439	493099	493913
CiFEr	1046039	1453696	1820910	2091505	2283173
TEEKAP	1001155	1001152	1001150	1001155	1001153

Figure 4.10: Required time to perform computation

Figure 4.10 presents a comparison of the required time to perform the computation $\mathcal{C}$ on

D. In the case of MPC, not only computation but also actual "data transfer" occur during the computation phase and rely on costly oblivious transfers, which significantly contribute to its slow performance. In contrast, while FE does not require network activity during computation, the underlying mathematical operations are complex and resource-intensive, resulting in significant computation time.

ROT is almost three times slower than $non-priv$. Another study [136] shows that computations carried out within an enclave are only about twice as slow as those performed outside the enclave. However, it is important to note that ROT also requires checking the blockchain state to verify data expiry before executing the computation, which ultimately contributes to ROT's nearly three times processing time compared to *non-priv*.



	20	40	60	80	100
ROT	805	805	805	805	805
ABY	135373481	135488096	135651872	135766505	135930254
TEEKAP	47991	48979	49471	49635	49713

Figure 4.11: Number of bytes transferred while performing computation

In terms of absolute performance values, ROT delivers computational results within 700 μS, even after validating data expiry after consulting blockchain. In contrast, TEEKAP takes over a second to provide computation results. This delay in TEEKAP is due to the necessity for DU's enclave to engage with multiple access committee members through complex protocols to confirm data expiry, leading to significant network transfer requirements (as illustrated in Figure 4.11)), as well as increased time. Overall, when compared to other alternatives, ROT is between 700-3,000$\times$ faster in terms of data processing time and demands between 1.04 - 6,000 $\times$ less network traffic.

Chapter 5

Bidirectional Privacy Preservation During Data-Sharing

When providing services, primary service providers who collect personal data directly from users often find it necessary to share this information with third parties. Such data-sharing scenarios raise privacy concerns not only for users but also for the service providers themselves. In this chapter, we aim to improve privacy in these data-sharing situations.

5.1 Motivation: Bidirectional Privacy and Practicality of Privacy Policy Enforcement

It is not essential for the shared data to remain solely with third parties; these third parties may also disseminate the personal data. Ultimately, this can result in personal data circulating through various layers, creating a tree-like structure. Numerous privacy regulations require service providers to disclose information regarding the individuals or companies with whom they plan to share the collected data. In theory, this transparency allows users to review the privacy policies of all parties involved and make informed choices.

For instance, when organizing a trip through a travel management company, travelers are often required to provide personal information such as age, medical conditions, passport numbers, email addresses, length of stay, room preferences, and dietary needs. The travel management company coordinates all aspects of the journey, including transportation, lodging, meals, and travel insurance, often working in conjunction with various partners

such as transportation providers, hotels, and insurance companies, which serve as secondary service providers.

Moreover, local food companies that provide meals to hotels can act as third-layer service providers, necessitating an understanding of the travellers' dietary preferences. In this context, travellers should begin by reviewing the travel management company's privacy policy to identify the associated hotel, travel, and insurance providers. Next, it is essential to examine the privacy policies of these secondary service providers to uncover any potential third-layer service providers, such as the food company mentioned. Travellers should then review the privacy policies of each identified entity, assess each entity's trustworthiness, and share the necessary personal data accordingly.

However, evaluating an organization's trustworthiness, particularly in terms of the consistency between stated privacy policies and actual practices, presents a significant challenge. Obtaining a certification from a privacy auditor could help establish this trustworthiness. However, the substantial costs involved in the auditing process [137] may discourage organizations - especially smaller ones - from pursuing such certifications on a regular basis. Furthermore, privacy policies may be updated after personal data has already been collected. It can be challenging for service users to receive notifications about policy changes from service providers beyond the first layer. This situation allows service providers to leverage previously-gathered data in accordance with the new policies, without detection.

Consequently, there are several valid concerns regarding privacy when sharing personal information across various platforms. If any of the involved entities fail to comply with established standards and personal data is compromised, the privacy of individuals may be irreparably lost. Therefore, it is crucial to undertake initiatives that lower auditing costs, ensure compliance with privacy policies following audits, and appropriately prevent access to data under any newly-revised privacy terms.

We also argue that current practices surrounding privacy protection in data-sharing scenarios are contributing to new privacy concerns, particularly for service providers. While privacy regulations compel these providers to disclose information about the next layer of data users, they often feel uncomfortable doing so. Their hesitation to divulge details or even acknowledge the existence of their suppliers or partners can be valid, as such information can be considered strategic and may be classified as a trade secret [138, 139]. Additionally, service providers may also seek to protect the identities of their clients from their suppliers while transmitting data.

Nonetheless, this desire can conflict with the suppliers' need to verify the authenticity of the personal data received. In our earlier example, an insurance company might encounter

difficulties in issuing valid insurance, without first confirming the accuracy of the client's medical information, which could unintentionally expose the client's identity. On the other hand, the travel company may wish to achieve anonymity to prevent the insurance company from reaching out directly to the traveler with subsidized offers. Therefore, our objective is to preserve privacy in both directions (i.e. for the client, but also for the provider, as appropriate). We term this bidirectional privacy preservation.

We are also concerned with another practical issue. Ironically, service providers frequently must shoulder most, if not all of the extra complexities and costs, when implementing PETs intended to benefit service users. This fact discourages providers from adopting PETs in practice. Therefore, alongside delivering essential privacy guarantees to the service users, our objective is to present some key advantages for service providers as well, both in terms of financial considerations and the desired level of privacy protection. This could act as a motivating factor for adoption, as they are the ones who are often required to pay for the upgrade.

5.2 Related Work

There seems to be a lack of non-legal research that specifically addresses privacy concerns and trade secret protection, regarding service providers. However, there are several studies that focus on data sharing and auditing.

5.2.1 Ensuring Data Owner's Privacy During Data Sharing

Private data is often shared among various parties, and the exchange of diverse information across sectors such as healthcare and finance can lead to numerous advantages [140, 141]. Additionally, there are situations where sharing collected private data is a necessary requirement [142]. However, it is also crucial for data owners to have the option to opt out of specific data processing practices employed by data users, as well as to provide explicit consent regarding exactly how their personal data can be used.

While initiatives for selective opt-out and opt-in choices are emerging [143], data owners often face an all-or-nothing choice with most non-primary data users. In this context, Iyilade et al. introduced an XML-like policy language known as P2U [144], designed to empower data owners with greater control over their privacy. Hails [145] is a proposed web framework intended for a declarative policy language and an access control mechanism focused on improving the privacy of personal data accessed by third-party applications

on a website. POLICHECK [146] is founded on the principle of automated data-flow verification. This method incorporates the concept of the receiving entity during the assessment of data flow, making it especially effective for ensuring compliance with policies when personal data is shared among multiple parties.

Alansari [94] demonstrated that tracking and control of data-sharing activities can be effectively managed by documenting the details of all such transactions through a trusted entity utilizing tamper-proof media, such as blockchain. In addition to monitoring data-sharing practices, it is crucial to ensure that all data recipients process information in a privacy-preserving manner. Zeinab et al. propose a trust and reputation-based monitoring mechanism [147] that operates based on the data owner's reports of suspected privacy violations. Their solution automatically identifies and penalizes any data users who exploit personal data for purposes for which the data owners have not consented.

Li et al. [148] emphasized the importance of sharing only a necessary subset of personal data with third parties. While this partial sharing enhances the privacy of the data owner, ensuring the authenticity and integrity of personal data continues to be a significant concern for third-party users. To address this issue, they employed a redactable signature scheme. They proposed a cloud framework specifically designed for sharing health data with third parties while effectively removing sensitive information from the dataset.

5.2.2 Reducing Auditing Burden

A notable shortage of security and privacy auditors has been reported [149]. In response, researchers have explored the possibility of automating source code audits. In this context, Kunz et al. [150] employed the code property graph to partially automate the privacy threat modeling process. This approach automatically generates a data flow diagram from the source code, highlighting the privacy properties of data flows, which can then be analyzed semi-automatically through queries.

Tang and Østvold [151] leveraged a static analysis technique to characterize the flow of privacy-related data, assisting developers, auditors, and non-technical individuals such as lawyers in documenting software privacy and data protection behaviors. Zimmeck et al. [152, 153] further leveraged static code analysis reports to automate the assessment of privacy compliance by applying machine learning techniques. Nevertheless, achieving effective artificial intelligence (AI) and machine-learning-based compliance checking remains a significant challenge, particularly in complex codebases.

Researchers also looked at the challenge of achieving alignment between a privacy policy and its corresponding source code implementation from a novel angle: Generating privacy

policies directly from the source code itself. Researchers have utilized AI and machine learning techniques to develop comprehensive privacy policies derived from a codebase [154,155]. Furthermore, technologies such as proof-carrying code (PCC) [156] and automated theorem provers have been investigated for this purpose. However, these approaches constrain developers to specific programming languages when developing their source code, limiting its applicability in practical use case scenarios.

Krahn et al. [157] utilized TEEs to enforce a privacy policy in the realm of third-party storage services. In the broader context of general computation, Birrell et al. introduced the concept of delegated monitoring [96], where an enclave located at the remote data storage's site verifies the identity and credentials of third parties requesting access, before permitting them to access any personal data. However, simply verifying the identity of the recipient may not always be adequate to ensure privacy.

Mazumdar et al. [98] also explored the use of TEEs in this domain, proposing a method to reliably verify the privacy properties of source code directly at the data user's location. As a demonstration, they executed a static analysis tool within an enclave, with the resulting report being verified by another enclave to ensure the preservation of the source code's privacy properties. However, it is crucial to recognize that a code that successfully passes static analysis does not necessarily guarantee privacy in every context. In fact, the automatic assessment of privacy policy alignment within source code remains a significant area of research [158–160].

5.3 *BPPM:* A framework for Bidirectional Privacy preservation with Practicality during Multi-layer data sharing

The service user (or the data owner), acquires services from the primary service provider (or the data user), who in turn depends on secondary sub-service providers. This arrangement can be extended recursively, forming a hierarchical structure of service providers, resembling a tree. In *BPPM*, no node has a complete view of the entire tree. Each node is only aware of its immediate parent and child nodes. Consequently, only the primary service provider knows the identity of the service user.

5.3.1 Overview: Aggregated Policy, Limited Sharing and Reusing of Audited Code

The jth data user at the ith layer is referred to as $DU_{i,j}$. To deliver any requested sub-services, $DU_{i,j}$ may require specific personal data attributes (DA) (such as passport numbers, country of origin, dates of birth, etc.) to be processed in particular ways. $DU_{i,j}$ delineates these data requirements along with the intended processing procedures in a structure known as the **P**ersonal **D**ata processing **S**tatement ($PDS_{i,j}$), which is included in its privacy policy.

The data owner, DO, knows only about the primary data user, $DU_{1,1}$, and is unaware of any others. However, the data processing details for all data users of all the layers must be accessible to DO to enable informed decision-making before any personal data is shared. To facilitate this, each data user, $DU_{i,j}$, is responsible for collecting PDS structures from the privacy policy documents of its child nodes. $DU_{i,j}$ then adds its own data processing statements and outlines the consolidated information in $PDS_{i,j}$, which can be accessed by $DU_{i,j}$'s parent. This procedure is recursive, and $DU_{1,1}$ ends up compiling the complete details of the entire data usage tree into $PDS_{1,1}$. DO reviews this $PDS_{1,1}$ from $DU_{1,1}$'s privacy policy document and grants consent for some or all of the data processing statements (i.e., $PDS_{DO} \subseteq PDS_{1,1}$).

Subsequently, DO compiles the requested personal data attributes into the structure DS_{DO}. Both PDS_{DO} and DS_{DO} are transmitted to $DU_{1,1}$. To ensure authenticity and confidentiality, DO signs and encrypts both PDS_{DO} and DS_{DO} prior to transmission. Upon receipt, $DU_{1,1}$ processes DS_{DO} in its encrypted form. If necessary, relevant subsets may be forwarded to child nodes. This forwarding process can occur recursively, allowing DO's personal data to traverse the tree structure.

Importantly, our framework ensures that personal data is never exposed in plaintext. Instead, we utilize TEEs to guarantee the confidentiality of the personal data. Furthermore, $BPPM$ restricts computation on this data to a secure code component that is solely signed by auditors. Additionally, our framework fosters the development and maintenance of a software library ecosystem that comprises pre-audited source code components. This approach promotes the reuse of code components, ultimately reducing both development and auditing costs.

The notation of our scheme is summarized in Table A.4.

5.3.2 Data Sharing Stakeholders and Flow of Personal Data

$DU_{i,j}$ starts by setting up an enclave (Stage 1 in Figure 5.1) that executes a predefined secure base code ($Prog^{BPPM}_{Eclv}$) on a platform equipped with a TEE. Following this, $DU_{i,j}$ loads additional code components that have been pre-audited and are free from privacy leaks. These code components correspond to the personal data-processing statements outlined in $PDS_{i,j}$ and are obtained from a trusted Code Provider, CP, during Stage 1.1.

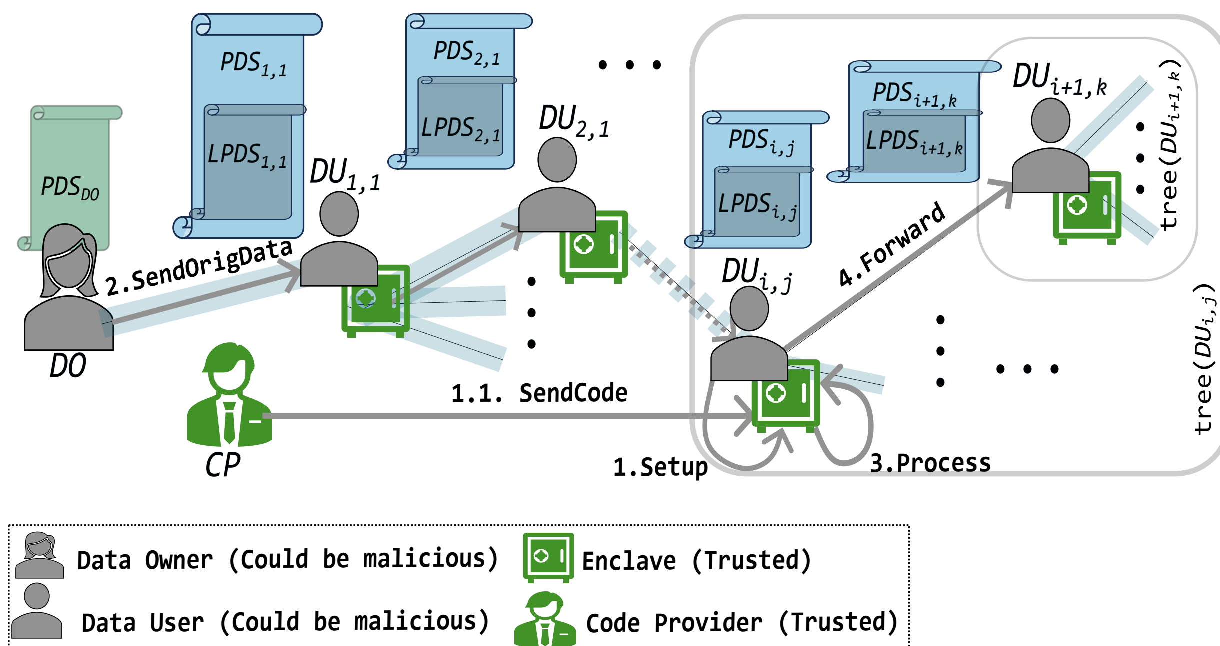


Figure 5.1: Top-level overview of *BPPM*

The primary data user, $DU_{1,1}$, finishes its setup process (Stage 1 for $DU_{i=1,j=1}$) and consolidates the PDS structure from the entire data usage tree into $PDS_{1,1}$. After this, DO is able to examine $PDS_{1,1}$ as outlined in the privacy policy document of $DU_{1,1}$. DO can then decide to deny certain proposed data processing statements while agreeing to the selection, PDS_{DO}. Following this, the DO securely transmits PDS_{DO}, along with the pertinent personal data, DS_{DO}, to $DU_{1,1}$'s enclave (Stage 2 in figure 5.1).

Although $PDS_{1,1}$ contains data processing statements for the entire tree, only a subset of it, referred to as $LPDS_{1,1}$, is executed by $DU_{1,1}$, while the child nodes handle the remaining statements. After receiving personal data and processing consent in Stage 2,

$DU_{1,1}$ may, when necessary, request its own enclave to compute results based on a specific processing statement S, where $S \in (PDS_{DO} \bigcap LPDS_{1,1})$ (this occurs in Stage 3 for $DU_{i=1,j=1}$).

Whenever needed, $DU_{1,1}$ asks its enclave to send the necessary subset, $PDS_{frw} \subseteq (PDS_{DO} \bigcap PDS_{2,k})$, to $DU_{2,k} \in$ **children**$(DU_{1,1})$, along with the relevant subset of DS_{DO} (Stage 4 for $DU_{i=1,j=1}$). Here, **children**$(DU_{1,1})$ represents the set of child nodes associated with $DU_{1,1}$. Once received, $DU_{2,k}$ may then process the personal data locally or pass a subset on to its own descendants. This way, the personal data from the DO can propagate to any random node in the data usage tree, $DU_{i,j}$.

One crucial aspect of this sharing process is that DO expresses consent by digitally signing the document PDS_{DO}. However, when passing the information on to subsequent layers of data users, it becomes essential to not only pass a subset of DS_{DO} but also to eliminate any superfluous data-processing statements from the signed PDS_{DO}. This removal would invalidate the signature, complicating the verification of the reduced list's authenticity. To tackle this challenge, we leverage the redactable-signature scheme (RSS) [161] within the *BPPM* framework.

5.4 *BPPM*:Details

We now outline the comprehensive design of *BPPM*. First, we define the threat model, followed by an explanation of what is meant by the *Guarantee of Privacy Preservation*. Next, we demonstrate how to establish a piracy-free code-sharing ecosystem. We then describe the necessary data structures utilized in *BPPM*, along with the specifics of our protocol.

5.4.1 Threat model and Assumption

The threat model remains consistent with our previous framework, *ROT*. In the context of *BPPM*, this means that both data owners and data users - regardless of the layer - may exhibit malicious behavior. The only requirement for data users in *BPPM* at any layer is to have a TEE-enabled platform. We assume that CP acts honestly and maintains confidentiality about the communication details with $DU_{i,j}$. Like *ROT*, *BPPM* also utilizes typical (and insecure) public communication channels. Consequently, *BPPM* addresses the resulting concerns by employing required strategies (e.g., using an IND-CPA-secure encryption scheme) within the protocol execution.

In contrast to *ROT*, *BPPM* necessitates that the *DO* holds a long-term, PKI-verifiable public key certificate, denoted as $Cert_{DO}$. It is assumed that the *DO* does not share the corresponding secret key, sk_{DO}, with anyone. While auditing source code for security and privacy vulnerabilities is not included in our framework, *BPPM* does ensure the verification of the auditor's signature on all installed code components within the enclave. We assume that the CP will have access to all the audited code ($C[]$) before participating in *BPPM*.

In *BPPM*, we also assume that TEEs reliably maintain their security attributes. However, the adversary is allowed to control (modify or view) all aspects outside the enclave, including the operating system, communication channels, and RAM contents. They may also supply malicious inputs when accessing the enclave's entry points and could attempt to disable or circumvent the TEE. Our protocol addresses those concerns.

5.4.2 Guarantee of Privacy Preservation

Our protocol ensures that all data users - at any level - participating in *BPPM* provide the *Guarantee of Privacy Preservation*. This guarantee encompasses three key aspects. First, the data owner's personal data is never exposed in plaintext; untrusted data users can only access the output of computations performed on that data. This is achieved by processing personal data within a secure enclave, which accepts encrypted personal data as input and generates plaintext output. Second, no additional computations on personal data are permitted beyond those explicitly authorized by the data owner. Our protocol ensures that the enclave verifies the data owner's explicit permission before performing the requested computation. Finally, the output of these computations does not compromise privacy, as it is ensured that only audited code executes on the data owner's personal data.

5.4.3 Piracy-free Source-code Sharing

The privacy of the data owner may be at risk if the code running within the enclave inadvertently or intentionally leaks sensitive information. Moreover, the code executed in the enclave must adhere to the defined data processing statement, available in the privacy policy document. For instance, if this statement indicates that the date of birth is solely used to verify age, the code should not repurpose this information for any other use, such as sending promotional offers.

These considerations must be evaluated during the auditing process (e.g., SOC2 audits) and confirmed by a trusted auditor. However, this process can often be both time-consuming and costly [137]. Furthermore, unless exceptional circumstances arise, only

standard processing is required for personal data (for instance, using date of birth for age verification). Therefore, whenever feasible, it may be beneficial to reuse pre-audited code.

Therefore, we propose the presence of a Code Provider, who develops source code components that facilitate various standard operations on diverse personal data attributes and gets them audited by a trusted auditor. The code provider maintains a database of pre-audited code which can be purchased by data users as needed. However, there exists a risk that a malicious data user, upon acquiring code components from the code provider, may engage in the theft of intellectual property or secretly distribute them to other entities. This poses a significant deterrent for the code provider, who has invested time and resources in creating and auditing these code components. Such concerns could undermine the motivation to foster a code reuse ecosystem. Therefore, implementing piracy protection is essential.

BPPM utilizes the availability of TEE at each data user's location to safeguard against piracy. Instead of transmitting the plaintext code components, the code components are encrypted with a public key that is unique to the recipient's enclave. This ensures that only the designated and trusted enclave, which possesses the corresponding secret key, can decrypt and execute the code.

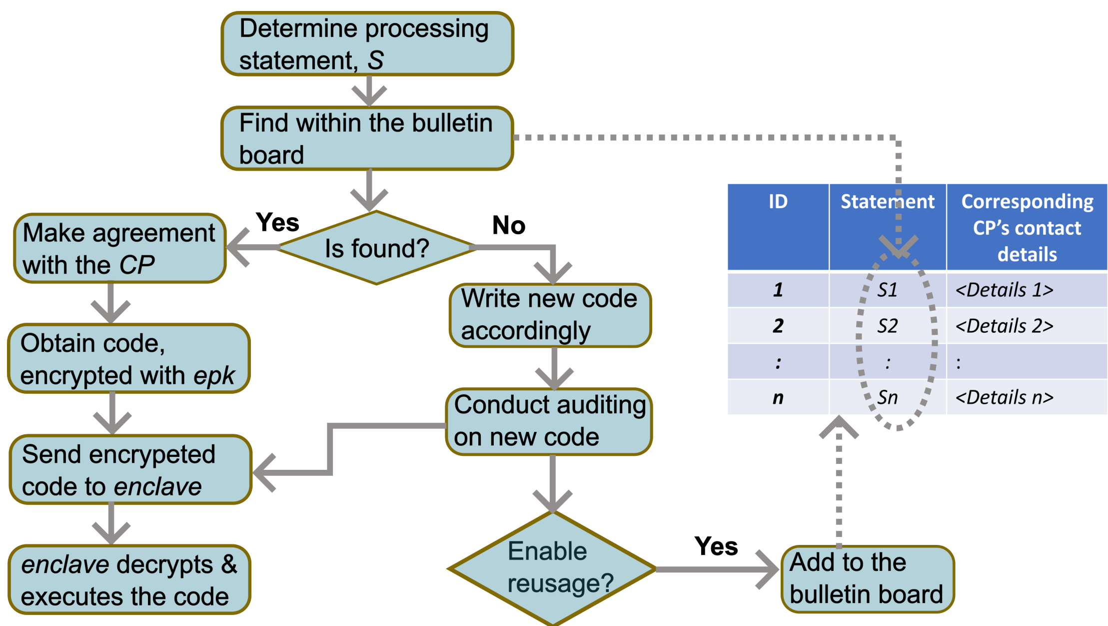


Figure 5.2: Code reusage scenario

However, a pre-existing code may not always be available for every required data usage statement ($S \in LPDS_{i,j}$). In such instances, $DU_{i,j}$ would need to develop the unavailable code components and have them audited before they can be used. In this scenario, $DU_{i,j}$ functions as its own code provider. While this approach may entail higher costs, $DU_{i,j}$ can announce the availability of the newly developed code components through a public bulletin board, thereby becoming a code provider. Other data users seeking similar functionality can then purchase this audited code, allowing $DU_{i,j}$ to recoup some of the expenses related to development and auditing.

Various parties, including different data users and specialized software developers, may take on the role of a code provider. The overall code-reuse scenario is depicted in Figure 5.2. When $DU_{i,j}$ develops specific code components and makes them accessible to other data users, it can potentially yield financial advantages for $DU_{i,j}$. However, it's important to note that this is not a mandatory aspect of *BPPM*. In fact, a data user might choose to limit the utilization of its developed code, even in encrypted format, particularly if that code provides them with a strategic or commercial advantage.

5.4.4 Structures of Personal Data and Processing Consent

Iyilade et al. developed the "Purpose-to-Use" (P2U) privacy policy specification language for secondary data sharing [144]. Building upon this foundation, we have created two straightforward XML-like data structures (see Figure 5.3) that facilitate the transfer of personal data along with the associated data processing consent within *BPPM*. In contrast to P2U, our data structures provide data owners' anonymity, while still offering privacy protections for both data owners and data users. Additionally, our structures guarantee the authenticity of the personal data and the consent provided by the data owner.

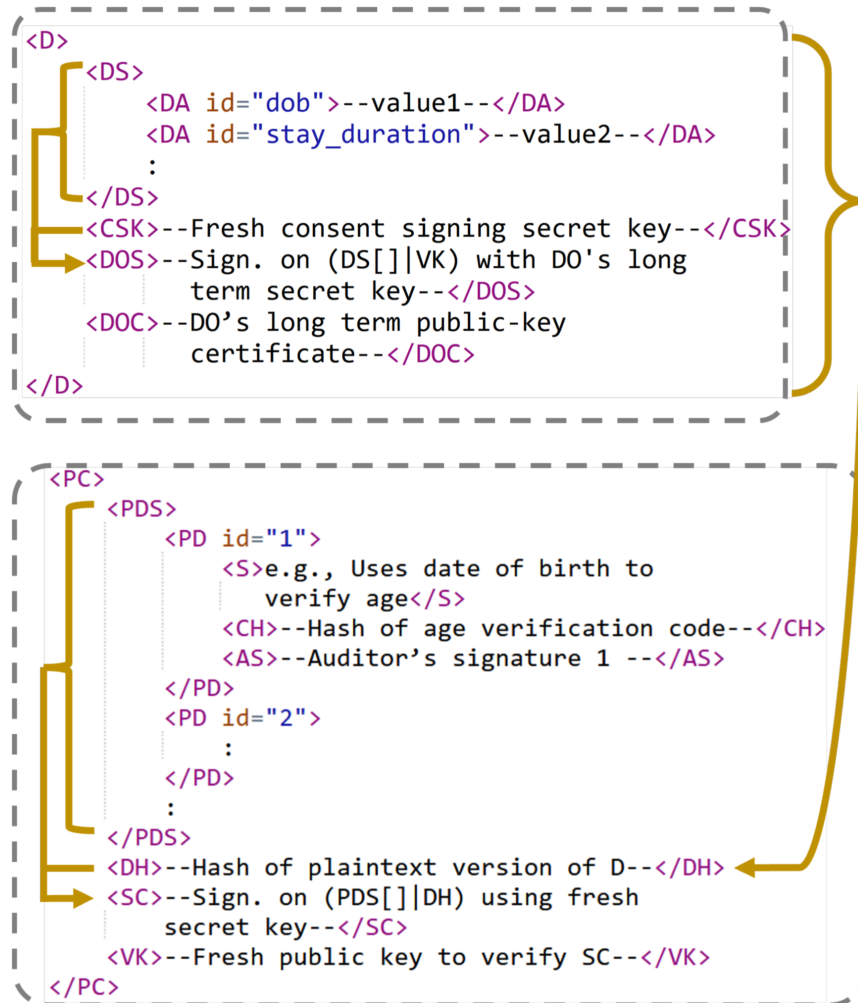


Figure 5.3: Data-structures PC and D and their relationship

Personal Data structure (D)

The data owner provides its personal data through the personal **D**ata structure, D. The data owner transmits this in encrypted format. Only the enclave has the capability to decrypt and access its details. The array $D.DS[]$ contains the actual **D**ata **S**et, consisting of various personal **D**ata **A**ttributes (DA). To facilitate the processing of consent, the data owner generates a new key pair, denoted as (pk_{con}, sk_{con}).

The secret key sk_{con} is employed to sign the agreed-upon list of data processing state-

ments, and this **C**onsent **S**igning secret **K**ey is stored in $D.CSK$. During the forwarding stage of *BPPM*, a trusted enclave can utilize this secret key to generate a signature on a reduced version of the original consent.

The ephemeral key pair, (pk_{con}, sk_{con}), by itself cannot ensure authenticity. Consequently, the data owner also possesses a long-term key pair, (pk_{DO}, sk_{DO}), and signs the public part of the ephemeral key-pair using sk_{DO}. A PKI-verifiable public key certificate, referred to as $Cert_{DO}$ (which is stored in $D.DOC$), authenticates pk_{con}.

Since $Cert_{DO}$ is not shared outside the enclave, the identity of the data owner, which is also included within $Cert_{DO}$, remains confidential. $D.DOS$ contains the **D**ata **O**wner's **S**ignature, created with sk_{DO}, on the combination of $D.DS[]$ and pk_{con}. The enclave, with access to $D.DOC$, can verify the authenticity of $D.DOS$.

Personal data Processing Consent (PC)

The complete PC-structure encompasses the details of the DO's personal data **P**rocessing **C**onsent. Data users can access this structure in plaintext format, outside of their enclave environment. PC includes the DO's agreed-upon list of processing statements, referred to as $PDS[]$.

This is described in greater detail in the following section, along with other relevant elements. $PC.DH$ contains the plaintext **D**'s **H**ash. The **S**ignature on the **C**onsent, $PC.SC$, records the signature on the concatenation of $PDS[]$ and $PC.DH$. This signature is generated using $D.CSK$ and can be verified with the consent **V**erification **K**ey (pk_{con}), as outlined in $PC.VK$.

Personal data Processing Details Set (PDS)

A sub-structure of PC is the personal data **P**rocessing **D**etails **S**et. PDS includes a list of **P**rocessing **D**etails (PD). Each element of PD-list is divided into three parts: $PD.S$, $PD.CH$ and $PD.AS$. $PD.S$ stores the data processing **S**tatement in a human-readable format and outlines the details regarding the required personal data attributes from the service user, the purpose for collecting them, how they will be processed, and the consequences if the service user chooses not to provide consent. For example, if the service user opts out of supplying its date of birth, they will not be able to claim the senior citizens' discount. Data owners review these $PD.S$ statements prior to making decisions about data sharing.

$PD.CH$ stores the hash value of the corresponding audited source code, and the **A**uditor's **S**ignature is kept in $PD.AS$. During the code signing process, the privacy auditor combines $PD.CH$ and $PD.S$. This binds a code with the specific stated purpose.

5.4.5 Details of Protocol Stages

The $BPPM$ protocol consists of four main stages: **Setup**, **Send Original Data**, **Process**, and **Forward**. The relationship among these stages is illustrated in Figure 5.1. The following sections provide a detailed description of each stage.

Setup Stage

All data users of the data usage tree must complete this stage before processing any service requests.

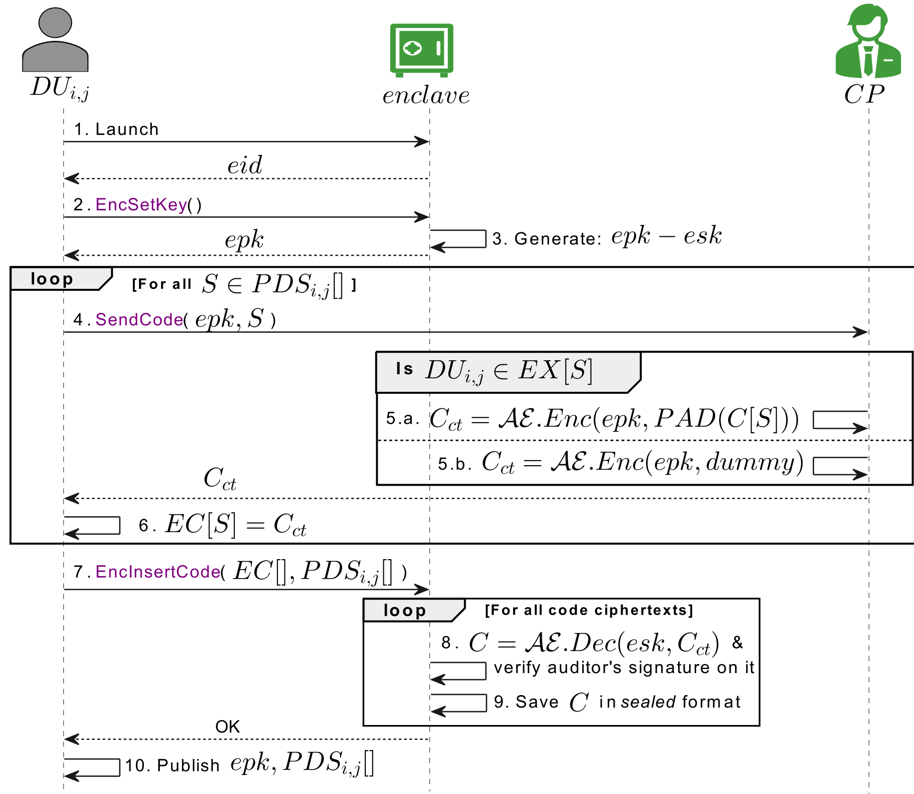


Figure 5.4: Detailed sequence of "Setup" stage

$DU_{i,j}$ **first launches and initializes a new enclave (see Figure 5.4: steps 1 through 3):** $DU_{i,j}$ initiates a new enclave to execute the predefined trusted base code, $Prog_{Eclv}^{BPPM}$ (refer to Figure 5.9). Subsequently, $DU_{i,j}$ requests the newly created enclave to generate an asymmetric key pair, $(epk - esk)$. The enclave discloses the public key epk, while keeping the secret key esk confidential. This public-key allows both the data owner and the code provider to securely transmit messages to that specific enclave. Before utilizing epk, the remote party verifies the attestation of epk to ensure its authenticity.

Next, $DU_{i,j}$ **obtains the audited code ciphertexts from** CP **(steps 4 through 6):** An external network observer may monitor the interactions between $DU_{i,j}$ and CP to ascertain the number of code components received from the CP. This information could reveal out of $PDS_{i,j}$ how much processing is to be conducted locally by $DU_{i,j}$. To alleviate this risk, $DU_{i,j}$ utilizes dummy requests during its communication with the CP, effectively concealing the true number of requesting code components. CP then sends back encrypted versions of all licensed code to $DU_{i,j}$, accompanied by additional dummy ciphertext. Furthermore, the CP ensures that both the licensed and dummy are padded to maintain uniform length, making it impossible to distinguish between them.

$DU_{i,j}$ **installs the received code ciphertexts into its enclave (steps 7 through 10):** Since $DU_{i,j}$ may be dishonest, it could potentially attempt to install malicious source code within its enclave, which could lead to the leakage of private information during execution. Therefore, upon receiving the code ciphertexts, the base code ($Prog_{Eclv}^{BPPM}$) of the enclave first decrypts the ciphertexts and then verifies the corresponding code hash and auditor signature. Given the limited size of enclaves, the obtained code components are stored outside of the enclave for future use, but they are kept in a sealed format. This sealed format ensures that only the specific enclave will be able to load and execute that sealed code at a later time.

SendOrgData Stage

This stage (see Figure 5.5) occurs when DO submits its personal data along with approved processing consent to the enclave of $DU_{1,1}$. The enclave verifies the authenticity of this submission and subsequently reveals the plaintext version of the consent structure (PC) to $DU_{1,1}$, while keeping the data (D) confidential.

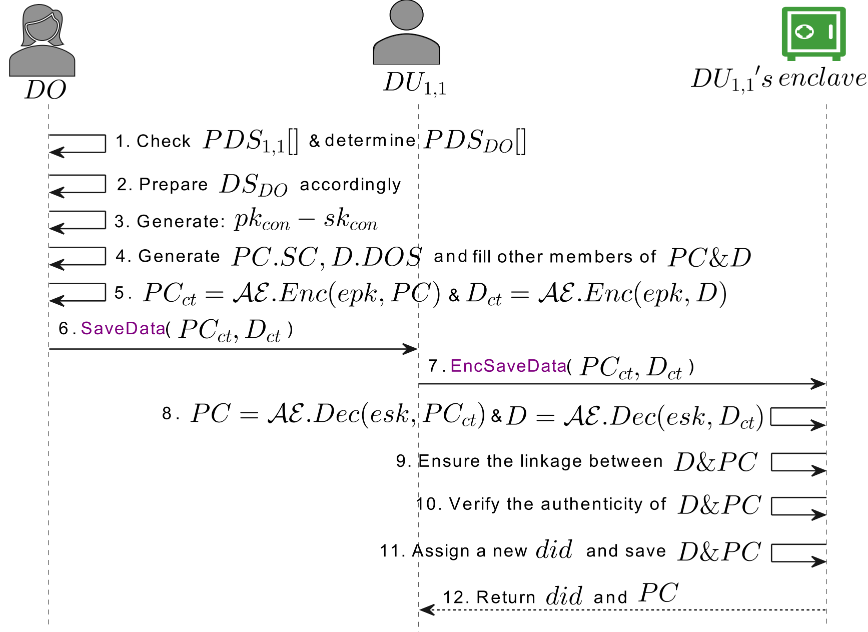


Figure 5.5: Detailed sequence of "Send Original Data" stage

DO sends D & PC to $DU_{1,1}$'s enclave (Figure 5.5, steps 1 through 6): To begin, DO agrees to a subset of data-processing statements proposed by $DU_{1,1}$ and subsequently populates $PC.PDS[]$ and $D.DS[]$ accordingly. To ensure authenticity, DO also digitally signs the agreed-upon processing statements. However, DO cannot simply use its long-term secret key, sk_{DO}, to generate the signature. Doing so would require sharing the sk_{DO} to create a signature for the reduced version of the consent, which is not advisable for security reasons. Furthermore, the PC structure is visible outside the enclave environment, making it possible for an inquisitive data user to trace the data owner's identity through the signature.

Therefore, DO generates a new key pair, $pk_{con} - sk_{con}$, for producing the signer anonymous signature, $PC.SC$. Since this key pair is freshly created, data users are unable to ascertain the identity of the data owner by observing $PC.SC$. However, DO certifies pk_{con} and D with an additional signature, $D.DOS$, using its long-term secret key, sk_{DO}.

This does not pose a concern, as only the trusted enclave can access and verify this signature. To facilitate the redaction process, DO stores sk_{con} in $D.CSK$, which can only be utilized by the trusted enclave to generate a new consent signature for a subset of $PC.PDS[]$. Subsequently, DO fills out the remaining fields of D and PC, and sends them to $DU_{1,1}$ after encrypting them with the enclave's public key, epk.

$DU_{1,1}$'s enclave verifies the received information (Steps 7 through 10): Upon receiving the consent and data ciphertexts (PC_{ct} and D_{ct}), $DU_{1,1}$ forwards them to its enclave. If $DU_{1,1}$ is malicious at this stage, it may engage in a variety of malicious actions. These could include replaying personal data transmitted by one data owner with the consent of another, or reusing an earlier version of personal data submitted by the same data owner with new consent, etc. Therefore, after decrypting the inputs, the enclave verifies the link between PC and D, by inspecting $PC.SC$ and $PC.DH$. Since DO never discloses sk_{DO}, $DU_{1,1}$ is unable to execute any forgery attacks.

On the other hand, a malicious data owner might attempt to exploit the service by replaying the encrypted version of another data owner's D alongside its own PC structure. The enclave safeguards against this scenario by inspecting $D.DOC$, $D.DOS$, $PC.VK$, $PC.SC$, and their interconnections to ensure that such an occurrence has not taken place.

The enclave securely stores them and only reveals the plaintext PC (Steps 11 and 12): Upon verification, the enclave securely stores both PC and D in a sealed format and subsequently discloses the plaintext PC to $DU_{1,1}$. If DO has not provided consent for all data processing activities, certain services may be inaccessible. The plaintext PC allows $DU_{1,1}$ to clearly comprehend the consents granted by DO, enabling it to determine which services can be offered based on those permissions. Ultimately, the enclave associates both D and PC with a newly generated data identifier, *did*. This allows $DU_{1,1}$ to utilize that *did* in the future to request processing related to this specific data-consent pair.

Processing Stage

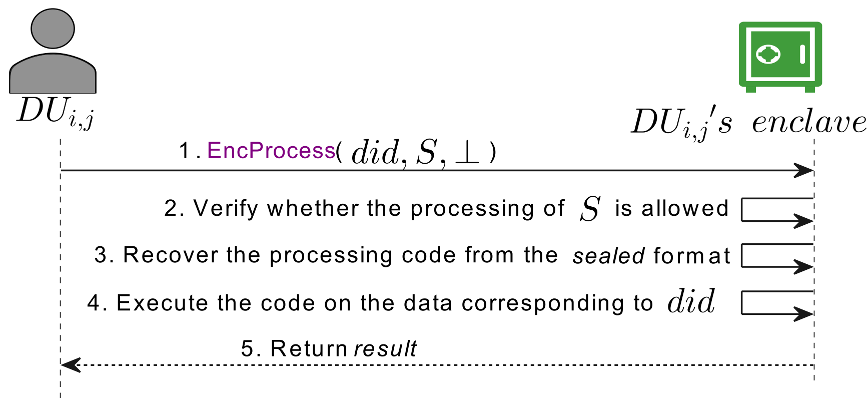


Figure 5.6: Detailed sequence of "Process" stage

$DU_{i,j}$ receives the data-consent pair either from the DO (if $i = j = 1$) or from **parent**$(DU_{i,j})$. Subsequently, in this data processing stage (Figure 5.6), $DU_{i,j}$'s enclave conducts the requested computation on the personal data and returns the computation result.

$DU_{i,j}$'s enclave performs computation and returns the result (Figure 5.6, Steps 1 through 5): Since only the trusted enclave has access to personal data, the data user submits a data processing request to its enclave. $DU_{i,j}$ submits a specific data-processing statement, S, along with the *did* index to identify a particular data-consent pair. However, it is possible that even if S is included in $LPDS_{i,j}$, the data sender (either DO or **parent**$(DU_{i,j})$) has not granted permission for S.

Therefore, prior to processing, the enclave verifies whether S is included within the consent ($PC.PDS[]$). If present, the enclave of $DU_{i,j}$ identifies the sealed code component associated with S and loads that within the enclave. The enclave then executes the loaded code component on the specified personal data and returns the plaintext computation result to $DU_{i,j}$.

BPPM can be optionally enhanced to yield differentially private computation results [61]. To implement this, the enclave may introduce noise prior to outputting the result. Furthermore, the enclave actively monitors the privacy budget using the specific internal state associated with the *did*. Once the privacy budget is depleted, the enclave will refrain from disclosing the result. However, in order to ensure a differentially private solution, *BPPM* must reliably maintain the privacy budget and provide protection against rollback attacks. This can be accomplished by adhering to our previous solution, *ROT*. The enclave is required to verify the freshness of the privacy budget, by consulting the blockchain before executing the computation.

Forwarding Stage

In this stage, $DU_{i,j}$ forwards the received personal data to one or more of its *child* nodes for further processing (see Figure 5.7). Prior to forwarding, $DU_{i,j}$ may restrict the data-processing capability of the child node. $DU_{i,j}$ can re-enter this stage multiple times to send the same personal data (potentially with varying processing capabilities) to different child nodes.

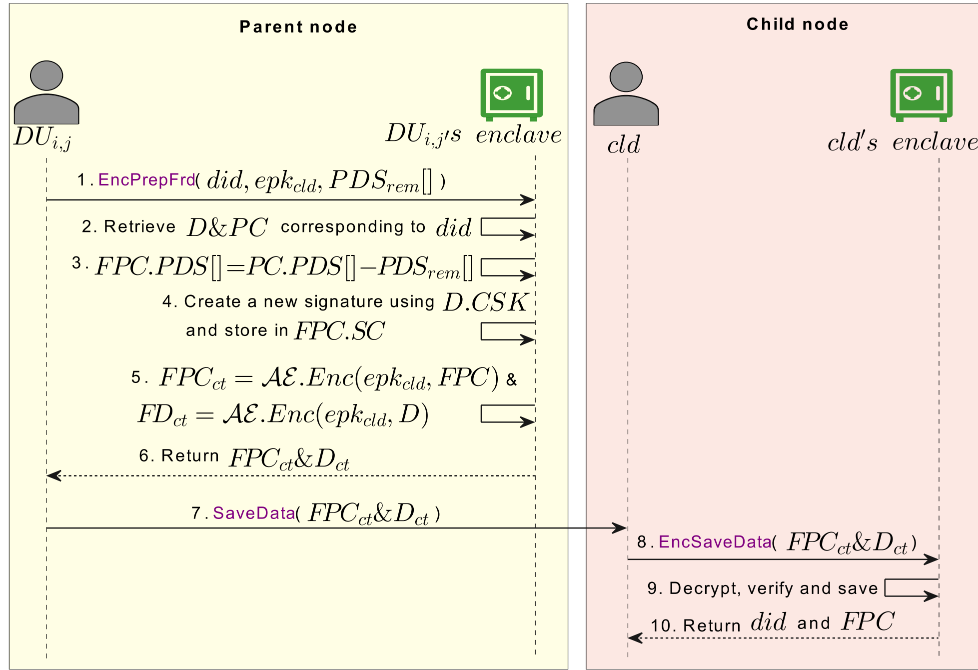


Figure 5.7: Detailed sequence of "Forward" stage

$DU_{i,j}$'s enclave prepares the personal data and reduced consent (Figure 5.7 Steps 1 to 6):
Since $DU_{i,j}$ does not have direct access to personal data and cannot produce a new signature for the reduced version of the consent, it sends a request to its enclave for assistance. The enclave then removes the specified $PDS_{rem}[]$ from $PC.PDS[]$ and creates the corresponding consent structure (FPC). The enclave also generates the signature $FPC.SC$ using the secret key $D.CSK$. Since only DO and the trusted enclave have access to this signing key, it is impossible to forge this new signature. Subsequently, $DU_{i,j}$'s enclave encrypts D and FPC using the public key of the child's enclave, before outputting them to $DU_{i,j}$ for forwarding those ciphertexts to the child. $DU_{i,j}$'s enclave learns the public-key of the child's enclave from the information published by the child node (during the "Setup" stage of the child node. Refer to 5.4).

It is important to note that $D.DS[]$ remains unchanged, as the minor bandwidth savings do not warrant the increased protocol complexity, particularly since $D.DS[]$ is typically small in most applications. The child's enclave will only be able to process the subset of $D.DS[]$ permitted by $FPC.PDS[]$. Therefore, even if there are additional data attributes within $D.DS[]$, they will remain inaccessible to the child node.

$DU_{i,j}$**'s enclave verifies and stores the received data-consent pair (Steps 7 to 10):**
$DU_{i,j}$ transmits the ciphertexts to the child node. Upon receipt, the child node forwards these ciphertexts to its enclave. After this, the enclave's operations mirror those performed during the *Send Original Data* stage. Specifically, the enclave decrypts the ciphertexts, verifies their authenticity, ensures their correlation, and securely stores them in a sealed format for future use. Finally, the enclave reveals plaintext of the received plaintext, FPC.

5.5 Privacy Analysis

Privacy analysis of *BPPM* proceeds in stages. First, we formally represent our protocol and specify its privacy and security goals. After this, the expected ideal functionality for *BPPM* is defined based on these privacy and security goals. Then, the defined ideal functionality is used to prove the privacy and security properties under the UC framework.

5.5.1 Formal Representation of *BPPM*

We formally represent *BPPM* in a manner similar to the formal representation of *ROT*. Figure 5.8 displays the formal representation of the *BPPM* protocol ($Prot_{BPPM}$). In *BPPM*, each enclave is initialized with a fixed base program, $Prog_{Eclv}^{BPPM}$, which serves as the initial trusted computing base (TCB). Figure 5.9 provides a formal description of that base enclave program. Similar to $Prog_{Eclv}^{ROT}$, to facilitate equivocation, we have introduced a backdoor at the "EncProcess" entry point of $Prog_{Eclv}^{BPPM}$ (line 31). Once installed, the functionality of the enclave can be expanded with trusted and audited code components provided by the code provider. However, the installation of new code components does not introduce any additional entry points to $Prog_{Eclv}^{BPPM}$.

In $Prot_{BPPM}$, the DO, all data users ($\forall_{i,j}\ \ DU_{i,j}$), and the CP each have distinct roles. CP's SendCode entry point and the SaveData entry point of $DU_{i,j}$ function as internal machines in $Prot_{BPPM}$ and are not exposed to $\mathcal{Z}$. *BPPM* employs two arrays, $C[]$ and $EX[]$, both indexed by the personal data processing statements, S. Access to these arrays is limited to the CP and the ideal functionality, $\mathcal{F}_{BPPM}$ (refer to Section 5.5.3 for more details).

$C[S]$ contains the plaintext source code associated with the data processing statement, S. There are costs involved in generating $C[S]$ and auditing it, which may require data users to compensate the CP for using $C[S]$. The $EX[]$ array keeps track of data users who

have been granted permission to execute the code $C[S]$. In other words, for a data user $DU_{i,j}$ to access the code, $C[S]$, they must have a prior agreement with the CP. When $DU_{i,j}$ makes a code request, CP first checks their authorization by referring to $EX[S]$ before providing the code ciphertext.

A pre-image resistant hash function H is utilized in $BPPM$ to generate digests for plaintext code components as well as for D. Similar to ROT, $BPPM$ employs two EUF-CMA-secure digital signature schemes. The scheme $\Sigma(KGen, Sig, Vf)$ is used to verify the authenticity of $D.DOS$, $PD.AS$, and $PC.SC$, while $\Sigma_{TEE}(KGen, Sig, Vf)$ is employed to attest to the legitimacy of the output from the TEE. To facilitate secure communication with the enclave, an IND-CPA-secure asymmetric encryption scheme $\mathcal{AE}(KGen, Enc, Dec)$ is utilized.

$Prot_{BPPM}[\Sigma, \Sigma_{TEE}, \mathcal{AE}, H, C[], EX[], DO, DU_{i,j}, CP]$

$\underline{DU_{i,j}}$**:**

1: On receive ("SetUp", $PDS_{i,j}[]$) from $\mathcal{Z}$:
2: $EC[] := \emptyset$ ▷ For storing encrypted code components, indexed by S
3: $CList[] := \emptyset$ ▷ For storing consent, indexed by did
4: send("install", $Prog^{BPPM}_{Eclv}$) to G_{att} wait for eid
5: send("resume", eid, ("EncSetKey")) to G_{att} wait for $(epk, \sigma_{TEE}(epk))$
6: $\forall_{PD}$ in $PDS_{i,j}[]$: ▷ Acquire code ciphertexts for all the processing statements
7: send("SendCode", $PD.S, eid, epk, \sigma_{TEE}(epk)$) to CP wait for c_{ct}
8: $EC[PD.S] := c_{ct}$
9: send("resume", eid, ("EncInsertCode", $EC[], PDS_{i,j}[]$)) to G_{att}
10: publish($eid, epk, \sigma_{TEE}(epk), PDS_{i,j}[]$)) ▷ For others to securely interact with $DU_{i,j}$'s enclave
11: On receive ("SaveData", D_{ct}, PC_{ct}) from $S \in \{DO, \textbf{parent}(DU_{i,j})\}$: ▷ Acts as an internal machine
12: send("resume", eid, ("EncSaveData", $D_{ct}, PC_{ct}, \perp, \perp$)) to G_{att} wait for (did, PC) ▷ Also returns their attestations, not shown for brevity
13: $CList[did] := PC$ ▷ Save plaintext consent
14: On receive ("Process", did, S) from $\mathcal{Z}$:
15: send("resume", eid, ("EncProcess", did, S)) to G_{att} wait for $result$
16: **output** $result$
17: On receive ("ForwardData", $did, PDS_{rem}[], cld, epk_{cld}, \sigma_{TEE}(epk_{cld})$) from $\mathcal{Z}$:
18: send("resume",eid,("EncPrepFrd",$did, PDS_{rem}[], epk_{cld}, \sigma_{TEE}(epk_{cld})$))to G_{att} wait for D_{ct}, FPC_{ct}
19: send("SaveData", D_{ct}, FPC_{ct}) to cld

$\underline{CP}$**:**

20: On initialize ($C[], EX[]$): store $C[], EX[]$ for future use ▷ Plaintext code & execution permissions
21: On receive ("SendCode", $S, eid, epk, \sigma_{TEE}(epk)$) from $DU_{i,j}$: ▷ Acts as an internal machine
22: assert $\Sigma_{TEE}.Vf(\sigma_{TEE}(epk))$
23: **if** $DU_{i,j} \in EX[S]$ **then:** send($\mathcal{AE}.Enc(epk, PAD(C[S]))$) to $DU_{i,j}$
24: **else:** send($\mathcal{AE}.Enc(epk, PAD(dummy))$) to $DU_{i,j}$

$\underline{DO}$**:**

25: On receive ("SendOrgData", $PDS_{DO}[], DS_{DO}[], sk_{DO}, Cert_{DO}$) from $\mathcal{Z}$:
26: wait for $(eid, epk, \sigma_{TEE}(epk))$ from $DU_{1,1}$
27: assert $\Sigma_{TEE}.Vf(\sigma_{TEE}(epk))$
28: $PC.PDS[] := PDS_{DO}[], D.DS[] := DS_{DO}[]$ ▷ Set the approved subsets
29: $(pk_{con} - sk_{con}) \leftarrow \Sigma.KeyGen(1^\lambda)$ ▷ Generate fresh key-pair for managing the consent
30: $D.CSK := sk_{con}, D.DOC := Cert_{DO}, PC.VK := pk_{con}$
31: $D.DOS := \Sigma.Sig(sk_{DO}, (D.DS[]|PC.VK))$
32: $PC.DH := H(D)$ ▷ Save hash
33: $PC.SC := \Sigma.Sig(sk_{con}, (PC.PDS[]|PC.DH))$
34: $D_{ct} := \mathcal{AE}.Enc(epk, D), PC_{ct} := \mathcal{AE}.Enc(epk, PC)$
35: send("SaveData", D_{ct}, PC_{ct}) to $DU_{1,1}$

Figure 5.8: Formal representation of the $BPPM$ protocol

$Prog_{Eclv}^{BPPM}$

1: On receive ("EncSetKey"):
2: assert ($state = \varnothing$)
3: $(epk - esk) \leftarrow \mathcal{AE}.KeyGen(1^\lambda)$
4: $state := InsertCode$ and **output** epk
5: On receive ("EncInsertCode", $EC[], PDS[]$):
6: assert ($state = InsertCode$) ▷ Ensures "EncSetKey" is already invoked
7: $C_{loc}[] := \emptyset$ ▷ Initialize the array that holds the locations of the sealed code
8: $did := \emptyset, DList[] := \emptyset$ ▷ Initialize the storage area
9: $\forall_S \in EC[]$: ▷ Store all non-dummy code components in sealed format for later use
10: $c := UNPAD(\mathcal{AE}.Dec(esk, EC[S]))$:
11: **if** $c :\neq dummy$ **then**:
12: assert ($\Sigma.Vf(Cert_{Ad}.pub_key, PDS[S].AS, (H(c)|S))$) ▷ Verify auditor's signature
13: store c in permanent media in a sealed format
14: $C_{loc}[S]$:=Location of the sealed code ▷ To be used later to load the code within the enclave
15: $state := SaveData$
16: On receive ("EncSaveData", D_{ct}, PC_{ct}):
17: assert ($state = SaveData$)
18: $D := \mathcal{AE}.Dec(esk, D_{ct}), PC := \mathcal{AE}.Dec(esk, PC_{ct})$
19: assert ($H(D) = PC.DH$) ▷ Verify linkage between D and PC
20: assert ($\mathcal{AE}.Dec(D.CSK, \mathcal{AE}.Enc(PC.VK, < pattern >)) =< pattern >$) ▷ Verify linkage between $PC.VK$ and $D.CSK$
21: assert ($\Sigma.Vf(D.DOC.pub_key, D.DOS, (D.DS[]|PC.VK))$)
22: assert ($\Sigma.Vf(PC.VK, PC.SC, (PC.PDS[]|PC.DH))$)
23: $did++$ and then $DList[did].D := D, DList[did].PC := PC$ ▷ Store D and PC
24: $state := DataReceived$ and **output** (did, PC)
25: On receive ("EncProcess", did, S, y):
26: assert ($state = DataReceived$)
27: assert ($\exists PD \in DList[did].PC.PDS[] : PD.S = S$) ▷ Ensure that the sender has permitted S
28: c:=recover sealed code from the location $C_{loc}[S]$
29: load c into enclave's memory
30: **if** ($y \neq \bot$): **output** $c(DList[did].D)$ ▷ Execute code on the data and output computation result
31: **else**: **output** y ▷ Backdoor, implanted for the purpose of equivocation
32: On receive ("EncPrepFrd", $did, PDS_{rem}[], epk_{cld}, \sigma_{TEE}(epk_{cld})$):
33: assert ($state = DataReceived$)
34: assert $\Sigma_{TEE}.Vf(\sigma_{TEE}(epk_{cld}))$ ▷ Verify authenticity of the public-key
35: $FPC := DList[did].PC$ and then $FPC.PDS[] = FPC.PDS[] - PDS_{rem}[]$ ▷ Reduce consent
36: $FPC.SC = \Sigma.Sig(DList[did].D.CSK, (FPC.PDS[]|FPC.DH))$ ▷ Create new signature
37: **output** $(\mathcal{AE}.Enc(epk_{cld}, DList[did].D), \mathcal{AE}.Enc(epk_{cld}, FPC))$

Figure 5.9: Base enclave program of *BPPM*

5.5.2 Privacy and Security Goals

Our protocol must be equipped to ensure the following security and privacy goals:

1. $DU_{i,j}$ is required to provide a *Guarantee of Privacy Preservation* (refer to Section 5.4.2 for details) to **parent**$(DU_{i,j})$. This guarantee must be issued on behalf of the entire **tree**$(DU_{i,j})$ (which is the sub-tree of the complete data usage tree, rooted at $DU_{i,j}$), without disclosing any information about the data users in **children**$(DU_{i,j})$. In a recursive manner, $DU_{1,1}$ must provide a guarantee of privacy preservation to DO on behalf of the entire data usage **tree**$(DU_{1,1})$, ensuring that no information about other members in **tree**$(DU_{1,1})$ is revealed.

2. $DU_{i,j}$ should only learn **parent**$(DU_{i,j})$ as the source of the data it receives and **children**$(DU_{i,j})$ as the subsequent tier of data users. Furthermore, $DU_{i,j}$ must remain oblivious to the identity and existence of its siblings. As a result, only $DU_{1,1}$ should be aware of the true end service user (or the DO).

3. A malicious data owner must not be able to submit replayed data to gain unauthorized access to services.

4. DO must be able to approve a single consolidated list, PDS_{DO}, while remaining unaware of which data processings occur locally by $DU_{1,1}$ versus which are outsourced.

5. Only $DU_{i,j}$ and CP remain aware of $LPDS_{i,j}$. CP is considered trusted and does not disclose the details of $LPDS_{i,j}$, as revealing this could expose $DU_{i,j}$'s business strategy.

6. When forwarding data to $DU_{i,j}$, the **parent**$(DU_{i,j})$ must reduce the received PDS to PDS_{fwd} and DS to DS_{fwd}. PDS_{fwd} includes only the subset of personal data-processing statements agreed upon as necessary by **tree**$(DU_{i,j})$, while DS_{fwd} comprises the associated personal data. This reduction not only limits $DU_{i,j}$'s processing capabilities but also safeguards against an honest yet curious $DU_{i,j}$ gaining access to any additional information or insights regarding the processing activities carried out by its parent or siblings.

7. Although PDS_{fwd} and DS_{fwd} are reduced versions of the original datasets sent by the DO, $DU_{i,j}$ requires assurance regarding their authenticity. In other words, $BPPM$ must provide a guarantee to $DU_{i,j}$ that **parent**$(DU_{i,j})$ cannot add or modify any content while preparing PDS_{fwd} (or DS_{fwd}) from PDS (or DS), even though content may be removed.

8. To ensure that source code respects privacy and complies with the specified data processing statements, it is essential to conduct an audit. However, this process can be both expensive and time-consuming. Therefore, it is important to reuse pre-audited code in a controlled manner while also protecting against the unauthorized use of such privacy-preserving source code.

5.5.3 Ideal Functionality

$\mathcal{F}_{BPPM}[\Sigma, \Sigma_{TEE}, \mathcal{AE}, H, C[], EX], DO, DU_{i,j}]$

1: On receive ("SetUp", $PDS_{i,j}[]$) from $DU_{i,j}$:
2: notify $\mathcal{A}$ about ("SetUp", $DU_{i,j}$, $PDS_{i,j}[]$) and block until $\mathcal{A}$ replies
3: $DU_{i,j}.DList[] := \emptyset$ ▷ Initialize $DU_{i,j}$ specific storage
4: $DU_{i,j}.did := 0$ ▷ Initialize $DU_{i,j}$ specific index
5: $DU_{i,j}.PDS[] := PDS_{i,j}[]$
6: send public delayed output $PDS_{i,j}[]$
7: $DU_{i,j}.state = SaveData$
8: On receive ("SendOrigData", $PDS_{DO}[], DS_{DO}[], sk_{DO}, Cert_{DO}$) from DO:
9: assert ($DU_{i,j}.state = SaveData$)
10: notify $\mathcal{A}$ about ("SendOrigData", DO, $|PDS_{DO}[]|, |DS_{DO}[]|$) and block until $\mathcal{A}$ replies
11: $DU_{1,1}.did++$
12: store ($DS_{DO}[], PDS_{DO}[]$) within $DU_{1,1}.DList[DU_{1,1}.did]$
13: send ($PDS_{DO}[], DU_{1,1}.did$) to $DU_{1,1}$
14: $DU_{i,j}.state = DataReceived$
15: On receive ("Process", did, S, y) from $DU_{i,j}$:
16: assert ($DU_{i,j}.state = DataReceived$)
17: assert (($S \in DU_{i,j}.DList[did].PC.PDS[]$) and ($DU_{i,j} \in EX[S]$))
18: notify $\mathcal{A}$ about ("Process", did, S) and block until $\mathcal{A}$ replies
19: **if** $y = \perp$: send $C[S](DU_{i,j}.DList[did].D)$ to $DU_{i,j}$
20: **else:** send y to $DU_{i,j}$
21: On receive ("ForwardData", $did, PDS_{rem}[], cld, epk_{cld}, \sigma_{TEE}(epk_{cld})$) from $DU_{i,j}$:
22: assert ($DU_{i,j}.state = DataReceived$)
23: notify $\mathcal{A}$ about ("ForwardData", $DU_{i,j}, cld$, $|DU_{i,j}.DList[did].DS[]|, |DU_{i,j}.DList[did].PDS[] - PDS_{rem}[]|$) and block until $\mathcal{A}$ replies
24: $cld.did++$
25: copy ($DU_{i,j}.DList[did].DS[], (DU_{i,j}.DList[did].PDS[] - PDS_{rem}[])$) to $cld.DList[cld.did][]$
26: $cld.state = DataReceived$
27: send ($cld.DList[cld.did].PDS[], cld.did$) to cld

Figure 5.10: $\mathcal{F}_{BPPM}$: Ideal Functionality of $BPPM$

Based on the desired privacy and security objectives outlined in section 5.5.2, as well as the specified threat model and assumptions discussed in section 5.4.1, we establish an ideal functionality, denoted as $\mathcal{F}_{BPPM}$ (illustrated in Figure 5.10). Notably, CP is not involved in the ideal functionality, as its entry points are not exposed to $\mathcal{Z}$. Moreover, since the ideal functionality has access to both $C[]$ and $EX[]$, it inherently possesses knowledge of the code and execution permissions.

SetUp entry point of $\mathcal{F}_{BPPM}$:

During the Setup call, the information that $DU_{i,j}$ is currently engaged in Setup becomes apparent to $\mathcal{A}$ due to the visibility of the multiple SendCode calls. Additionally, $PDS_{i,j}[]$ is accessible to an external party since it is included in the data user's privacy policy document. To address this, $\mathcal{F}_{BPPM}$ informs $\mathcal{A}$ with the 3-tuple: (Setup, $DU_{i,j}$, $PDS_{i,j}[]$). Next, $\mathcal{F}_{BPPM}$ sets up the necessary storage space, tailored to the invoking data user, and records the input $PDS_{i,j}[]$. This information will be utilized later during the data processing and forwarding stage.

SendOrigData entry point of $\mathcal{F}_{BPPM}$:

During this call, only the size of the inputs is revealed to the adversary, as the data owner encrypts the information prior to transmission over the network. If the adversary allows this network communication, $\mathcal{F}_{BPPM}$ retains the approved data processing statements along with the corresponding personal data. Additionally, since the actual execution of the protocol provides the primary data user with the list of approved data processing statements and the newly-generated *did* at the conclusion of this stage, $\mathcal{F}_{BPPM}$ also discloses them to $DU_{1,1}$.

Process entry point of $\mathcal{F}_{BPPM}$:

$\mathcal{F}_{BPPM}$ first verifies that the $DU_{i,j}$ possesses the necessary code execution permission privileges. Additionally, it checks that the consent structure associated with *did* permits the processing of S. If both conditions are satisfied, the relevant processing on the personal data is executed, and the resulting computation is returned.

In a manner similar to $\mathcal{F}_{ROT}$, for the purpose of simulability, Sim may also communicate the computation result via input argument y, which in $\mathcal{F}_{BPPM}$ can output to DU. This will not pose any security risks for the same reasons.

ForwardData entry point of $\mathcal{F}_{BPPM}$:

$\mathcal{F}_{BPPM}$ already contains the data-consent pair for the parent in its internal data structure, $DU_{i,j}.DList[]$. Therefore, upon invoking this entry point, it removes the requested $PDS_{rem}[]$ from the parent's processing statements and presents the resulting list to the child. Additionally, $\mathcal{F}_{BPPM}$ internally copies the data and reduced processing statements from the parent's specific internal data structure to that of the child. Furthermore, to address external observations during this stage, $\mathcal{F}_{BPPM}$ discloses the identities of the communicating parties along with the size of the list and data that is being forwarded.

5.5.4 UC-proof

In a manner akin to the proof for ROT, we demonstrate the existence and design of Sim. Then we establish that Z's view remains indistinguishable between the real world and ideal world across the four fundamental design stages previously outlined: Setup, SendOrigData, ForwardData and ProcessData. Appendix: H elaborates on the design of the simulator and presents a rationale for its indistinguishability, encompassing all potential combinations of corrupt and honest parties.

5.6 Implementation and Performance Analysis

We now detail the essential aspects of our *BPPM* implementation. After this, we will compare the performance of *BPPM* to a non-privacy-preserving scenario, concentrating on computation time and communication bandwidth.

5.6.1 Implementation Details

We developed a proof of concept for *BPPM* in C-language, which is available online [162]. Similar to ROT, Intel-SGX serves as the underlying TEE technology and utilized an SGX-enabled VM instance, DC4SV3 [163]. This DC4SV3, instance consists of a quad-core processor (Intel Xeon Platinum @2.8 GHz) and is equipped with 32GB of RAM. Ubuntu 20.04 is used as the operating system. To prepare the enclave-side code, we use the Gramine shim library [164, 165]. Our base enclave code, $Prog_{Eclv}^{BPPM}$, consists of approximately 2.1 KLOC (thousand lines of code). For cryptographic primitives and network operations, we depend on the MBed-TLS library [166].

Similar to *ROT*, we utilize an attested TLS channel for communication with the enclave. As a result, instead of encrypting the entire data using the recipient enclave's public key, we employ hybrid encryption (i.e., the sender generates a fresh symmetric key to encrypt the plaintext and that symmetric key is encrypted with the recipient's public key) to send data or code, confidentially, to an enclave.

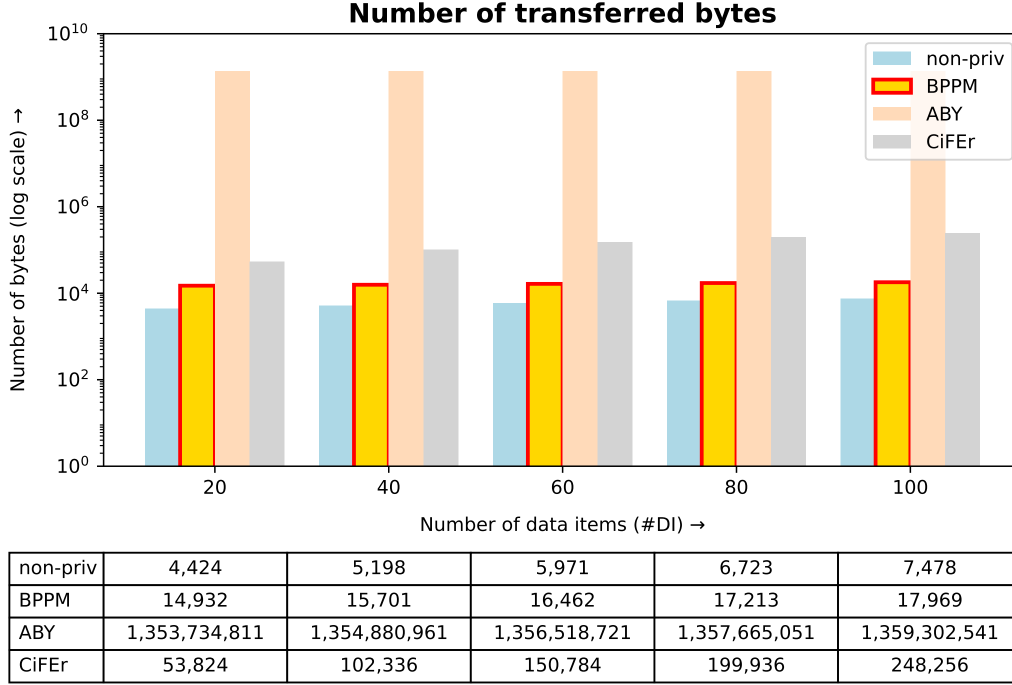
In *BPPM*, it is crucial to enhance the computation capability of the data user's base enclave. Rather than adopting the approach used by other researchers, which involves sending a new enclave [167], we transferred the necessary code components in the form of encrypted dynamic libraries. This method not only reduces storage requirements but also shortens network transfer times. Additionally, it eliminates the overhead associated with launching a new enclave. For example, a standard enclave size of 1088KB can be replaced with a 16KB binary of a dynamic library. The loading of this dynamic library takes approximately 1.17ms, compared to around 15.4ms required for enclave initialization, along with an extra 1.1ms for the local attestation of the new enclave.

Our approach addresses a significant practical concern associated with TEE-based solutions: the size limitations of enclaves. To optimize space within the base enclave, our approach allows libraries to be unloaded and securely stored on permanent media when they are not in use. To protect against piracy, the unloaded binaries are saved in a sealed format, ensuring that only the same instance of the base enclave can load and execute them in the future. Since the binaries are never stored in plaintext outside of the enclave, reverse engineering is impossible.

5.6.2 Performance Analysis

We assessed the performance of *BPPM* by conducting each experiment 100 times and calculating the average results. Performance metrics for *BPPM* and other candidates were collected while running all tests in the same environment. For these experiments, we employed the local loopback interface, allowing us to focus on the volume of data transferred over the network rather than actual network latency.

In this context, we also find a lack of directly-comparable solutions. Therefore, as with *ROT*, we compare *BPPM* with both CiFEr and ABY. However, we have excluded TEEKAP from our comparison, as it is not applicable to the data sharing scenario at all. We examine how performance is impacted as the number of data attributes varies from 20 to 100, using the same rationale. In our analysis, we assume that, on average, each personal data attribute (e.g., age or passport number) is represented by a 64-bit unsigned integer.



non-priv	4,424	5,198	5,971	6,723	7,478
BPPM	14,932	15,701	16,462	17,213	17,969
ABY	1,353,734,811	1,354,880,961	1,356,518,721	1,357,665,051	1,359,302,541
CiFEr	53,824	102,336	150,784	199,936	248,256

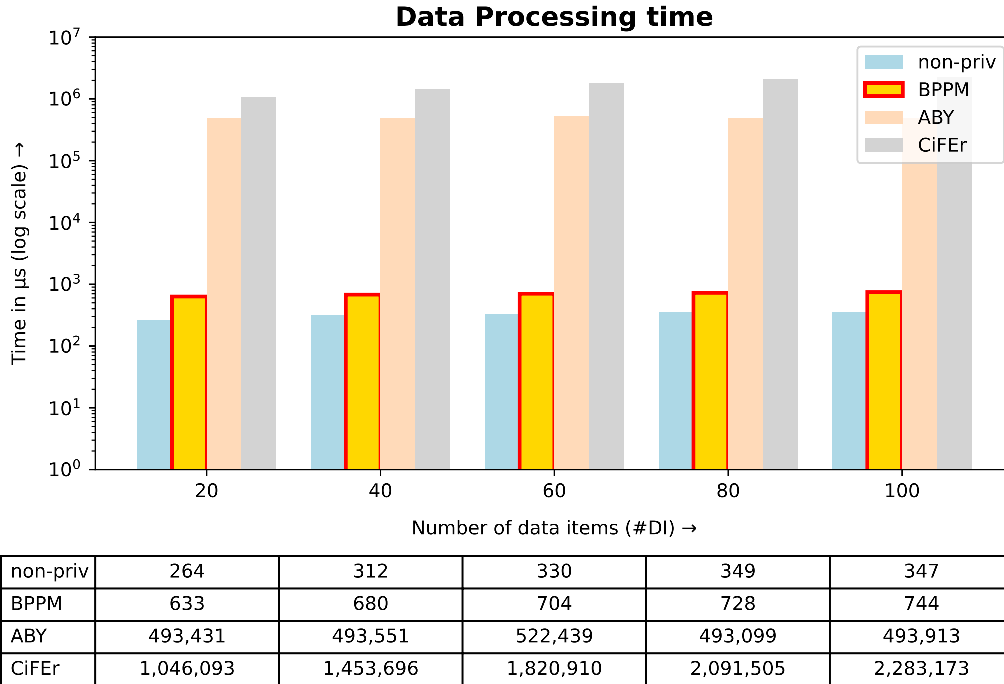
Figure 5.11: Communication cost comparison

In a manner similar to *ROT*, we also examine the overhead that *BPPM* introduces compared to a non-privacy-preserving scenario, $non-priv$, which we previously defined as follows: After consenting to the privacy policy, the service user transmits its personal data to the service provider over a server-authenticated TLS channel. The service provider then verifies the authenticity of the received personal data and stores it in encrypted format within a database. Whenever the service provider needs to conduct computations on the data, it retrieves the encrypted data, decrypts it, and performs the necessary computations.

In *BPPM*, when personal data is transferred, the corresponding *PC* structure (occupying ∼3.5 KB in a typical scenario) is also included in the transfer. Additionally, each transfer undergoes a remote attestation process that incurs an extra cost of about 7 KB. As a result, compared to a *non-priv* transfer, *BPPM* entails an additional communication cost of approximately 10.5 KB - comprising 7 KB for remote attestation and 3.5 KB for the *PC* structure - when transferring personal data.

Figure 5.11 illustrates the comparison when there is only one computation on the data user's side (i.e., $|PDS[]| = 1$). It is important to note that *BPPM* requires only one round of communication, regardless of the number of data-processing operations the data user performs or how many times these operations are executed. All processing parameters can be defined through a single *PC*-structure (see Figure 5.3). While the size of the *PC*-structure may slightly increase with the number of processing statements (approximately 1KB per statement), a single transfer suffices.

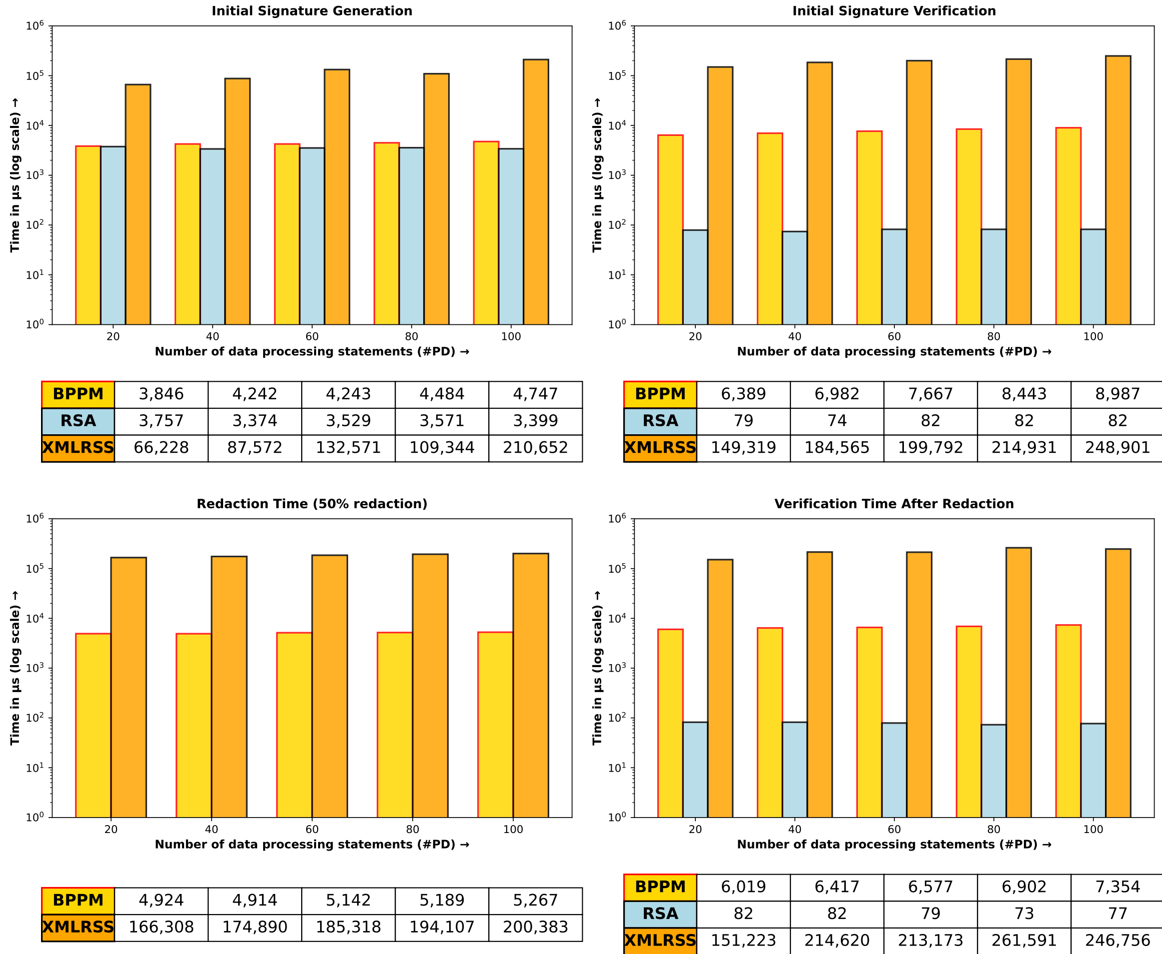
After analyzing the experimental results, we found CiFEr ciphertext is 64 times larger than the corresponding plaintext, which significantly increases the required communication costs. In contrast, *BPPM* exhibits a communication cost that is 3 to 14 times smaller. Meanwhile, ABY incurs approximately 7500 to 9000 times the communication cost of *BPPM*, largely due to the inherent demands of oblivious transfers.



	20	40	60	80	100
non-priv	264	312	330	349	347
BPPM	633	680	704	728	744
ABY	493,431	493,551	522,439	493,099	493,913
CiFEr	1,046,093	1,453,696	1,820,910	2,091,505	2,283,173

Figure 5.12: Processing time comparison

Figure 5.11 illustrates the details. One important aspect is, both FE and MPC necessitate the repetition of the entire network transfer process for each data processing operation, even when applied to the same data. Consequently, the communication costs for FE and MPC must be multiplied by the number of data processing statements (i.e., $|PDS[]|$).



BPPM	3,846	4,242	4,243	4,484	4,747
RSA	3,757	3,374	3,529	3,571	3,399
XMLRSS	66,228	87,572	132,571	109,344	210,652

BPPM	6,389	6,982	7,667	8,443	8,987
RSA	79	74	82	82	82
XMLRSS	149,319	184,565	199,792	214,931	248,901

BPPM	4,924	4,914	5,142	5,189	5,267
XMLRSS	166,308	174,890	185,318	194,107	200,383

BPPM	6,019	6,417	6,577	6,902	7,354
RSA	82	82	79	73	77
XMLRSS	151,223	214,620	213,173	261,591	246,756

Figure 5.13: Redactable Signature Performance Comparsion

Unsurprisingly, the comparison of processing times reveals some similarities to the analysis conducted in the case of *ROT*. However, in contrast to *ROT*, which is three

times slower than the *non-priv* mode, *BPPM* is only about twice as slow as the *non-priv* mode. This aligns with the anticipated performance decline associated with the enclave environment [136], particularly due to the lack of interaction with the blockchain in *BPPM*. Nonetheless, *BPPM* remains an impressive 1600-3000× faster than CiFEr and 600-750× faster than ABY (refer to Figure 5.12 for details).

During the ForwardData stage, *BPPM* generates a reduced version of the *PC* structure, while preserving the anonymity of the *DO*. This can be achieved using a specific type of RSS called a signer-anonymous designated-verifier redactable signature scheme (AD-RS) [168]. However, it is worth noting that RSS schemes are often slow due to their reliance on computationally intensive cryptographic techniques, such as zero-knowledge proofs.

In *BPPM*, we achieve the same effect as AD-RS by leveraging the confidentiality and attested execution properties inherent in TEE technology. We compare the performance of *BPPM* against an existing RSS implementation, XMLRSS [169]. Additionally, we assess the overhead introduced by *BPPM* in relation to a standard (non-redactable) RSA-2048 signature scheme, which serves as a baseline for a non-privacy-preserving authentication mechanism. While *BPPM* is not as fast as RSA-2048, it still exhibits a notable performance improvement over XMLRSS. Specifically, *BPPM* generates initial signatures over 15 times faster, verifies signatures more than 24 times quicker, and redacts data more than 34 times faster (see Figure 5.13 for details).

5.7 Cost-Benefit Analysis

In this section, we examine the costs and benefits associated with the parties involved in *BPPM*. Similar to other PETs, *BPPM* provides privacy preservation, but incurs computation and communication overhead.

5.7.1 Costs and Benefits for Service Users

In summary, *BPPM* results in double the processing latency and an additional 10.5 KB of network bandwidth usage. Nevertheless, when compared to existing PETs, *BPPM* demonstrates a significantly higher efficiency and offers a unique and essential set of features. By leveraging the attested execution property, *BPPM* ensures that privacy policies are consistently enforced. It streamlines data-sharing decisions for service users by offering a comprehensive list of all data-processing statements, removing the necessity to navigate

through the privacy policies of every involved party. *BPPM*'s strategy aligns with ongoing efforts to enhance transparency and control in data-sharing practices [143].

By enabling service users to select any arbitrary subset from the consolidated list of data processing statements, *BPPM* enhances their control over data processing, extending beyond just the primary data user. Furthermore, the unified list facilitates the seamless integration of existing automated privacy negotiation tools, such as P3P [170] and Privacy Bird [171], thereby easing the burden on service users.

Research indicates that identical privacy policies can result in varying code implementations [172, 173]. In fact, the same privacy statement can carry different implications, depending on the jurisdiction. Moreover, both the privacy policy and the related processing code may change after personal data has been collected, often without notifying the service users [174]. This scenario creates a potential loophole for service providers. In *BPPM*, service users not only consent to the data processing terms outlined in the privacy policy but also to a specific processing code that has been approved by an auditor, effectively addressing these concerns.

5.7.2 Costs and Benefits for Service providers

In today's environment, service providers find themselves needing to allocate substantial resources to comply with various privacy regulations such as GDPR [175], PIPEDA [176], and CCPA [177]. Table. 5.1 offers an estimate of the costs that a mid-sized organization (with 50 to 250 employees) might incur to adhere to these three key privacy regulations.

Addressing the intricacies of compliance necessitates substantial expertise in privacy-related matters, a challenge that is especially impactful for smaller organizations [178]. However, research suggests that specific personal data is predominantly utilized for a limited range of purposes [174, 179]. This insight promotes the reuse of privacy-related software, ultimately resulting in significant cost savings.

Acquiring processing code components from a code provider entails certain costs. While *BPPM* has not yet been implemented in practical scenarios, we expect that, if adopted, these costs would be significantly lower than those presented in Table 5.1. Importantly, service providers can now transfer auditing and privacy responsibilities to their code providers, thereby simplifying processes and reducing the typical delays associated with software development.

Cost Category	**GDPR**	**CCPA**	**PIPEDA**
Initial incorporation of privacy features in existing software	> 136.8K CAD [180, 181]	>136.8K CAD [182, 183]	>41.0K CAD [184]
Annual Software Maintenance	>68.4K CAD [180, 181]	Not found	>20.5K CAD [184]
Privacy Officers' Annual Salary	>273.7K CAD [180, 185, 186]	>273.7K CAD [180, 185, 186]	>273.7K CAD [180, 185, 186]
Annual Privacy Consultant	>13.7K CAD [180, 187]	>13.7K CAD [180]	>13.7K CAD [180]
Annual Privacy-Specific Audit	> 41.0K CAD [181, 187]	> 27.4K CAD [183]	Not found
Total annual cost	> 534.1K	> 451.9K	> 349.2K

Table 5.1: Compliance Cost Comparison Across Regulations (assuming 1 CAD=0.73 USD)

From the perspective of code providers, the opportunity for extensive reuse of privacy-preserving processing code enables them to continually develop and enhance their libraries. Additionally, the piracy-free distribution model of *BPPM* not only encourages code providers but also fosters a sustainable ecosystem.

Currently, there is a notable shortage of privacy auditors [149], and the auditing process is time-intensive, which can impact business operations. *BPPM* effectively addresses these challenges by enabling the auditing of the same data processing statement and its corresponding data processing code at the code provider's location, performed only once. This method not only alleviates ambiguity surrounding personal data processing [173, 188] but also promotes the standardization of personal data processing practices across various organizations worldwide.

Auditors are tasked with not only monitoring the storage and retention of personal data but also conducting a comprehensive review of all software flow associated with the personal data. It's important to note that these processes may evolve over time, necessitating periodic re-audits. *BPPM* significantly reduces the auditing scope, since it does not expose personal data in plaintext.

Consequently, many data handling and auditing requirements related to data storage are greatly diminished. Furthermore, aside from the code provided by the code provider, no additional components of the service provider's software are capable of accessing personal data. This eliminates the need for privacy inspections across the rest of the codebase.

Chapter 6

Conclusions and Future Work

In our research, we focused primarily on two key aspects of web services: the safeguarding of access patterns and the protection of personal data. We explored subcategories within each of these areas in greater detail. In this chapter, we summarize our findings and discuss possible directions for future work that align with our research interests.

6.1 Regarding *RouterORAM*

We introduce *RouterORAM*, an ORAM solution that safeguards the client's access patterns, while accessing remote private storage. It reduces the access latency and client workload from $O(\log \mathsf{N})$ to $O(1)$. While it imposes no restrictions on the client's access patterns, its advantages are maximized in scenarios characterized by bursty, read-dominated access, which are quite common while accessing remote storage. *RouterORAM* is founded on the innovative concept of server-assisted routing for intentionally misplaced blocks.

RouterORAM leverages the server's unused computational capacity via homomorphic evaluation, thereby reducing both access latency and the client's workload without compromising privacy. On rare occasions, *RouterORAM* may not deliver an accurate response, requiring the client to retry. However, we have verified that this retry mechanism does not introduce any privacy risks. Furthermore, simulations assessing *RouterORAM*'s long-term behavior indicate that both latency and client workload remain O(1), even in scenarios involving retrials.

While *RouterORAM* demonstrates a reduction in latency and client workload from a theoretical complexity standpoint, we acknowledge that we have not yet implemented or

deployed it in real-world applications. Indeed, further research is possible, particularly concerning implementation efficiency. The constant $O(1)$ term in our analysis is primarily influenced by the execution time of the underlying homomorphic evaluations, which may be optimised by leveraging specific properties of various homomorphic encryption methods.

Since *RouterORAM* does not require maintaining any client local stash, it remains stateless. The combination of its parallelizability and statelessness opens avenues for future development of a distributed oblivious file system. *RouterORAM* can be used as a protective measure against side-channel attacks aimed at TEEs. Building on prior research [189], we also propose an enhancement to the concept of developing a compiler that transforms programs to be free of side-channel leakage, all while improving execution speed, even for platforms without TEEs.

Several studies [190, 191] have demonstrated that integrating ORAM techniques at the hardware level can significantly improve the efficiency of program execution while preserving access pattern privacy. Given that the access pattern of *RouterORAM* closely mirrors program execution, exploring the application of *RouterORAM* in developing hardware architectures aimed at protecting access patterns, as well as in the design of future TEEs, represents an interesting area for future research.

An intriguing aspect to consider in this context is that, if *RouterORAM* is used alongside TEEs, we can substitute FHE with TEEs' inherent ability to execute computation on encrypted data. This approach has proven to be significantly more efficient than FHE thus far. Consequently, this combination could further enhance efficiency. Moreover, by leveraging the attested execution property of TEEs, it may be possible to defend against more formidable adversaries, such as actively malicious servers, rather than only against the traditional honest-but-curious model.

6.2 Regarding PIR-schemes

In our efforts to obscure access patterns in the context of public databases, we have developed three PIR schemes aimed at reducing the server overhead. Our first scheme achieves a server overhead reduction to $O(\sqrt{\mathsf{N}})$. Building upon this foundation, our second scheme incorporates a differentially private approach that further decreases the server overhead. However, while our first scheme demonstrates asymptotic efficiency, we encountered a practical challenge: the individual operations are considerably intensive. To tackle this issue, we devised a third scheme that maintains the $O(\sqrt{\mathsf{N}})$ overhead while significantly reducing the number of intensive operations to just $O(1)$.

PIR is a building block for several other PETs and can be used to enhance their privacy guarantees. For example, the Tor network [192] helps to hide the identities of its users while they communicate over the internet. Tor relies on a number of relay nodes, spread across the world. However, before establishing a Tor end-to-end channel, the user must pick a few relay nodes from a central database. If the central database administrator monitors this picking process, Tor's privacy guarantee is reduced. In this case, our PIR solutions can be used during the relay node choosing phase to enhance the privacy while reducing the overhead of the process.

Usually, with PIR (or with most of the PETs), available features get restricted. Since a PIR scheme hides all access patterns from the server, the server cannot keep track of per-item usage statistics. However, that might be crucial information for certain applications, such as YouTube, where the server must pay content creators based on the number of views. An additional benefit of our PIR schemes is their ability to provide usage statistics to the server without compromising the privacy of individual clients.

In traditional PIR, it is generally assumed that the database's content remains static. However, in our approach, the entire database is reorganized after each epoch, indicating that the database administrator can update the database content periodically. More interestingly, our PIR scheme utilizes the ORAM strategy, and an ORAM allows a single client to read and write. As a result, a compelling direction for future research would be to investigate the feasibility of enabling the database administrator to execute update operations at any time.

During our empirical analysis of the PIR scheme (refer to Section 3.4.13), we observed that DPF evaluations are taking a comparatively longer time. This delay can be attributed to the large size of the FHE ciphertexts stored in the shelter. However, it is crucial to highlight that we do not require any homomorphic evaluation on the contents of the shelter (for our *DPF-Variant Scheme*).

One possible theoretical research direction could be to investigate whether a provable relationship exists between ORAM and PIR, particularly in a multi-server context. If such a relationship is established, it would be essential to ascertain whether it is possible to determine the lower bound of overhead in multi-server PIR systems. Additionally, it is important to check if this lower bound matches that of the ORAM lower bound of $\log \mathsf{N}$. A positive resolution to these inquiries may provide new opportunities for achieving PIR with even lower overhead.

In our analysis of the privacy-performance tradeoff of our DP-IR scheme (Section 3.3.7), we assumed that the adversary can control all other requests except for the targeted client's request. Furthermore, the adversary has the ability to manipulate the order of PIR re-

quests, granting them significant leverage. However, if a new honest-but-curious server is introduced between the client and our DP-IR scheme, then its role would be to batch and mix the requests and responses.

This has some similarity with the concept of an anonymous channel discussed by Toledo et al. [60, Section 3.3]). Indeed, it could lead to a substantial enhancement in privacy with a comparably low overhead. Lastly, while a lower limit analysis of DP-IR schemes within the balls-and-bins model has been conducted [65], an exploration of the lower limits in non-balls-and-bins models remains a promising theoretical direction.

6.3 Regarding *ROT*

We present *ROT*, a framework that ensures the personal data of the owner is used solely for authorized computations. Additionally, the data owner can define an expiration period, after which the personal data becomes unusable. *ROT* leverages TEE technology on the data user's side, which is susceptible to rollback attacks.

We demonstrate how a public blockchain can be utilized to protect against such vulnerabilities. Our implementation of *ROT* includes performance assessments that highlight its practicality. Notably, *ROT* is not only significantly more efficient and practical than other related cryptographic primitives, but it also results in considerably lower computational and communication overhead compared to other TEE-based solutions.

In addition to TEE, FHE can also be utilized to develop a data-expiry mechanism. Recently, Manulis and Nguyen introduced the concept of FHE with verifiability [193]. In their approach, they first integrate a CPA-secure FHE scheme within a CCA2-secure encryption framework. They then modify the evaluation algorithm to connect the input ciphertext with the resulting output ciphertext.

During the evaluation process, their FHE scheme generates a proof alongside the output ciphertext. The proof demonstrates that the ciphertext was produced by applying the function f to the input ciphertext. Before revealing the plaintext, the decryption algorithm checks this proof.

It might be possible to expand this scheme to construct a recursive proof. Such a proof would not only link the input ciphertext to the output ciphertext, but would also relate to previous proofs. If this is feasible, the decryption algorithm must ensure that the function f is evaluated fewer than t times before revealing the plaintext, effectively implementing a data expiry mechanism.

6.4 Regarding *BPPM*

While *ROT* safeguards personal data in a single location, *BPPM* is designed for situations where personal data must be shared among a hierarchy of data users or the service providers. One significant barrier to the adoption of PETs in real-world scenarios is their high costs, which are often borne by the service providers. Therefore, in addition to addressing various privacy concerns for data owners, *BPPM* offers several benefits for service providers and helps reduce operational costs.

Governments in several countries have begun treating privacy concerns very seriously and have already started to take action from a legal standpoint. In several countries, privacy auditing or the use of PETs has become mandatory, which is not only expensive but also time-consuming. Our solution is an attractive option for those scenarios. It not only upholds the established privacy standards but also reduces complexity and operating costs for the service providers. These serve as compelling motivating factors for *BPPM*'s adoption in practical applications, compared with other available options.

As previously discussed (Section 5.4.5), our *BPPM* scheme can be seamlessly integrated with blockchain technology to generate differentially private outputs while providing rollback protection. This concept can be further refined to address the inference problem. Each time a new computation result involving personal data is disclosed, it reveals additional insights about that data. Essentially, each evaluation diminishes the adversary's uncertainty regarding the personal data. Different parties can collude and accumulate these bits of information, potentially uncovering a significant amount of data, even if the actual personal data remains encrypted.

It may be worthwhile to explore if it is "somehow" possible to measure the amount of information leaked by each evaluation. If feasible, the privacy auditors may also specify the amount of leakage when signing the processing code components. In that case, the enclave could record the remaining privacy balance on an immutable medium, such as blockchain, following each evaluation. During subsequent evaluation requests, the enclave would be able to verify, irrespective of which party's enclave, whether the remaining privacy balance allows for further computation.

6.5 Overall Future work and Conclusion

Privacy comes at a cost, which includes increased complexity, usability challenges, and economic implications. However, the current PETs designed to prevent the leakage of

personal data and access patterns are often prohibitively expensive due to impractical overheads. As a result, protection against these two privacy issues in web services has been largely overlooked, leading to persistent privacy breaches.

The overall goal of our research is to reduce these overheads and make it attainable to address these privacy concerns in everyday life. In addition to protecting users' privacy, we also aim to provide some benefits for service providers, who typically absorb a significant portion of the costs associated with privacy measures. We hope that these advancements will encourage both service users and providers to adopt these PETs in practical applications.

For the sake of privacy, users usually sacrifice features. For example, using a PIR system means they cannot get personalized recommendations. Whether losing that feature for a better privacy guarantee is fine or not depends on the situation and is actually subjective. Hence, as a broad-level future work, it can be investigated whether our solutions can be updated to make the privacy features or their degree of privacy tunable, on a per-user basis.

Finally, we recognize that our targeted research area represents only a fraction of the broader privacy issues associated with web services. There are numerous other facets and challenges that need to be examined in this context. For a thorough analysis, please refer to the detailed insights available here [194]. Indeed, enhancing privacy encompasses more than just technical hurdles. There is also a need to raise awareness and educate users about privacy matters [195], which is a broad and multidisciplinary area of research.

References

[1] R. Pass, E. Shi, and F. Tramèr, "Formal Abstractions for Attested Execution Secure Processors," in *Advances in Cryptology – EUROCRYPT 2017*, J.-S. Coron and J. B. Nielsen, Eds. Cham: Springer International Publishing, 2017, vol. 10210, pp. 260–289, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-319-56620-7_10

[2] D. Boneh and V. Shoup, "A Graduate Course in Applied Cryptography , version 0.6," 2023. [Online]. Available: https://toc.cryptobook.us/book.pdf

[3] S. Oya and F. Kerschbaum, "Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 127–142. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/oya

[4] C. Liu, L. Zhu, M. Wang, and Y.-a. Tan, "Search pattern leakage in searchable encryption: Attacks and new construction," *Information Sciences*, vol. 265, pp. 176–188, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025513008293

[5] S. K. Paul and D. A. Knox, "RouterORAM: An O(1)-Latency and Client-Work ORAM," in *Foundations and Practice of Security*, K. Adi, S. Bourdeau, C. Durand, V. Viet Triem Tong, A. Dulipovici, Y. Kermarrec, and J. Garcia-Alfaro, Eds. Cham: Springer Nature Switzerland, 2025, vol. 15532, pp. 393–413, series Title: Lecture Notes in Computer Science. [Online]. Available: https://link.springer.com/10.1007/978-3-031-87499-4_26

[6] P. Sumit, Kumar, "ROT: Retention and Operation limitation using TEE," Jan. 2024. [Online]. Available: https://github.com/sumitkumarpaul/data_rotting.git

[7] S. K. Paul and D. A. Knox, "Bidirectional privacy preservation in web services," *Computers*, vol. 14, no. 11, 2025. [Online]. Available: https://www.mdpi.com/2073-431X/14/11/484

[8] K. S. Tinani, B. Choithwani, B. Patil, P. Faiyazkhan, and T. Salat, "Study on usage pattern of public cloud storage," *ijcse*, vol. 7, no. 6, pp. 922–927, June 2019. [Online]. Available: https://doi.org/10.26438/ijcse/v7i6.922927

[9] "Build a custom key service for client-side encryption | Google Workspace." [Online]. Available: https://developers.google.com/workspace/cse/guides/overview

[10] "How to turn on Advanced Data Protection for iCloud." [Online]. Available: https://support.apple.com/en-us/108756

[11] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," 2012. [Online]. Available: https://www.ndss-symposium.org/ndss2012/access-pattern-disclosure-searchable-encryption-ramification-attack-and-mitigation

[12] stevevi, "Canada Protected B - Azure Compliance." [Online]. Available: https://learn.microsoft.com/en-us/azure/compliance/offerings/offering-canada-protected-b

[13] "Percent of Corporate Data Stored in the Cloud (2024)," Mar. 2023. [Online]. Available: https://explodingtopics.com/blog/corporate-cloud-data

[14] "Gartner Forecasts Worldwide Public Cloud End-User Spending to Total $723 Billion in 2025," Gartner Press Release.

[15] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *J. ACM*, vol. 43, no. 3, pp. 431–473, May 1996. [Online]. Available: https://doi.org/10.1145/233551.233553

[16] O. Goldreich, "Towards a theory of software protection and simulation by oblivious RAMs," in *Proceedings of the nineteenth annual ACM conference on Theory of computing - STOC '87*. New York, New York, United States: ACM Press, 1987, pp. 182–194. [Online]. Available: http://portal.acm.org/citation.cfm?doid=28395.28416

[17] E. Stefanov *et al.*, "Path oram: an extremely simple oblivious ram protocol," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*. Berlin, Germany: ACM Press, 2013, pp. 299–310. [Online]. Available: https://doi.org/10.1145/2508859.2516660

[18] R. Balasubramonian, “Memory security,” in *Innovations in the Memory System*, ser. Synthesis Lectures on Computer Architecture. Cham: Springer International Publishing, 2019, pp. 81–101. [Online]. Available: https://doi.org/10.1007/978-3-031-01763-6_11

[19] G. Asharov, I. Komargodski, W.-K. Lin, K. Nayak, E. Peserico, and E. Shi, “Optorama: Optimal oblivious ram,” in *Advances in Cryptology – EUROCRYPT 2020*, ser. Lecture Notes in Computer Science, A. Canteaut and Y. Ishai, Eds. Cham: Springer International Publishing, 2020, vol. 12106, pp. 403–432. [Online]. Available: https://doi.org/10.1007/978-3-030-45724-2_14

[20] “The price of computer storage has fallen exponentially since the 1950s.” [Online]. Available: https://ourworldindata.org/data-insights/the-price-of-computer-storage-has-fallen-exponentially-since-the-1950s

[21] L. Ren, C. Fletcher, A. Kwon, E. Stefanov, E. Shi, M. Van Dijk, and S. Devadas, “Constants count: Practical improvements to oblivious ram,” in *Proceedings of the 24th USENIX Security Symposium*, 2015, pp. 415–430.

[22] E.-O. Blass, T. Mayberry, G. Noubir, and K. Onarlioglu, “Toward robust hidden volumes using write-only oblivious ram,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. Scottsdale Arizona USA: ACM, November 2014, pp. 203–214. [Online]. Available: https://doi.org/10.1145/2660267.2660313

[23] D. S. Roche, A. Aviv, S. G. Choi, and T. Mayberry, “Deterministic, stash-free write-only oram,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Dallas Texas USA: ACM, October 2017, pp. 507–521. [Online]. Available: https://doi.org/10.1145/3133956.3134051

[24] S. Tople, Y. Jia, and P. Saxena, “Pro-oram: Practical read-only oblivious ram,” in *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, 2019, pp. 197–211.

[25] O. Ohrimenko, M. T. Goodrich, R. Tamassia, and E. Upfal, “The melbourne shuffle: Improving oblivious storage in the cloud,” in *International Colloquium on Automata, Languages, and Programming*. Springer, 2014, pp. 556–567.

[26] E. Stefanov and E. Shi, “Oblivistore: High performance oblivious cloud storage,” in *2013 IEEE Symposium on Security and Privacy*. Berkeley, CA: IEEE, May 2013, pp. 253–267. [Online]. Available: https://doi.org/10.1109/SP.2013.25

[27] J. Dautrich, E. Stefanov, and E. Shi, "Burst oram: Minimizing oram response times for bursty access patterns," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 749–764.

[28] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. P. Fitzek, and N. Aaraj, "Survey on Fully Homomorphic Encryption, Theory, and Applications," *Proceedings of the IEEE*, vol. 110, no. 10, pp. 1572–1609, Oct. 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9910347/

[29] D. Apon, J. Katz, E. Shi, and A. Thiruvengadam, "Verifiable oblivious storage," in *Public-Key Cryptography – PKC 2014*, 2014, vol. 8383, pp. 131–148.

[30] S. Devadas, M. Van Dijk, C. W. Fletcher, L. Ren, E. Shi, and D. Wichs, "Onion oram: A constant bandwidth blowup oblivious ram," in *Theory of Cryptography*, ser. LNCS, E. Kushilevitz and T. Malkin, Eds., 2016, vol. 9563, pp. 145–174.

[31] H. Chen, I. Chillotti, and L. Ren, "Onion ring oram: Efficient constant bandwidth oblivious ram from (leveled) tfhe," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 345–360.

[32] K. Cong, D. Das, G. Nicolas, and J. Park, "Poster: Panacea — stateless and non-interactive oblivious ram," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 3585–3587.

[33] K. G. Larsen and J. B. Nielsen, "Yes, There is an Oblivious RAM Lower Bound!" in *Advances in Cryptology – CRYPTO 2018*, H. Shacham and A. Boldyreva, Eds. Cham: Springer International Publishing, 2018, pp. 523–542.

[34] W. W. Hsu and A. J. Smith, "Characteristics of i/o traffic in personal computer and server workloads," *IBM Syst. J.*, vol. 42, no. 2, pp. 347–372, 2003. [Online]. Available: https://doi.org/10.1147/sj.422.0347

[35] Y. Chen, K. Srinivasan, G. Goodson, and R. Katz, "Design implications for enterprise storage systems via multi-dimensional trace analysis," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. Cascais Portugal: ACM, October 2011, pp. 43–56. [Online]. Available: https://doi.org/10.1145/2043556.2043562

[36] A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller, "Measurement and analysis of large-scale network file system workloads," in *ATC'08: USENIX 2008 Annual Technical Conference*, 2008, pp. 213–226.

[37] J. Katz and L. Yehuda, *Introduction to Modern Cryptography, Second Edition*, 2014.

[38] P. Sumit, Kumar, "Oram-simulator," Online, September 2024, rust. [Online]. Available: https://github.com/sumitkumarpaul/oram

[39] E. Mathieu, "The price of computer storage has fallen exponentially since the 1950s," Tech. Rep., May 2024. [Online]. Available: https://ourworldindata.org/data-insights/the-price-of-computer-storage-has-fallen-exponentially-since-the-1950s

[40] S. Fei, Z. Yan, W. Ding, and H. Xie, "Security vulnerabilities of sgx and countermeasures: A survey," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–36, 2022. [Online]. Available: https://doi.org/10.1145/3456631

[41] S. Sasy, S. Gorbunov, and C. W. Fletcher, "ZeroTrace : Oblivious Memory Primitives from Intel SGX," in *Proceedings 2018 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2018. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_02B-4_Sasy_paper.pdf

[42] A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing Digital Side-Channels through Obfuscated Execution," 2015, pp. 431–446. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/rane

[43] A. Limaye and T. Adegbija, "A Workload Characterization of the SPEC CPU2017 Benchmark Suite," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Belfast: IEEE, Apr. 2018, pp. 149–158. [Online]. Available: https://ieeexplore.ieee.org/document/8366949/

[44] T. Siraj, M. Anwar, and A.-A. Bhuiyan, "Inferring And Tracking User Interest In Web- Based Social Network," vol. 10, no. 01, 2021.

[45] A. Petit, T. Cerqueus, A. Boutet, S. B. Mokhtar, D. Coquil, L. Brunie, and H. Kosch, "SimAttack: private web search under fire," *Journal of Internet Services and Applications*, vol. 7, no. 1, p. 2, Dec. 2016. [Online]. Available: http://jisajournal.springeropen.com/articles/10.1186/s13174-016-0044-x

[46] "Report on Domain Name Front Running," ICANN Security and Stability Advisory Committee (SSAC)., Tech. Rep. SAC 024, Feb. 2008. [Online]. Available: https://itp.cdn.icann.org/en/files/security-and-stability-advisory-committee-ssac-reports/sac-024-en.pdf

[47] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Proceedings of IEEE 36th Annual Foundations of Computer Science*. Milwaukee, WI, USA: IEEE Comput. Soc. Press, 1995, pp. 41–50. [Online]. Available: http://ieeexplore.ieee.org/document/492461/

[48] A. Beimel, Y. Ishai, and T. Malkin, "Reducing the Servers Computation in Private Information Retrieval: PIR with Preprocessing," in *Advances in Cryptology — CRYPTO 2000*, G. Goos, J. Hartmanis, J. Van Leeuwen, and M. Bellare, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, vol. 1880, pp. 55–73, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/3-540-44598-6_4

[49] W.-K. Lin, E. Mook, and D. Wichs, "Doubly Efficient Private Information Retrieval and Fully Homomorphic RAM Computation from Ring LWE," in *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*. Orlando FL USA: ACM, Jun. 2023, pp. 595–608. [Online]. Available: https://dl.acm.org/doi/10.1145/3564246.3585175

[50] H. Corrigan-Gibbs, A. Henzinger, and D. Kogan, "Single-Server Private Information Retrieval with Sublinear Amortized Time," in *Advances in Cryptology – EUROCRYPT 2022*, O. Dunkelman and S. Dziembowski, Eds. Cham: Springer International Publishing, 2022, vol. 13276, pp. 3–33, series Title: Lecture Notes in Computer Science. [Online]. Available: https://link.springer.com/10.1007/978-3-031-07085-3_1

[51] H. Corrigan-Gibbs and D. Kogan, "Private Information Retrieval with Sublinear Online Time," in *Advances in Cryptology – EUROCRYPT 2020*, A. Canteaut and Y. Ishai, Eds. Cham: Springer International Publishing, 2020, vol. 12105, pp. 44–75, series Title: Lecture Notes in Computer Science. [Online]. Available: https://link.springer.com/10.1007/978-3-030-45721-1_3

[52] H. Corrigan-Gibbs, A. Henzinger, and D. Kogan, "Single-server private information retrieval with sublinear amortized time," in *Advances in Cryptology – EUROCRYPT 2022*, O. Dunkelman and S. Dziembowski, Eds. Cham: Springer International Publishing, 2022, pp. 3–33.

[53] M. Zhou, A. Park, W. Zheng, and E. Shi, "Piano: Extremely Simple, Single-Server PIR with Sublinear Server Computation," in *2024 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, May 2024, pp. 4296–4314. [Online]. Available: https://ieeexplore.ieee.org/document/10646686/

[54] G. Persiano and K. Yeo, "Limits of preprocessing for single-server PIR," *IACR Cryptol. ePrint Arch.*, vol. 2022, p. 235, 2022. [Online]. Available: https://eprint.iacr.org/2022/235

[55] R. Ostrovsky and V. Shoup, "Private information storage (extended abstract)," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing - STOC '97*. El Paso, Texas, United States: ACM Press, 1997, pp. 294–303. [Online]. Available: http://portal.acm.org/citation.cfm?doid=258533.258606

[56] C. Adams, *Limiting Disclosure by Hiding the Attribute*. Cham: Springer International Publishing, 2021, pp. 143–173. [Online]. Available: https://doi.org/10.1007/978-3-030-81043-6_7

[57] L. Jiang and L. Ju, "FHEBench: Benchmarking Fully Homomorphic Encryption Schemes," Mar. 2022, arXiv:2203.00728 [cs]. [Online]. Available: http://arxiv.org/abs/2203.00728

[58] A. A. Badawi, A. Alexandru, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, C. Pascoe, Y. Polyakov, I. Quah, S. R.V., K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan, and V. Zucca, "OpenFHE: Open-source fully homomorphic encryption library," Cryptology ePrint Archive, Paper 2022/915, 2022, https://eprint.iacr.org/2022/915. [Online]. Available: https://eprint.iacr.org/2022/915

[59] "Server CPU Market Dynamics in 2025." [Online]. Available: https://www.fusionww.com/insights/blog/server-cpu-market-dynamics-in-2025

[60] R. R. Toledo, G. Danezis, and I. Goldberg, "Lower-Cost ϵ-Private Information Retrieval," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 4, pp. 184–201, Oct. 2016. [Online]. Available: https://petsymposium.org/popets/2016/popets-2016-0035.php

[61] C. Dwork, "Differential Privacy," in *Automata, Languages and Programming*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, vol. 4052, pp. 1–12, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/11787006_1

[62] B. Pejó and D. Desfontaines, *Guide to Differential Privacy Modifications: A Taxonomy of Variants and Extensions*, ser. SpringerBriefs in Computer Science. Cham: Springer International Publishing, 2022. [Online]. Available: https://link.springer.com/10.1007/978-3-030-96398-9

[63] D. Kifer and A. Machanavajjhala, "No free lunch in data privacy," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 193–204. [Online]. Available: https://doi.org/10.1145/1989323.1989345

[64] K. D. Albab, R. Issa, M. Varia, and K. Graffi, "Batched differentially private information retrieval," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 3327–3344. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/albab

[65] S. Patel, G. Persiano, and K. Yeo, "What Storage Access Privacy is Achievable with Small Overhead?" in *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. Amsterdam Netherlands: ACM, Jun. 2019, pp. 182–199. [Online]. Available: https://dl.acm.org/doi/10.1145/3294052.3319695

[66] OpenAI, "Equation Simplification Steps for \epsilon," Jan. 2026, prompt: "Plug the value of $\mathsf{E=\frac{MZ}{K}}$ and $\mathsf{\mathcal{L}=\frac{K}{Z}}$ and after that, simplify the following equation: \begin{gather*} \begin{aligned} \mathsf{e^{\epsilon}=\frac{\left\{\mathsf{\left(1-\frac{K}{(E-1)}\right)\times\frac{1}{Z}\times\left(1 - \frac{\mathcal{L}}{M} \right)^{\, (\mathcal{L}-1)}}\right\}+\left\{\mathsf{\frac{K}{(E-1)}\times\left(\frac{\mathcal{L}^2}{M} \right)\times \left(1 - \frac{\mathcal{L}}{M} \right)^{\, (\mathcal{L}-1)}}\right\}}{\left\{\mathsf{\frac{(\mathcal{L}\times Z -1)}{M\times Z}\times\left(1 - \frac{\mathcal{L}}{M} \right)^{\, (\mathcal{L}-1)}}\right\}+\left[\mathsf{\left\{1- \frac{(\mathcal{L}\times Z -1)}{M\times Z}\right\}\times\left(\frac{\mathcal{L}^2}{M} \right)\times \left(1 - \frac{\mathcal{L}}{M} \right)^{\, (\mathcal{L}-1)}}\right]}} \end{aligned} \end{gather*}". [Online]. Available: https://chatgpt.com/share/6967d54a-1ddc-8008-ae14-762a988a3244

[67] "Distributed Point Functions and Their Applications," in *Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 640–658, iSSN: 0302-9743, 1611-3349. [Online]. Available: http://link.springer.com/10.1007/978-3-642-55220-5_35

[68] E. Boyle, N. Gilboa, and Y. Ishai, "Function Secret Sharing: Improvements and Extensions," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. Vienna Austria: ACM, Oct. 2016, pp. 1292–1303. [Online]. Available: https://dl.acm.org/doi/10.1145/2976749.2978429

[69] J. Doerner and A. Shelat, "Scaling ORAM for Secure Computation," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Dallas Texas USA: ACM, Oct. 2017, pp. 523–535. [Online]. Available: https://dl.acm.org/doi/10.1145/3133956.3133967

[70] A. Vadapalli, R. Henry, and I. Goldberg, "Duoram: A bandwidth-efficient distributed oram for 2- and 3-party computation," in *Proceedings of the 32nd USENIX Conference on Security Symposium*, ser. SEC '23. USA: USENIX Association, 2023. [Online]. Available: https://dl.acm.org/doi/10.5555/3620237.3620456

[71] L. Braun, M. Pancholi, R. Rachuri, and M. Simkin, "Ramen: Souper Fast Three-Party Computation for RAM Programs," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. Copenhagen Denmark: ACM, Nov. 2023, pp. 3284–3297. [Online]. Available: https://dl.acm.org/doi/10.1145/3576915.3623115

[72] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985. [Online]. Available: https://doi.org/10.1109/TIT.1985.1057074

[73] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," *Journal of the ACM*, vol. 51, no. 2, pp. 231–262, Mar. 2004. [Online]. Available: https://dl.acm.org/doi/10.1145/972639.972643

[74] S. Pohlig and M. Hellman, "An improved algorithm for computing logarithms overGF(p)and its cryptographic significance (Corresp.)," *IEEE Transactions on Information Theory*, vol. 24, no. 1, pp. 106–110, Jan. 1978. [Online]. Available: https://ieeexplore.ieee.org/document/1055817

[75] OpenAI, "Equation Simplification Steps for \epsilon," Jan. 2026, prompt: "Consider a list \(L = \{ t_1, t_2, \ldots, t_K \} \), where \(t_1, t_2, \ldots, t_K \) are distinct and randomly selected elements from a \(q \)-order subgroup of \(\mathbb{Z}_{p}^{*} \), with \(q \) being a prime number. We then create another list \(L' = \{ t_1 \mod r, t_2 \mod r, \ldots, t_K \mod r \} \), where \(r \) is a smaller prime number, significantly less than \(q \). We already know that

there is no collision in L. However, due to wrapping around mod r, it might create new collisions in L'. Given that \(p \) is a 3072-bit prime, \(q \) is a 256-bit prime, \(r \) is a 64-bit prime, and \(K \) is 40960, what is the probability that this new list \(T \) will contain at least one collision?". [Online]. Available: https://chatgpt.com/share/696e921b-b834-8008-9d2b-4974fe6fac2c

[76] O. Goldreich, "Foundations of cryptography. 2: Basic applications." Cambridge: Cambridge Univ. Press, 2009, num Pages: 373.

[77] Y. Lindell, "How to Simulate It – A Tutorial on the Simulation Proof Technique," in *Tutorials on the Foundations of Cryptography*, Y. Lindell, Ed. Cham: Springer International Publishing, 2017, pp. 277–346, series Title: Information Security and Cryptography. [Online]. Available: http://link.springer.com/10.1007/978-3-319-57048-8_6

[78] P. Sumit, Kumar, "PPIR," 2025. [Online]. Available: https://github.com/sumitkumarpaul/pir.git

[79] "The GNU MP Bignum Library." [Online]. Available: https://gmplib.org/

[80] F. Wang, D. Ko, and S. Eskandarian, "Function Secret Sharing (FSS) Library," 2017. [Online]. Available: https://github.com/frankw2/libfss.git

[81] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. Cambridge Massachusetts: ACM, Jan. 2012, pp. 309–325. [Online]. Available: https://dl.acm.org/doi/10.1145/2090236.2090262

[82] kimlaine Kim Laine, T. T. Zaccai, kiromaru Radames Cruz, and W. W. Dai, "Kuku," Feb. 2021. [Online]. Available: https://github.com/microsoft/Kuku.git

[83] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan, "One server for the price of two: simple and fast single-server private information retrieval," in *Proceedings of the 32nd USENIX Conference on Security Symposium*, ser. SEC '23. USA: USENIX Association, 2023. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/3620237.3620455

[84] S. Lu and R. Ostrovsky, "Distributed Oblivious RAM for Secure Two-Party Computation," in *Theory of Cryptography*, A. Sahai, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, vol. 7785, pp. 377–396,

series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-642-36594-2_22

[85] B. Falk, R. Ostrovsky, M. Shtepel, and J. Zhang, "Gigadoram: breaking the billion address barrier," in *Proceedings of the 32nd USENIX Conference on Security Symposium*, ser. SEC '23. USA: USENIX Association, 2023. [Online]. Available: https://dl.acm.org/doi/10.5555/3620237.3620454

[86] "Cloud Interconnect FAQ." [Online]. Available: https://docs.cloud.google.com/network-connectivity/docs/interconnect/support/faq

[87] H. Berghel, ""Free" Online Services and Gonzo Capitalism," *Computer*, vol. 56, no. 7, pp. 86–92, Jul. 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10154195/

[88] C. Mascia, M. Sala, and I. Villa, "A survey on functional encryption," *Advances in Mathematics of Communications*, vol. 17, no. 5, pp. 1251–1289, 2023. [Online]. Available: https://www.aimsciences.org/article/id/69dc52a9-c02d-44df-93c5-ee85e1d20d37

[89] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov, "IRON: Functional Encryption using Intel SGX," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Dallas Texas USA: ACM, Oct. 2017, pp. 765–782, gSCC: 0000165. [Online]. Available: https://dl.acm.org/doi/10.1145/3133956.3134106

[90] "eXtensible Access Control Markup Language (XACML) Version 3.0 Plus Errata 01." [Online]. Available: https://docs.oasis-open.org/xacml/3.0/errata01/os/xacml-3.0-core-spec-errata01-os-complete.html

[91] K. Liu, Q. Wang, J. Han, and H. Wu, "A Privacy Protection Method for P2P-based Web Service Discovery," in *IEEE International Conference on e-Business Engineering (ICEBE'07)*. Hong Kong, China: IEEE, Oct. 2007, pp. 551–558, gSCC: 0000002. [Online]. Available: http://ieeexplore.ieee.org/document/4402147/

[92] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979. [Online]. Available: https://dl.acm.org/doi/10.1145/359168.359176

[93] "What Are Smart Contracts on Blockchain? | IBM," Jul. 2021. [Online]. Available: https://www.ibm.com/think/topics/smart-contracts

[94] S. Alansari, "A blockchain-based approach for secure, transparent and accountable personal data sharing," Ph.D. dissertation, University Of Southampton, Jan. 2020. [Online]. Available: https://eprints.soton.ac.uk/447633/1/Final_thesis.pdf

[95] R. Banik, "Estimating Smart Contract Costs," Dec. 2021. [Online]. Available: https://medium.com/scrappy-squirrels/estimating-smart-contract-costs-f65acf818c26

[96] E. Birrell, A. Gjerdrum, R. Van Renesse, H. Johansen, D. Johansen, and F. B. Schneider, "SGX Enforcement of Use-Based Privacy," in *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*. Toronto Canada: ACM, Jan. 2018, pp. 155–167. [Online]. Available: https://dl.acm.org/doi/10.1145/3267323.3268954

[97] P. G. Wagner, P. Birnstill, and J. Beyerer, "Distributed Usage Control Enforcement through Trusted Platform Modules and SGX Enclaves," in *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*. Indianapolis Indiana USA: ACM, Jun. 2018, pp. 85–91, gSCC: 0000010. [Online]. Available: https://dl.acm.org/doi/10.1145/3205977.3205990

[98] M. Mazmudar and I. Goldberg, "Mitigator: Privacy policy compliance using trusted hardware," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, no. 3, pp. 204–221, Jul. 2020. [Online]. Available: https://petsymposium.org/popets/2020/popets-2020-0049.php

[99] V. Costan and S. Devadas, "Intel SGX explained," Cryptology ePrint Archive, Paper 2016/086, 2016. [Online]. Available: https://eprint.iacr.org/2016/086

[100] C. Castelluccia, E. De Cristofaro, A. Francillon, and M.-A. Kaafar, "EphPub: Toward robust Ephemeral Publishing," in *2011 19th IEEE International Conference on Network Protocols*. Vancouver, AB, Canada: IEEE, Oct. 2011, pp. 165–175. [Online]. Available: http://ieeexplore.ieee.org/document/6089048/

[101] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy, "Vanish: increasing data privacy with self-destructing data," in *Proceedings of the 18th Conference on USENIX Security Symposium*, ser. SSYM'09. USA: USENIX Association, 2009, p. 299–316. [Online]. Available: https://dl.acm.org/doi/10.5555/1855768.1855787

[102] "Distributed hash table," Sep. 2025, page Version ID: 1311913768. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Distributed_hash_table&oldid=1311913768

[103] R. Geambasu, A. A. Levy, T. Kohno, A. Krishnamurthy, and H. M. Levy, "Comet: an active distributed key-value store," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. USA: USENIX Association, 2010, p. 323–336. [Online]. Available: https://dl.acm.org/doi/10.5555/1924943.1924966

[104] "Active Storage Overview." [Online]. Available: https://guides.rubyonrails.org/active_storage_overview.html

[105] H. Liu, H. Luo, S. Li, T. Dong, G. Chen, Y. Meng, and H. Zhu, "Privacy Computing with Right to Be Forgotten in Trusted Execution Environment," in *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*. Kuala Lumpur, Malaysia: IEEE, Dec. 2023, pp. 2566–2571. [Online]. Available: https://ieeexplore.ieee.org/document/10437471/

[106] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun, "Rote: rollback protection for trusted execution," in *Proceedings of the 26th USENIX Conference on Security Symposium*, ser. SEC'17. USA: USENIX Association, 2017, p. 1289–1306. [Online]. Available: https://dl.acm.org/doi/10.5555/3241189.3241289

[107] M. Gao, H. Dang, and E.-C. Chang, "TEEKAP: Self-Expiring Data Capsule using Trusted Execution Environment," in *Annual Computer Security Applications Conference*. Virtual Event USA: ACM, Dec. 2021, pp. 235–247. [Online]. Available: https://dl.acm.org/doi/10.1145/3485832.3485919

[108] R. Strackx and F. Piessens, "Ariadne: A minimal approach to state continuity," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 875–892. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/strackx

[109] B. Parno, J. R. Lorch, J. R. Douceur, J. Mickens, and J. M. McCune, "Memoir: Practical state continuity for protected modules," in *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, ser. SP '11. USA: IEEE Computer Society, 2011, p. 379–394. [Online]. Available: https://doi.org/10.1109/SP.2011.38

[110] officedocspr5, "Trusted Platform Module Technology Overview." [Online]. Available: https://learn.microsoft.com/en-us/windows/security/hardware-security/tpm/trusted-platform-module-overview

[111] R. Strackx, B. Jacobs, and F. Piessens, "ICE: a passive, high-speed, state-continuity scheme," in *Proceedings of the 30th Annual Computer Security Applications Conference*, ser. ACSAC '14. New York, NY, USA: Association for Computing Machinery, Dec. 2014, pp. 106–115. [Online]. Available: https://dl.acm.org/doi/10.1145/2664243.2664259

[112] M. van Dijk, J. Rhodes, L. F. G. Sarmenta, and S. Devadas, "Offline untrusted storage with immediate detection of forking and replay attacks," in *Proceedings of the 2007 ACM workshop on Scalable trusted computing*, ser. STC '07. New York, NY, USA: Association for Computing Machinery, Nov. 2007, pp. 41–48. [Online]. Available: https://dl.acm.org/doi/10.1145/1314354.1314364

[113] N. Karapanos, A. Filios, R. A. Popa, and S. Capkun, "Verena: End-to-End Integrity Protection for Web Applications," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 895–913, iSSN: 2375-1207. [Online]. Available: https://ieeexplore.ieee.org/document/7546541

[114] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *3rd Symposium on Operating Systems Design and Implementation (OSDI 99)*. New Orleans, LA: USENIX Association, Feb. 1999. [Online]. Available: https://www.usenix.org/conference/osdi-99/practical-byzantine-fault-tolerance

[115] M. Brandenburger, C. Cachin, M. Lorenz, and R. Kapitza, "Rollback and Forking Detection for Trusted Execution Environments Using Lightweight Collective Memory," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Denver, CO, USA: IEEE, Jun. 2017, pp. 157–168. [Online]. Available: http://ieeexplore.ieee.org/document/8023119/

[116] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun, "Rote: rollback protection for trusted execution," in *Proceedings of the 26th USENIX Conference on Security Symposium*, ser. SEC'17. USA: USENIX Association, 2017, p. 1289–1306.

[117] A. Back, "Hashcash - a denial of service counter-measure," 2002. [Online]. Available: https://api.semanticscholar.org/CorpusID:256306

[118] chrisdh79, "New Intel chips won't play Blu-ray disks due to SGX deprecation," Jan. 2022. [Online]. Available: https://www.reddit.com/r/gadgets/comments/s3xhkf/new_intel_chips_wont_play_bluray_disks_due_to_sgx/

[119] PRNewswire, "Confidential Computing Market worth $59.4 billion by 2028 - Exclusive Report by MarketsandMarkets™." [Online]. Available: https://www.benzinga.com/pressreleases/23/06/n32692290/confidential-computing-market-worth-59-4-billion-by-2028-exclusive-report-by-marketsandmarkets

[120] M. Pei, H. Tschofenig, D. Thaler, and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture," Internet Engineering Task Force, Request for Comments RFC 9397, Jul. 2023, num Pages: 31. [Online]. Available: https://datatracker.ietf.org/doc/rfc9397

[121] "Ethereum.org: The complete guide to Ethereum." [Online]. Available: https://ethereum.org/

[122] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," Internet Engineering Task Force, Request for Comments RFC 5280, May 2008, num Pages: 151. [Online]. Available: https://datatracker.ietf.org/doc/rfc5280

[123] S. Farrell, R. Housley, and S. Turner, "An Internet Attribute Certificate Profile for Authorization," RFC Editor, Tech. Rep. RFC5755, Jan. 2010. [Online]. Available: https://www.rfc-editor.org/info/rfc5755

[124] T. Knauth, M. Steiner, S. Chakrabarti, L. Lei, C. Xing, and M. Vij, "Integrating Remote Attestation with Transport Layer Security," Jul. 2019, arXiv:1801.05863 [cs]. [Online]. Available: http://arxiv.org/abs/1801.05863

[125] R. Walther, C. Weinhold, and M. Roitzsch, "Ratls: Integrating transport layer security with remote attestation," in *Applied Cryptography and Network Security Workshops: ACNS 2022 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, SiMLA, Rome, Italy, June 20–23, 2022, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 361–379. [Online]. Available: https://doi.org/10.1007/978-3-031-16815-4_20

[126] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for cpu based attestation and sealing," 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:14218854

[127] Nist, "Harvest Now, Decrypt Later (HNDL): The Quantum-Era Threat." [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/harvest-now-decrypt-later-hndl

[128] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. Stockholm, Sweden: IEEE, Jun. 2019, pp. 185–200. [Online]. Available: https://ieeexplore.ieee.org/document/8806762/

[129] R. Canetti, "Universally composable security: a new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. Newport Beach, CA, USA: IEEE, 2001, pp. 136–145. [Online]. Available: https://ieeexplore.ieee.org/document/959888/

[130] "Intel SGX SDK for Linux* OS." [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/linux-overview.html

[131] "INFURA JSON-RPC methods." [Online]. Available: https://docs.infura.io/api/networks/ethereum/json-rpc-methods

[132] K. H. Han, W.-K. Lee, A. Karmakar, J. M. B. Mera, and S. O. Hwang, "cuFE: High Performance Privacy Preserving Support Vector Machine With Inner-Product Functional Encryption," *IEEE Transactions on Emerging Topics in Computing*, vol. 12, no. 1, pp. 328–343, Jan. 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10089390/

[133] D. Demmler, T. Schneider, and M. Zohner, "ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation," in *Proceedings 2015 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2015. [Online]. Available: https://www.ndss-symposium.org/ndss2015/ndss-2015-programme/aby---framework-efficient-mixed-protocol-secure-two-party-computation/

[134] "CiFEr - Functional Encryption library." [Online]. Available: https://github.com/fentec-project/CiFEr

[135] "Apps that track you and their alternatives." [Online]. Available: https://surfshark.com/apps-that-track-you

[136] F. Li, X. Li, and M. Gao, "Secure MLaaS with Temper: Trusted and Efficient Model Partitioning and Enclave Reuse," in *Annual Computer Security Applications Conference*. Austin TX USA: ACM, Dec. 2023, pp. 621–635. [Online]. Available: https://dl.acm.org/doi/10.1145/3627106.3627145

[137] "How Much Does a SOC 2 Audit Cost in 2025?" [Online]. Available: https://secureframe.com/hub/soc-2/audit-cost

[138] H. Malhotra, "Trade secret in intellectual property," Rochester, NY, Mar. 2021. [Online]. Available: https://papers.ssrn.com/abstract=3796211

[139] "Trade Secrets and Confidential Business Information," Mar. 2014. [Online]. Available: https://epthinktank.eu/2014/03/19/trade-secrets-and-confidential-business-information/

[140] T. Hulsen, "Sharing is caring—data sharing initiatives in healthcare," *International Journal of Environmental Research and Public Health*, vol. 17, no. 9, 2020. [Online]. Available: https://www.mdpi.com/1660-4601/17/9/3046

[141] "The Next Generation of Data-Sharing in Financial Services: Using Privacy Enhancing Techniques to Unlock New Value," Sep. 2019. [Online]. Available: https://www3.weforum.org/docs/WEF_Next_Gen_Data_Sharing_Financial_Services.pdf

[142] L. Goasduff, "Data Sharing Is a Business Necessity to Accelerate Digital Business," 2021. [Online]. Available: https://www.gartner.com/smarterwithgartner/data-sharing-is-a-business-necessity-to-accelerate-digital-business

[143] "TCF – Transparency & Consent Framework - IAB Europe." [Online]. Available: https://iabeurope.eu/transparency-consent-framework/

[144] J. Iyilade and J. Vassileva, "P2U: A Privacy Policy Specification Language for Secondary Data Sharing and Usage," in *2014 IEEE Security and Privacy Workshops*. San Jose, CA: IEEE, May 2014, pp. 18–22. [Online]. Available: http://ieeexplore.ieee.org/document/6957279/

[145] D. Giffin, A. Levy, D. Stefan, D. Terei, D. Mazières, J. Mitchell, and A. Russo, "Hails: Protecting data privacy in untrusted web applications," *Journal of Computer Security*, vol. 25, no. 4-5, pp. 427–461, Jul. 2017. [Online]. Available: https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/JCS-15801

[146] B. Andow, S. Y. Mahmud, J. Whitaker, W. Enck, B. Reaves, K. Singh, and S. Egelman, "Actions speak louder than words: Entity-Sensitive privacy policy and data flow analysis with PoliCheck," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 985–1002. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/andow

[147] Z. Noorian, J. Iyilade, M. Mohkami, and J. Vassileva, "Trust mechanism for enforcing compliance to secondary data use contracts," in *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, 2014, pp. 519–526. [Online]. Available: https://doi.org/10.1109/TrustCom.2014.66

[148] S. Li, J. Han, D. Tong, and J. Cui, "Redactable signature-based public auditing scheme with sensitive data sharing for cloud storage," *IEEE Systems Journal*, vol. 16, no. 3, pp. 3613–3624, 2022. [Online]. Available: https://doi.org/10.1109/JSYST.2022.3159832

[149] B. Curtis, "Technical competency gaps in 151,000 IT auditors in the audit industry," Nov. 2022. [Online]. Available: https://www.securitymagazine.com/articles/98555-technical-competency-gaps-in-151-000-it-auditors-in-the-audit-industry

[150] I. Kunz, K. Weiss, A. Schneider, and C. Banse, "Privacy Property Graph: Towards Automated Privacy Threat Modeling via Static Graph-based Analysis," *Proceedings on Privacy Enhancing Technologies*, vol. 2023, no. 2, pp. 171–187, Apr. 2023. [Online]. Available: https://petsymposium.org/popets/2023/popets-2023-0046.php

[151] F. Tang and B. M. Østvold, "Assessing software privacy using the privacy flow-graph," in *Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security*. Singapore Singapore: ACM, Nov. 2022, pp. 7–15. [Online]. Available: https://dl.acm.org/doi/10.1145/3549035.3561185

[152] S. Zimmeck and S. M. Bellovin, "Privee: an architecture for automatically analyzing web privacy policies," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, ser. SEC'14. USA: USENIX Association, 2014, p. 1–16. [Online]. Available: https://dl.acm.org/doi/10.5555/2671225.2671226

[153] S. Zimmeck, Z. Wang, L. Zou, R. Iyengar, B. Liu, F. Schaub, S. Wilson, N. Sadeh, S. M. Bellovin, and J. Reidenberg, "Automated Analysis of Privacy Requirements for Mobile Apps," in *Proceedings 2017 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2017. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/automated-analysis-privacy-requirements-mobile-apps/

[154] L. Yu, T. Zhang, X. Luo, and L. Xue, "AutoPPG: Towards Automatic Generation of Privacy Policy for Android Applications," in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile*

Devices. Denver Colorado USA: ACM, Oct. 2015, pp. 39–50. [Online]. Available: https://dl.acm.org/doi/10.1145/2808117.2808125

[155] S. Zimmeck, R. Goldstein, and D. Baraka, “PrivacyFlash Pro: Automating Privacy Policy Generation for Mobile Apps,” in *Proceedings 2021 Network and Distributed System Security Symposium*. Virtual: Internet Society, 2021. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/ndss2021_7C-3_24100_paper.pdf

[156] G. C. Necula, “Proof-Carrying Code. Design and Implementation,” in *Proof and System-Reliability*, H. Schwichtenberg and R. Steinbrüggen, Eds. Dordrecht: Springer Netherlands, 2002, pp. 261–288. [Online]. Available: http://link.springer.com/10.1007/978-94-010-0413-8_8

[157] R. Krahn, B. Trach, A. Vahldiek-Oberwagner, T. Knauth, P. Bhatotia, and C. Fetzer, “P esos: policy enhanced secure object store,” in *Proceedings of the Thirteenth EuroSys Conference*. Porto Portugal: ACM, Apr. 2018, pp. 1–17. [Online]. Available: https://dl.acm.org/doi/10.1145/3190508.3190518

[158] S. Tokas, O. Owe, and T. Ramezanifarkhani, “Static checking of gdpr-related privacy compliance for object-oriented distributed systems,” *Journal of Logical and Algebraic Methods in Programming*, vol. 125, p. 100733, 2022. [Online]. Available: https://doi.org/10.1016/j.jlamp.2021.100733

[159] J. M. del Álamo, D. S. Guaman, B. Gutiérrez, and A. Díez, “A systematic mapping study on automated analysis of privacy policies,” *Computing*, vol. 104, no. 9, pp. 2053–2076, 2022. [Online]. Available: https://doi.org/10.1007/s00607-022-01076-3

[160] S. Manandhar, K. Singh, and A. Nadkarni, “Towards automated regulation analysis for effective privacy compliance,” in *Proceedings of the 2024 Network and Distributed System Security (NDSS) Symposium*. The Internet Society, 2024, aRC framework for structuring regulatory text and extracting tuples for compliance analysis. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/towards-automated-regulation-analysis-for-effective-privacy-compliance/

[161] D. Slamanig and D. David, “Overview of Functional and Malleable Signature Schemes,” Jul. 2015. [Online]. Available: https://www.prismacloud.eu/PRISMACLOUD-D4.4-Overviewof-Functional-and-Malleable-Signature-Schemes.pdf

[162] P. Sumit, Kumar, "Bidirectional Privacy," Sep. 2025. [Online]. Available: https://github.com/sumitkumarpaul/bidirectional_privacy.git

[163] "DCsv3 and DCdsv3-series," Jan. 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-machines/dcv3-series

[164] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-sgx: a practical library os for unmodified applications on sgx," in *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '17. USA: USENIX Association, 2017, p. 645–658. [Online]. Available: https://dl.acm.org/doi/10.5555/3154690.3154752

[165] "Gramine Library OS with Intel SGX Support," 2023. [Online]. Available: https://github.com/gramineproject/gramine

[166] "Mbed TLS," 2023. [Online]. Available: https://github.com/Mbed-TLS/mbedtls

[167] H. M. Zum Felde, M. Morbitzer, and J. Schutte, "Securing Remote Policy Enforcement by a Multi-Enclave based Attestation Architecture," in *2021 IEEE 19th International Conference on Embedded and Ubiquitous Computing (EUC)*. Shenyang, China: IEEE, Oct. 2021, pp. 102–108, gSCC: 0000002. [Online]. Available: https://ieeexplore.ieee.org/document/9741981/

[168] D. Derler, S. Krenn, and D. Slamanig, "Signer-Anonymous Designated-Verifier Redactable Signatures for Cloud-Based Data Sharing," in *Cryptology and Network Security*, S. Foresti and G. Persiano, Eds. Cham: Springer International Publishing, 2016, vol. 10052, pp. 211–227, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-319-48965-0_13

[169] "XMLRSS - A Java Crypto Provider for Redactable Signatures," 2018. [Online]. Available: https://github.com/woefe/xmlrss

[170] L. Cranor, "P3P: making privacy policies more useful," *IEEE Security & Privacy*, vol. 1, no. 6, pp. 50–55, Nov. 2003. [Online]. Available: https://ieeexplore.ieee.org/document/1253568/

[171] "Privacy Bird: Find web sites that respect your privacy." [Online]. Available: http://www.privacybird.org/

[172] S. A. Horstmann, S. Domiks, M. Gutfleisch, M. Tran, Y. Acar, V. Moonsamy, and A. Naiakshina, ""Those things are written by lawyers, and programmers are reading

that." Mapping the Communication Gap Between Software Developers and Privacy Experts," *Proceedings on Privacy Enhancing Technologies*, vol. 2024, no. 1, pp. 151–170, Jan. 2024. [Online]. Available: https://petsymposium.org/popets/2024/popets-2024-0010.php

[173] T. Müller-Tribbensee, "Privacy Promise Vs. Tracking Reality in Pay-or-Tracking Walls," in *Privacy Technologies and Policy*, M. Jensen, C. Lauradoux, and K. Rannenberg, Eds. Cham: Springer Nature Switzerland, 2024, vol. 14831, pp. 168–188, series Title: Lecture Notes in Computer Science. [Online]. Available: https://link.springer.com/10.1007/978-3-031-68024-3_9

[174] I. Wagner, "Privacy Policies across the Ages: Content of Privacy Policies 1996–2021," *ACM Transactions on Privacy and Security*, vol. 26, no. 3, pp. 1–32, Aug. 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3590152

[175] "General Data Protection Regulation GDPR," 2016. [Online]. Available: https://gdpr-info.eu/

[176] "The Personal Information Protection and Electronic Documents Act (PIPEDA)," 2019. [Online]. Available: https://www.priv.gc.ca/en/privacy-topics/privacy-laws-in-canada/the-personal-information-protection-and-electronic-documents-act-pipeda/

[177] "California Consumer Privacy Act (CCPA)," 2024. [Online]. Available: https://oag.ca.gov/privacy/ccpa

[178] "CYBERSECURITY NEEDS FOR SMES," *Issues In Information Systems*, 2024. [Online]. Available: https://iacis.org/iis/2024/1_iis_2024-235-246.pdf

[179] J. M. Del Alamo, D. S. Guaman, B. García, and A. Diez, "A systematic mapping study on automated analysis of privacy policies," *Computing*, vol. 104, no. 9, pp. 2053–2076, Sep. 2022. [Online]. Available: https://link.springer.com/10.1007/s00607-022-01076-3

[180] "Hidden GDPR Compliance Expenses," Jun. 2025, section: Governance & Compliance. [Online]. Available: https://cybersierra.co/blog/hidden-gdpr-compliance-expenses/

[181] "GDPR audit cost: A guide - Thoropass." [Online]. Available: https://www.thoropass.com/blog/gdpr-audit-cost-a-guide

[182] "Developments from California: AG Estimates Costs of CCPA Compliance as CCPA Authors Contemplate Round II | News & Resources | Dorsey." [Online]. Available: https://www.dorsey.com/newsresources/publications/client-alerts/2019/12/ccpa-update

[183] "CCPA audit cost: A guide - Thoropass." [Online]. Available: https://www.thoropass.com/blog/ccpa-audit-cost-a-guide

[184] M. Gracy, "The Complete Guide to PIPEDA Compliance," Jan. 2025. [Online]. Available: https://sprinto.com/blog/pipeda-compliance/

[185] "Salary: Chief Privacy Officer (Dec, 2025) United States." [Online]. Available: https://www.ziprecruiter.com/Salaries/Chief-Privacy-Officer-Salary

[186] usfhealthonline, "What Is a Chief Privacy Officer? | Job Description & Salary," Aug. 2018. [Online]. Available: https://www.usfhealthonline.com/resources/health-informatics/chief-privacy-officer-job-description-salary/

[187] A. Saxena, "Compliance Q&A: How much does GDPR compliance cost?" Nov. 2024. [Online]. Available: https://sprinto.com/blog/gdpr-compliance-cost/

[188] S. Pan, D. Zhang, M. Staples, Z. Xing, J. Chen, X. Xu, and T. Hoang, "Is it a trap? a large-scale empirical study and comprehensive assessment of online automated privacy policy generators for mobile apps," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 5681–5698. [Online]. Available: https://dl.acm.org/doi/10.5555/3698900.3699218

[189] S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena, "Preventing page faults from telling your secrets," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 317–328. [Online]. Available: https://doi.org/10.1145/2897845.2897885

[190] C. Liu, A. Harris, M. Maas, M. Hicks, M. Tiwari, and E. Shi, "Ghostrider: A hardware-software system for memory trace oblivious computation," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 87–101. [Online]. Available: https://doi.org/10.1145/2694344.2694385

[191] M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz, and D. Song, “Phantom: practical oblivious computation in a secure processor,” in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS ’13. New York, NY, USA: Association for Computing Machinery, 2013, p. 311–324. [Online]. Available: https://doi.org/10.1145/2508859.2516692

[192] R. Dingledine, N. Mathewson, and P. Syverson, “Tor: The Second-Generation onion router,” in *13th USENIX Security Symposium (USENIX Security 04)*. San Diego, CA: USENIX Association, Aug. 2004. [Online]. Available: https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router

[193] M. Manulis and J. Nguyen, “Fully Homomorphic Encryption Beyond IND-CCA1 Security: Integrity Through Verifiability,” in *Advances in Cryptology – EUROCRYPT 2024*, M. Joye and G. Leander, Eds. Cham: Springer Nature Switzerland, 2024, pp. 63–93.

[194] C. Adams, *Introduction to Privacy Enhancing Technologies: A Classification-Based Approach to Understanding PETs*. Cham: Springer International Publishing, 2021. [Online]. Available: https://link.springer.com/10.1007/978-3-030-81043-6

[195] S. K. Paul and D. A. Knox, “A Taxonomy and Gap-Analysis in Digital Privacy Education,” in *Foundations and Practice of Security*, G.-V. Jourdan, L. Mounier, C. Adams, F. Sèdes, and J. Garcia-Alfaro, Eds. Cham: Springer Nature Switzerland, 2023, pp. 221–235.

[196] R. Zhou and Z. Lin, “An Improved Exponential ElGamal Encryption Scheme with Additive Homomorphism,” in *2022 International Conference on Blockchain Technology and Information Security (ICBCTIS)*. Huaihua City, China: IEEE, Jul. 2022, pp. 25–27. [Online]. Available: https://ieeexplore.ieee.org/document/9845122/

[197] A. Kirsch, M. Mitzenmacher, and U. Wieder, “More robust hashing: Cuckoo hashing with a stash,” in *Algorithms - ESA 2008*, D. Halperin and K. Mehlhorn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 611–622. [Online]. Available: https://dl.acm.org/doi/10.5555/1957995.1958011

[198] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, “VC3: Trustworthy Data Analytics in the Cloud Using SGX,” in *2015 IEEE Symposium on Security and Privacy*. San Jose, CA: IEEE, May 2015, pp. 38–54, gSCC: 0000755. [Online]. Available: https://ieeexplore.ieee.org/document/7163017/

[199] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Internet Engineering Task Force, Request for Comments RFC 8446, Aug. 2018, num Pages: 160. [Online]. Available: https://datatracker.ietf.org/doc/rfc8446

[200] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, Apr. 1984. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0022000084900709

[201] J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. Pietzuch, "Teechain: a secure payment network with asynchronous blockchain access," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. Huntsville Ontario Canada: ACM, Oct. 2019, pp. 63–79. [Online]. Available: https://dl.acm.org/doi/10.1145/3341301.3359627

Appendix A

Notation

Table A.1: Notation regarding *RouterORAM*

Notation	Description
N	Total number of distinct outsourced blocks, same as the number of leaves.
L	Height of the binary tree ($\lfloor \log \mathsf{N} \rfloor + 1$).
a	Identifier of a block, $\mathsf{a} \in [1, \mathsf{N}]$.
$\mathbb{D}(\mathsf{a})$	Data of the block, identified by a (or block a).
$\#(\mathsf{a})$	Number of replicas of block a, stored in the server.
$\#^{\mathrm{Re}}$	Total number of real block replicas present in the tree, $\#^{\mathrm{Re}} = \sum_{\mathsf{a} \in [1,\mathsf{N}]} \#(\mathsf{a})$
Z	Number of slots in a bucket. Each slot can hold an encrypted block.
β_{b}	The bucket having label b, where $\mathsf{b} \in [1, 2^{\mathsf{L}-1}]$.
$\beta_{\mathsf{b}}[\mathsf{i}]$	Content of the ith slot of β_{b}.
$\overline{(\cdot)}$	Ciphertext of $(\cdot)$ generated under the fully homomorphic encryption, FHE.
$\overline{\mathsf{C}} \leftarrow \overline{\mathsf{A}} \boxdot \overline{\mathsf{B}}$	FHE-ciphertext $\overline{\mathsf{C}}$ is produced after homomorphically evaluating the binary operation $\boxdot$ between FHE-ciphertext $\overline{\mathsf{A}}$ and FHE-ciphertext $\overline{\mathsf{B}}$.
$\mathsf{e}_{\hat{\mathsf{b}},\check{\mathsf{b}}}$	Edge connecting the upper level bucket $\beta_{\hat{\mathsf{b}}}$ with its child $\beta_{\check{\mathsf{b}}}$.
$\mathsf{blk.m.a}$	Block metadata part storing the block identifier, $\mathsf{blk.m.a} \in [1, N]$.
$\mathsf{blk.m.x}$	Block metadata part storing the mapped leaf label, $\mathsf{blk.m.x} \in [2^{\mathsf{L}-1}, 2^{\mathsf{L}} - 1]$.
$\mathsf{pos}[\mathsf{a}] := \langle \mathsf{b}_1, .., \mathsf{b}_{\#(\mathsf{a})} \rangle$	The content of the client-local position map corresponding to the block a. It is a list holding the details of all the replicas of the block a.

Table A.2: Notation regarding the PIR schemes

Notation	Description
$\mathsf{S}_\alpha, \mathsf{S}_\beta, \mathsf{S}_\gamma$	Participating servers in the protocol
$\mathbb{C}$	PIR client
N	Total number of blocks in the public database
B	Size of each block
I_i	Block index of the ith request
$\mathbb{D}[\mathsf{I}_\mathsf{i}]$	Plaintext content of block I_i
$\widetilde{\mathbb{D}}$	The shuffled and encrypted database

Continued on next page...

Notation	Description
M	Total number of buckets in the shuffled database
$\mathsf{pk_F} - \mathsf{sk_F}$	FHE keypair
$\widetilde{\mathbb{D}}[\mathsf{L}]$	L^{th} bucket in the shuffled database, where $0 \leq \mathsf{L} \leq (\mathsf{M}-1)$
$\mathsf{L_i}$	The touched bucket location during i^{th} request
K	Number of elements within the shelter
$\mathsf{sh^i}[]$	The shelter state after processing i^{th} PIR request
$\pi_\Phi[]$	The dummy position mapping
$\overline{(\cdot)}$	FHE generated ciphertext of $(\cdot)$
$\overline{\mathsf{C}} \leftarrow \overline{\mathsf{A} \boxdot \mathsf{B}}$	FHE-ciphertext $\overline{\mathsf{C}}$ is produced after homomorphically evaluating the binary operation $\boxdot$ between $\overline{\mathsf{A}}$ and $\overline{\mathsf{B}}$
$\mathcal{H}$	Hash function determining the shuffling
$\langle \mathsf{e}, \mathsf{s} \rangle$	Key of $\mathcal{H}$
	Notation specific to *DP-Variant Scheme*
E	Number of requests processed within an epoch
Z	Number of slots within a bucket
$\widetilde{\mathbb{D}}[\mathsf{L}][\mathsf{j}]$	Content of the jth slot of $\widetilde{\mathbb{D}}[\mathsf{L}]$
ϵ	The multiplicative term in differential privacy. See [61] for details.
δ	The additive term in differential privacy. See [61] for details.
	Notation specific to *DPF-Variant Scheme*
$\mathsf{S}_\delta, \mathsf{S}_\epsilon$	Two additional participating servers (along with $\mathsf{S}_\alpha, \mathsf{S}_\beta, \mathsf{S}_\gamma$)
p	A $\lVert\mathsf{p}\rVert$-bit prime number
q	A $\lVert\mathsf{q}\rVert$-bit safe prime number
$\mathbb{G}$	A cyclic subgroup of $\mathbb{Z}_\mathsf{p}^*$ of order q
g	Generator of $\mathbb{G}$
$\mathcal{E}(\cdot)$	Ciphertext of $(\cdot)$ encrypted under $\mathcal{E}$; which is the El-Gamal encryption scheme working in $\mathbb{G}$
$\mathsf{pk}_\mathcal{E} - \mathsf{sk}_\mathcal{E}$	Keypair of $\mathcal{E}$
$\mathcal{E}_\mathsf{q}(\cdot)$	Ciphertext of $(\cdot)$ encrypted under $\mathcal{E}_\mathsf{q}$; which is the El-Gamal encryption scheme working in $\mathbb{Z}_\mathsf{q}^*$
$\mathsf{pk}_{\mathcal{E}_\mathsf{q}} - \mathsf{sk}_{\mathcal{E}_\mathsf{q}}$	Keypair of $\mathcal{E}_\mathsf{q}$
$\widetilde{\mathbb{D}}_\alpha, \widetilde{\mathbb{D}}_\gamma$	Secret-shared shuffled databases in S_α and S_γ, respectively
$\mathcal{H}_\alpha, \mathcal{H}_\gamma$	Cuckoo hash functions used by S_α and S_γ, respectively
$\langle\cdot\rangle_\alpha, \langle\cdot\rangle_\gamma$	Secret share of $(\cdot)$, held by S_α and S_γ, respectively

Continued on next page...

Notation	Description
$\mathcal{K}_\alpha, \mathcal{K}_\gamma$	DPF-keys, used by S_α and S_γ, respectively
$\mathsf{T}_{\mathsf{I}_\mathsf{i}}$	Shuffled database tag corresponding to index I_i
ρ	Secret parameter of the tag-generating one-way function
T_ϕ	A dummy tag in shuffled database
Φ	Set of all the dummy tags within the shuffled database.
T_*	Selected tag for the shuffled database. It could be either $\mathsf{T}_{\mathsf{I}_\mathsf{i}}$ or T_ϕ
$\widehat{\mathsf{T}_{\mathsf{I}_\mathsf{i}}}$	Shelter tag corresponding to index I_i
$\mathsf{a}, \mathsf{b}, \mathsf{c}$	Random members of $\mathbb{G}$ chosen by $\mathsf{S}_\alpha, \mathsf{S}_\beta, \mathsf{S}_\gamma$ respectively; refreshed per request
$\widehat{\mathsf{t}_{\mathsf{I}_\mathsf{i}}}$	Short shelter tag corresponding to index I_i
r	A $\|\mathsf{r}\|$-bit prime number for generating $\widehat{\mathsf{t}_{\mathsf{I}_\mathsf{i}}}$ from $\widehat{\mathsf{T}_{\mathsf{I}_\mathsf{i}}}$
$\mathcal{M}$	The mask database, containing $\sqrt{\mathsf{N}}$-random masks
$\mathsf{sh}[\mathsf{k}] \triangleright \ddot{\mathsf{d}}$	Data part of the kth shelter element, which is a masked response
$\mathsf{sh}[\mathsf{k}] \triangleright \widehat{\mathsf{T}}$	Shelter tag of the kth shelter element
$\mathsf{sh}[\mathsf{k}] \triangleright \widehat{\mathsf{t}}$	Short shelter tag of the kth shelter element

Table A.3: Notation regarding *ROT*

Notation	**Description**
DO	Data owner or the service user
DU	Data user or the service provider
D	Data owner's personal data
DI	Data issuer, who issued D to DO
$SEAL_D$	Sealed version of D
$\mathbb{T}_\mathsf{L}$	Data retention time limit, specified by DO
$\mathcal{C}$	The computation to be performed on the personal data
$Eclv_\mathcal{C}$	The enclave which performs $\mathcal{C}$ on the personal data
$Eclv_\mathcal{C}^{DO}$	$Eclv_\mathcal{C}$, specifically signed by DO
$\mathcal{C}(D)$	Computation result generated by performing $\mathcal{C}$ on D
$\mathsf{Lib}_\mathsf{Enc}$	The trusted library, which contains the data processing enclaves
BC	Blockchain

Table A.4: Notation regarding *BPPM*

Notation	**Description**
DO	Data owner/service user
$DU_{i,j}$	j^{th} Data user/service provider of the ith layer
tree$(DU_{i,j})$	Data usage sub-tree rooted at $DU_{i,j}$. **tree**$(DU_{1,1})$ denotes the entire data usage tree.
parent$(DU_{i,j})$	The parent node of $DU_{i,j}$ within the data usage tree.
children$(DU_{i,j})$	The set of child nodes of $DU_{i,j}$ within the data usage tree.
CP	Code provider
$PDS_{i,j}$	Collection of proposed data processing statements of $DU_{i,j}$. It contains all the data processing statements of **tree**$(DU_{i,j})$ and is specified within the privacy policy document of $DU_{i,j}$.
PDS_{DO}	Collection of data processing statements agreed by DO.
DS_{DO}	Data set corresponding to PDS_{DO}.
$Prog_{Eclv}$	The trusted base code of the data user's enclave.
$LPDS_{i,j}$	A subset of $PDS_{i,j}$. $DU_{i,j}$ performs these data processing locally.
D	Structure through which personal data is sent. Details in Fig. 5.3.
PC	Structure through which processing consent is sent. Details in Fig. 5.3.

Appendix B

Distributed Point Function

A point function ($\mathsf{f_p}$) is defined as a function that evaluates to zero at every point in its domain, except at a specific target point (x^*), where it evaluates to a non-zero target value (Y^*). Distributed Point Functions (DPFs) provide an efficient means for sharing a point function between two or more evaluators. In a DPF context, although all the parties know the entire function's domain, all the evaluators remain unaware of x^* and Y^*. However, they can compute a secret-share of the output of $\mathsf{f_p}$, at any point in its domain.

A DPF consists of a pair of algorithms, $\mathsf{Gen}()$ and $\mathsf{Eval}()$. A trusted party, often referred to as a dealer, possesses the secret point function, $\mathsf{f_p}$ and executes the Gen algorithm to produce keys. For a two-party DPF, the dealer generates two keys, $(\mathcal{K}_0, \mathcal{K}_1)$. These keys are then provided to two evaluator parties, $\mathsf{p_0}$ and $\mathsf{p_1}$, respectively. Who individually utilize these keys to run the $\mathsf{Eval}()$ algorithm on any point within the function's domain, to generate secret shares of the actual point function. DPF meets the following correctness definition for all points x in the domain of $\mathsf{f_p}$:

$$\mathsf{Eval}(\mathcal{K}_0, \mathsf{x}) + \mathsf{Eval}(\mathcal{K}_1, \mathsf{x}) = \begin{cases} \mathsf{Y}*, & \text{if } \mathsf{x} = \mathsf{x}^* \\ 0 & \text{otherwise} \end{cases}$$

Regarding security, DPF meets the following simulatability definition: There exists a PPT simulator, $\mathsf{Sim}^{\mathsf{DPF}}$, such that, for any point function, $\mathsf{f_p}$, and bit $\mathsf{b} \in \{0, 1\}$, the distribution ensembles $\{\mathsf{Sim}^{\mathsf{DPF}}(1^\lambda, \mathsf{b})\}$ and $\{\mathcal{K}_\mathsf{b} | (\mathcal{K}_0, \mathcal{K}_1) \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{f_p})\}$ are computationally indistinguishable [67]. Furthermore, $\forall_\mathsf{x} \mathsf{Eval}(\mathcal{K}_\mathsf{b}, \mathsf{x})$ is indistinguishable from a randomly chosen element in the range of $\mathsf{Eval}()$. In other words, individual evaluators gain no information about $\mathsf{f_p}$ by observing either the key or the output of the evaluation algorithm.

Appendix C

Homomorphic Properties of El-Gamal Encryption

The El-Gamal encryption scheme has multiplicative homomorphic property [196]. Specifically, if $\mathsf{m}_1, \mathsf{m}_2 \in \mathbb{G}$ are two plaintext messages and $\mathcal{E}(\mathsf{m}_1)$ and $\mathcal{E}(\mathsf{m}_2)$ are their El-Gamal ciphertexts, then $\mathcal{E}(\mathsf{m}_1).\mathcal{E}(\mathsf{m}_1) = \mathcal{E}(\mathsf{m}_1.\mathsf{m}_2)$. This is true because, if $\mathsf{x} \in \mathbb{Z}_\mathsf{q}$ is the secret key and $(\mathsf{g}, \mathsf{p}, \mathsf{h} = \mathsf{g}^\mathsf{x})$ is the public-key of $\mathcal{E}$, then $\mathcal{E}(\mathsf{m}_1) = (\mathsf{g}^{\mathsf{r}_1}, \mathsf{m}_1\mathsf{h}^{\mathsf{r}_1})$ and $\mathcal{E}(\mathsf{m}_2) = (\mathsf{g}^{\mathsf{r}_2}, \mathsf{m}_2\mathsf{h}^{\mathsf{r}_2})$, where $\mathsf{r}_1, \mathsf{r}_2 \in \mathbb{Z}_\mathsf{q}$ are the randomness chosen during the encryption. Hence,

$$\begin{aligned}
&\mathcal{E}(\mathsf{m}_1).\mathcal{E}(\mathsf{m}_2) = (\mathsf{g}^{\mathsf{r}_2}, \mathsf{m}_2\mathsf{h}^{\mathsf{r}_2}).(\mathsf{g}^{\mathsf{r}_2}, \mathsf{m}_2\mathsf{h}^{\mathsf{r}_2}) = (\mathsf{g}^{\mathsf{r}_1}.\mathsf{g}^{\mathsf{r}_2}, \mathsf{m}_1\mathsf{h}^{\mathsf{r}_1}.\mathsf{m}_2\mathsf{h}^{\mathsf{r}_2}) \\
&= (\mathsf{g}^{\mathsf{r}_1+\mathsf{r}_2}, \mathsf{m}_1\mathsf{m}_2\mathsf{h}^{\mathsf{r}_1+\mathsf{r}_2})
\end{aligned}$$

While decrypting, it will produce:

$$\begin{aligned}
&\mathcal{E}.\mathsf{Dec}((\mathsf{g}^{\mathsf{r}_1+\mathsf{r}_2}, \mathsf{m}_1\mathsf{m}_2\mathsf{h}^{\mathsf{r}_1+\mathsf{r}_2}), \mathsf{x}) = (\mathsf{m}_1\mathsf{m}_2\mathsf{h}^{\mathsf{r}_1+\mathsf{r}_2}).(\mathsf{g}^{\mathsf{r}_1+\mathsf{r}_2})^{-\mathsf{x}} \\
&= \mathsf{m}_1\mathsf{m}_2\mathsf{g}^{\mathsf{x}(\mathsf{r}_1+\mathsf{r}_2)}.\mathsf{g}^{-\mathsf{x}(\mathsf{r}_1+\mathsf{r}_2)} = \mathsf{m}_1\mathsf{m}_2
\end{aligned}$$

Moreover, the El-Gamal encryption scheme also has a *semi*-homomorphic exponentiation property. Which means, an evaluator, having an El-Gamal ciphertext $\mathcal{E}(\mathsf{m}) = (\mathsf{g}^\mathsf{r}, \mathsf{mh}^\mathsf{r})$ and a *plaintext* exponent, $\mathsf{e} \in \mathbb{Z}_\mathsf{q}$, can produce $\mathcal{E}(\mathsf{m}^\mathsf{e})$, without requiring the decryption of $\mathcal{E}(\mathsf{m})$. This is possible by raising both the components of the ciphertext to the plaintext e. This means:

$$\mathcal{E}'(\mathsf{m}) = \mathcal{E}(\mathsf{m})^\mathsf{e} = ((\mathsf{g}^\mathsf{r})^\mathsf{e}, (\mathsf{mh}^\mathsf{r})^\mathsf{e}) = (\mathsf{g}^{\mathsf{re}}, \mathsf{m}^\mathsf{e}\mathsf{h}^{\mathsf{re}}).$$

While decrypting, it will produce:

$$\mathcal{E}.\mathsf{Dec}((\mathsf{g}^{\mathsf{re}}, \mathsf{m}^{\mathsf{e}}\mathsf{h}^{\mathsf{re}}), \mathsf{x}) = (\mathsf{m}^{\mathsf{e}}\mathsf{h}^{\mathsf{re}}).(\mathsf{g}^{\mathsf{re}})^{-\mathsf{x}}$$
$$= (\mathsf{m}^{\mathsf{e}}(\mathsf{g}^{\mathsf{x}})^{\mathsf{re}}).(\mathsf{g}^{-\mathsf{rex}}) = \mathsf{m}^{\mathsf{e}}$$

Appendix D

Cuckoo Hash with Stash

Cuckoo hashing is a method for resolving collisions in hash tables that employs two hash functions, $\mathsf{h_0}()$ and $\mathsf{h_1}()$. In a cuckoo hash table, each key, x, can be present in one of its two possible locations, $\mathsf{h_0(x)}$ or $\mathsf{h_1(x)}$. As a result, in a cuckoo hash table, a lookup requires checking at most two spots, achieving a worst-case time complexity of $O(1)$. However, inserting a new key into an existing table could be complicated. On insertion, if both designated locations are occupied, the new key "kicks out" the existing key to find a spot. The kicked-out key is relocated to its alternative location, potentially displacing another key.

Hence, while building the hash table, encountering a cycle or exceeding a maximum number of displacements is possible. If this occurs, a new pair of hash functions, $\mathsf{h'_0}()$ and $\mathsf{h'_1}()$, needs to be selected, followed by the rehashing of all elements. This process can be quite expensive, if the total number of keys, N, is large. However, Kirsch et al. [197] demonstrated that the probability of needing to rehash can be reduced to a negligible level, if we allow for a small-size *stash*-area (of size $O(\log \mathsf{N})$).

They proposed that, if a certain threshold of displacements occurs while inserting a new key, the new key should be placed in a stash region. As a result, when searching for a key, if it cannot be found in either of its two designated locations, the stash area must be checked. Given the small size of the stash, the lookup process may take $O(\log \mathsf{N})$ effort in the worst case, and remains effectively $O(1)$ for most of the cases. Their empirical analysis indicates that while generating a hash for 10000 keys, a rehashing probability of $(2 * 10^{-6})$ can be attained by retaining just three items in the stash.

Appendix E

Trusted Execution Environment

Trusted Execution Environment (TEE) is a hardware extension widely available in most modern-day processors. The manufacturer of the TEE hardware guarantees that whatever is written in the code, the TEE-enabled processor will execute as expectedly. Although it sounds obvious, it is not always so. Most of the computation nowadays happens on remote hardware platforms. Therefore, it is possible to manufacture a malicious piece of hardware that will execute CPU instructions differently. This means a trusted executable may behave differently on the remote machine. Therefore, verifying the code's security does not imply that its execution is secure.

Another challenge associated with the remote execution of software is the potential for malicious entities to modify the executable binary after the author has created it. This risk is increasingly relevant today, as most software is deployed remotely in the cloud by untrusted entities. In this context as well, TEE offer a significant advantage. TEEs provide a mechanism that ensures the hardware is executing an unaltered code.

E.1 Enclave

The secure execution environment unit on TEE is known as *enclave*. The concept of the enclave is quite similar to the *process* in the operating systems. Generally, an *enclave author* composes an enclave to perform an independent task. Then that enclave is deployed in the TEE hardware, and then the TEE hardware executes that. A single TEE hardware can execute multiple such enclaves simultaneously. Enclaves are complete execution environments. So, it has code, data, stack, heap, etc., sections.

Deploying an enclave is similar (but not the same) to process loading in the operating system. Before the deployment of an enclave, all its content remains in plain text. A privileged application (like a loader in OS), copies it into some special memory region using a special instruction. During this loading process, the enclave contents become encrypted using a special hardware secret key, which no one except the CPU knows. Not even the manufacturer of TEE hardware, or the maintainer/owner of the TEE platform, can access it. So, computation over the encrypted data is possible by simply sending that data after enclave creation, which is quite a common practice. In fact, computation confidentiality can also be achieved by making part of the executable code available after creating an enclave [198].

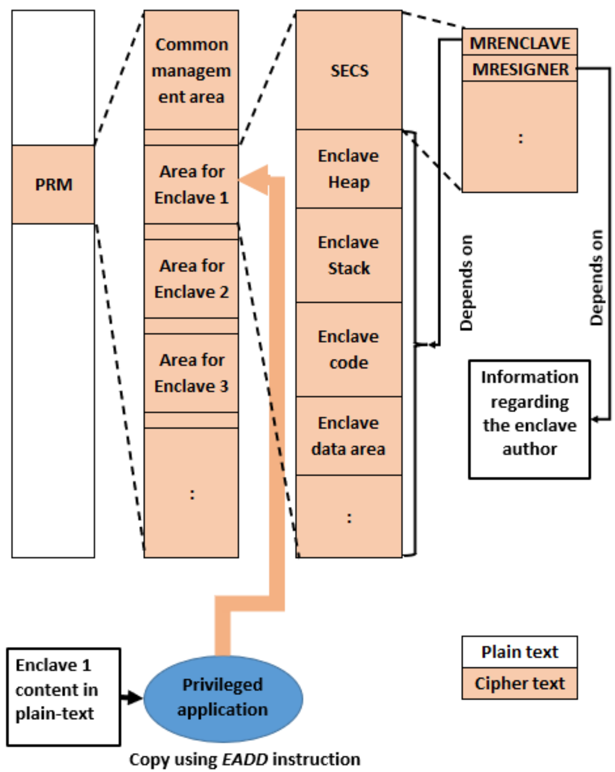


Figure E.1: Memory layout of *enclave*

Software Guard Extensions (SGX) is the TEE manufactured by Intel and available on most Intel processors used in server applications. An SGX enclave has a metadata section known as *SGX Enclave Control Structure (SECS)*, which depends on the content of other sections, different attributes of the enclave, information about the enclave author, etc. An

enclave's identity is almost synonymous with the content of its *SECS* [126]. The memory layout of an SGX enclave is shown in figure: E.1.

The enclave author composes an enclave according to the wished functionality. After composition, enclave contents remain in plaintext and can be shared as dynamically loaded libraries, packaged as *.so* or *.dll* files. Only a privileged application on SGX-enabled platform can build and deploy the enclave by executing *EADD* instruction. enclaves are deployed in a special memory region known as *processor-reserved memory (PRM)*, which is a sub-part of the RAM [99].

Confidentiality of an Enclave

The memory management unit (MMU) does not allow anyone(including privileged users) to access *PRM* region. Only SGX-enabled CPU can access it internally. Any user's read access returns all-0's, and write access does not make any changes to it. Even physically probing RAM or monitoring bus traffic does not expose the contents of the enclave. All data is stored in the PRM in an encrypted format using a CPU-specific secret key, remaining encrypted as it travels through the bus. It is only decrypted within the CPU registers just prior to execution.

Similar to the public methods exposed by an object in object-oriented programming, the enclave exposes a few entry points, known as *ECALL*s. Normal world programs can call those *ECALL*s and access the functionalities implemented within the enclave. Enclave authors determine how many and what functionalities will be exposed through *ECALL*s and implements those inside the enclave. On the other hand, an enclave can use the functionalities implemented in the outer world through *OCALL*s.

Integrity of an Enclave

After composition of the enclave (i.e., initializing its code, stack, heap, and other regions), it may be deployed in a remote TEE-platform with the help of some non-secure parties. Those non-secure parties may try to alter the enclave content before deployment. However, TEE-platform provides mechanisms to ensure the integrity of the deployed enclave.

In Intel-SGX, an enclave is built and deployed by copying its content (i.e., code, stack, heap etc.) part by part into the *PRM* region, using *EADD* instruction. This instruction takes the plaintext input and writes encrypted output to *PRM* region. This encryption uses CPU-specific secret key. Whenever an enclave's contents are changed in *PRM* region,

the $MRENCLAVE$ structure in the $SECS$ region changes according to the enclave's hash.

After deployment, the enclave must be started by $EINIT$ instruction. $SIGSTRUCT$ is an important structure in this stage, which is required as an input to the $EINIT$ instruction during enclave initialization. It is basically a certificate signed by the enclave author. It certifies the final expected value of the $MRENCLAVE$. Before starting an enclave, SGX hardware validates whether the signature in $SIGSTRUCT$ matches with $MRENCLAVE$ of the enclave. If a mismatch occurs, then SGX does not start the enclave. Otherwise, SGX computes a hash over $SIGSTRUCT$ and keeps that in a section called $MRSIGNER$ within the $SECS$. Since $MRSIGNER$ is generated by the trusted hardware, it guarantees that the enclave is not altered after its creation and executing the same code the enclave owner wants.

E.2 Attestation

To ensure the integrity of the enclave's execution, SGX provides attestation mechanisms that guarantee the unaltered enclave is executing on authentic SGX-enabled hardware. The attestation mechanisms use a structure called *REPORT*. This *REPORT* structure can only be created by an enclave, running on a genuine SGX-platform. Among other things, this structure contains the *MRENCLAVE* and *MRSIGNER* values of the enclave. There are two kinds of attestation mechanisms, *a) Local-attestation* and b)*Remote-attestation*.

During the local attestation mechanism, two different enclaves running on the same hardware platform can prove to each other that they are indeed running on the same SGX hardware platform. So, if the verifier enclave trusts its own hardware platform, it can also trust the prover enclave's environment. The central idea is that a *message authentication code (MAC)* is created on the fields of *REPORT* structure, which can only be generated and verified by entities who has access to the same the hardware's secret. This hardware secret is not accessible by any user, only SGX-platform can internally access the hardware-secret and produce the MAC. So, if the verifier enclave can verify the *REPORT*, it gets assurance that the prover enclave's stated $MRENCLAVE$ value is un-altered and that enclave is running on the same SGX-enabled platform.

A more common scenario involves an enclave author who composes an enclave intended for deployment on a remote SGX platform. The mechanism of ensuring that the exact same enclave is executed in an un-altered state on genuine SGX hardware is referred to as remote attestation. This mechanism utilizes the previously described local attestation

process, which occurs between the deployed enclave and a specialized trusted enclave on the same SGX platform, known as the quoting enclave ($Eclv_Q$). The quoting enclave possesses a private-public key pair (Pri_Q-Pub_Q). For each genuine SGX platform produced, the manufacturer certifies the corresponding public key (Pub_Q).

After confirming the execution environment of the deployed enclave through local attestation, $Eclv_Q$ signs the $REPORT$ of that enclave using Pri_Q. The signed $REPORT$ includes, among other details, the $MRENCLAVE$ value of the deployed enclave as well as some *user-data*. Anyone can verify the signed $REPORT$ by using the corresponding public key from the certificate issued by the manufacturer. The $user-data$ section within the $REPORT$ may be used by the user to transfer some application-specific data with the guarantee against *man-in-the-middle (MITM)* attacks.

Therefore, if one can validate the signature and the *MRENCLAVE* in the signed *REPORT* aligns with expectations, two confirmations are achieved. First, it verifies that the deployed enclave is running on a genuine SGX hardware platform (since only an authentic SGX platform can produce a verifiable signature). Second, it ensures the integrity of the enclave is upheld (as the *MRENCLAVE* value in the report matches expectations). In this context, it is important to note that SGX provides anonymous attestation, which allows a user to verify that the attestation is created by a genuine TEE-hardware without revealing its actual identity.

E.2.1 Attested-TLS

Attested-TLS [124] is the combination of *TLS*-protocol [199] and *SGX-remote attestation*. Like normal-TLS, *Attested-TLS* creates a secure end-to-end encrypted and integrity-protected channel. Additionally, during the *attested-TLS* channel establishment phase, the client can get a confirmation that, the server side of the channel is actually an enclave having the expected $MRENCLAVE$ value. It also ensures that the enclave is actually executing on a genuine SGX-platform. Which means, the channel-establishment phase provides the same assurance of a remote-attestation.

For providing the above mentioned features in *Attested-TLS*, an enclave acts as a server and generates a fresh private-public key pair (Pri_{Enc}-Pub_{Enc}). The enclave also obtains its own remote attestation $REPORT$, signed by the $Eclv_Q$. Along with enclave's *MRENCLAVE, MRSIGNER*, etc, this $REPORT$ also contains the hash of the newly generated Pub_{Enc} in its $user-data$ section. Which means, this remote attestation $REPORT$ also proves that the Pub_{Enc} is generated within the enclave running on a genuine SGX-hardware

platform. Finally, the enclave generates a self-signed certificate $Cert_{Enc}$ with Pub_{Enc} and attaches this remote attestation $REPORT$ in its extension field.

When a TLS client intends to connect with the enclave, the enclave sends $Cert_{Enc}$, to the client. Upon receiving this certificate, the client extracts the remote-attestation $REPORT$ from it and verifies the signature of $Eclv_Q$. This step confirms that the $REPORT$ has been generated by genuine SGX hardware. Next, the client checks whether the $MRENCLAVE$ and $MRSIGNER$ values stated in the $REPORT$ are as expected. Finally, the client verifies that the hash of the public key included in the received certificate matches the content of the $user - data$ section. If all checks are successful, the client establishes a TLS channel with the server and subsequently transfers data through this secured channel.

E.3 Sealing

Enclaves are temporary entities that reside in RAM. However, if necessary, they can store data in permanent storage, albeit in an encrypted format. Later, a new enclave- equipped with the necessary permissions and configurations - can generate the required key to retrieve the information from the encrypted state. This process is called sealing. Sealing employs symmetric cryptography. The essence of sealing lies in the fact that the key needed for both sealing and unsealing is not stored anywhere; instead, it is generated as needed. There are two distinct policy types available that govern the key generation process.

- ***Sealing to the Enclave Identity policy:*** In this policy, the key generation depends on the hardware secret and the *MRENCLAVE* value. So, none other than an exact copy of the enclave, on the same hardware platform can generate the key which can unseal the data.
- ***Sealing to the Sealing Identity policy:*** With this policy, the key generation depends on the identity of the enclave's author and version. This policy allows data sealed by one enclave to be unsealed by another enclave created by the same enclave-author. The unsealing enclave does not need to be the exact replica of the sealing enclave.

E.4 Formal modeling

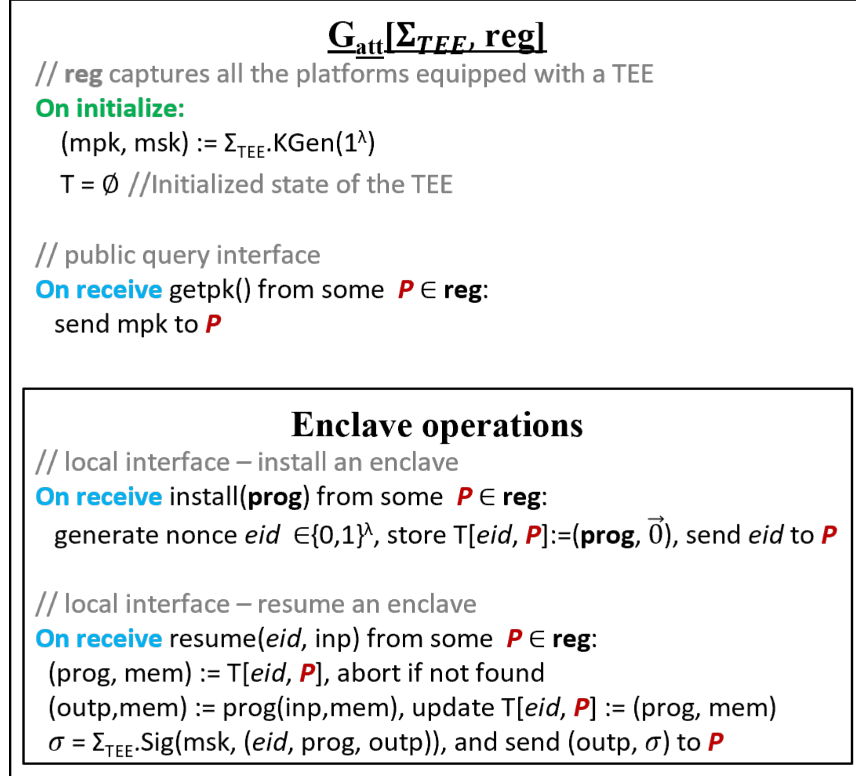


Figure E.2: Ideal functionality for G_{att} [1]

A formal model of a TEE is required to analyze the security of TEE-related protocols. Pass et al. formally defined G_{att} in [1], which represents the ideal functionality for trusted execution environments (Figure E.2). A party (P), with a TEE access first creates a new enclave to run the program code (*prog*) by an **install**() call. On success, G_{att} returns a unique ID, known as enclave ID (*eid*), to the caller. The program code defines several entry points to the enclave.

After obtaining the *eid*, P can invoke those entry points by a **resume**() call with corresponding inputs. Accordingly, the enclave performs a specific computation according to the *prog*. When the computation is complete, the output *outp* and its *attestation* are returned to the caller. The attestation of the *outp* is denoted by $\sigma_{TEE}(outp)$, which is a signature produced by a secret key sk_{TEE}, stored in TEE hardware. $\sigma_{TEE}(outp)$ is generated after com-

bining the eid, $prog$ and $outp$ together: $\sigma_{TEE}(outp) = \Sigma_{TEE}.Sig(sk_{TEE}, (eid, prog, outp))$.

To verify this $\sigma_{TEE}(outp)$, the corresponding public key pk_{TEE} can be obtained by invoking $G_{att}.getpk()$. Successful verification of $\sigma_{TEE}(outp)$ confirms that the specific instance of the enclave, executing the exact same expected code on a legitimate TEE-platform, produced the computation result.

Appendix F

Universal Composability

Goldwasser and Micali introduced the concept of Semantic Security [200]. A semantically secure encryption scheme demonstrates that having access to the ciphertext does not provide the adversary, $\mathcal{A}$, with any additional information. In this context, $\mathcal{A}$ can be considered a black-box algorithm that processes the ciphertext as input and produces some information regarding the plaintext as output. Semantic security guarantees that, irrespective of the internal workings of this black box, $\mathcal{A}$ can produce the same output without relying on the ciphertext.

F.1 Background concept: Simulation-based Security

Simulation-based security [77] builds upon the concept of semantic security to establish the security of *protocol executions*. In this context, rather than observing the ciphertext, an adversary examines the trace of the protocol's execution, or the *view* of the protocol's execution. Much like semantic security, the aim here is to demonstrate that anything the adversary can learn from this view can also be determined without any access to it.

Simulation-based security introduces some important concepts: *Real world*, *Ideal world*, and *Simulator*. The real world represents what actually occurs during the execution of the protocol. $\mathcal{A}$ exists in this real world and outputs *something* by utilizing the protocol-generated trace as its input.

In contrast, the ideal world is secure, by definition. The simulator, Sim, acts as the adversary residing in the ideal world. Sim actually runs $\mathcal{A}$ as a sub-protocol, without knowing its internal details. Since this occurs entirely within the ideal world, Sim cannot

gain any knowledge, as the ideal world is secure by definition. Consequently, Sim supplies *simulated* inputs (protocol traces) to $\mathcal{A}$ while executing it as a sub-protocol.

The fundamental proof strategy involves crafting the simulation process of Sim such that the simulated inputs are indistinguishable to $\mathcal{A}$ from those generated during the real execution of the protocol. Since $\mathcal{A}$ cannot differentiate between these two sets of inputs, it follows that it cannot produce differing outputs.

It is worth noting that, depending on the desired adversarial model, $\mathcal{A}$ can also control the behavior of the involved parties and instruct them to perform anything, including deviating from the protocol steps in any arbitrary way. This manipulation enables $\mathcal{A}$ to generate a variety of protocol traces, which may significantly help $\mathcal{A}$ to break the security.

F.2 Additional concepts in UC-framework: Environment and Ideal Functionality

Ran Canetti further extends this concept by introducing the Universal-Composability (UC) framework [129]. The primary objective of the UC framework is to ensure the security of a given protocol even when it operates in parallel with other potentially insecure protocols. To achieve this, the UC framework introduces an environment machine, denoted as $\mathcal{Z}$, which supplies inputs to both the participating parties in the protocol and the adversary. Once the execution of the protocol is complete, $\mathcal{Z}$ collects the outputs from all parties, including the adversary. A protocol that is UC-secure guarantees that the set of outputs is indistinguishable between the real world and the ideal world.

The UC-framework also introduces the concept of ideal functionality, $\mathcal{F}$. This trusted entity exists exclusively in the ideal world and fulfills the same objectives for which the actual protocol is designed. The specific design of $\mathcal{F}$ is determined by the objectives of the actual protocol. In the ideal world, all parties submit their inputs to $\mathcal{F}$, which performs the necessary operations on their behalf and delivers accurate outputs to them. Additionally, $\mathcal{F}$ may intentionally leak certain information after receiving inputs from the parties, with the nature of the leakage depending on the protocol's security and privacy goals.

F.3 Real-world protocol execution

Figure F.1 illustrates a top-level overview of a real-world protocol execution within the UC framework. Prior to initiating the protocol execution, $\mathcal{Z}$ identifies the corrupt party

among the participants involved. $\mathcal{A}$ has control over the behavior and inputs of these corrupt parties. Furthermore, $\mathcal{A}$ is able to observe the outputs and internal states of the corrupted parties. The honest parties, who faithfully adhere to the protocol steps and over whom $\mathcal{A}$ has no control, receive their inputs directly from $\mathcal{Z}$ and subsequently relay their outputs to the environment. Additionally, $\mathcal{Z}$ may may receive information from $\mathcal{A}$.

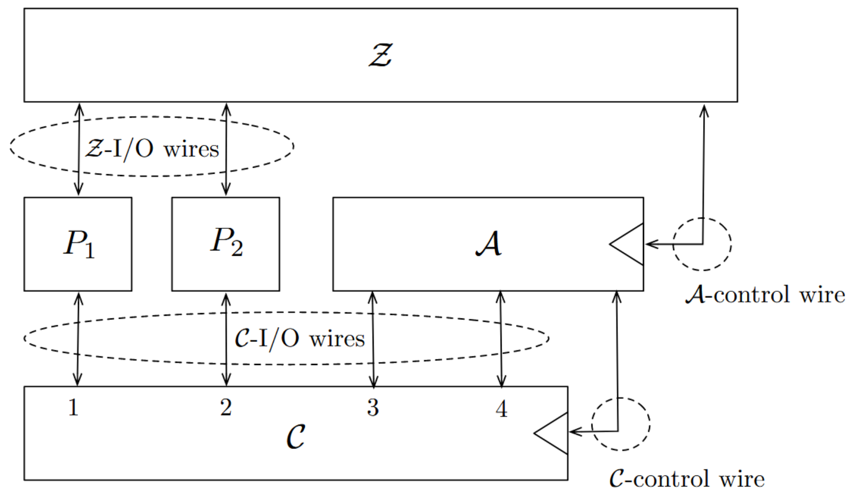


Figure F.1: Real-world protocol execution in UC-Framework [2, Chapter 23.5.1]

When the parties communicate with one another (irrespective of honest or corrupt), they do so through a common communication network, $\mathcal{C}$, which $\mathcal{A}$ can influence. Specifically, $\mathcal{A}$ can block or delay the delivery of messages directed to the honest parties. Depending on the communication model, $\mathcal{A}$ may also gain access to certain leakage information from $\mathcal{C}$ (for example, details about who is sending messages to whom, the size of the messages, etc.).

F.4 Ideal-world protocol execution

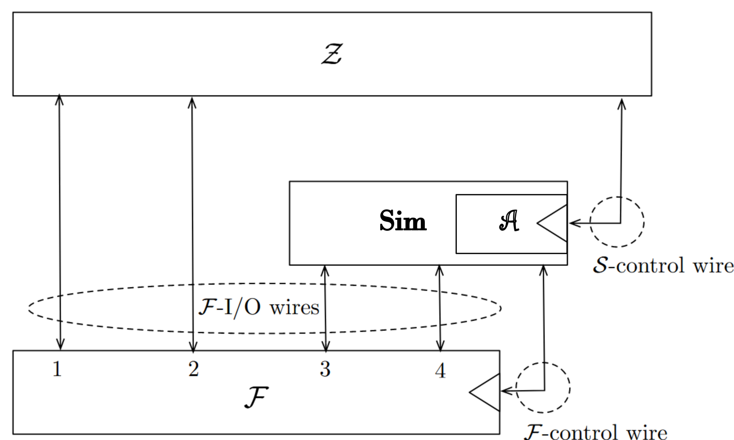


Figure F.2: Ideal-world protocol execution in UC-Framework [2, Chapter 23.5.2]

In an ideal world (Figure F.2), the honest parties do not engage in any protocol steps, including communication among themselves. Upon receiving inputs from $\mathcal{Z}$, these honest parties relay the inputs directly to the ideal functionality, $\mathcal{F}$, which performs all necessary operations on their behalf. The corrupted parties, meanwhile, remain under the influence of $\mathcal{A}$ and execute the protocol steps as dictated by $\mathcal{A}$. However, the simulator, Sim, runs $\mathcal{A}$ as a sub-protocol and can observe the inputs and outputs of the corrupted parties.

Sim has the ability to provide a protocol input to $\mathcal{F}$ that corresponds to a corrupt party, and it may also send control messages to $\mathcal{F}$. Similar to $\mathcal{A}$ in the real world, Sim conveys certain information to $\mathcal{Z}$. It is important to note that both the real and ideal worlds present the same interface to $\mathcal{Z}$.

A typical UC-proof demonstrates that $\mathcal{Z}$ cannot distinguish between the real world and the ideal world. It is important to note that honest parties do not send any protocol message to the corrupted parties. Therefore, the proof strategy focuses on designing Sim in such a way that, whenever a corrupt party in the real world would normally receive a message from an honest party, Sim must step in to generate a simulated message that is indistinguishable from the real-world protocol message.

Since the corrupt parties (ultimately $\mathcal{A}$) cannot tell them apart, $\mathcal{A}$ generates an output that mirrors that of the real world. Furthermore, because Sim runs $\mathcal{A}$ as a sub-protocol, it can forward the output produced by $\mathcal{A}$ to $\mathcal{Z}$. Consequently, $\mathcal{Z}$ is unable to differentiate between the real and ideal worlds.

Appendix G

UC-proof of ROT

We adhere to the general strategies of UC-security proofs found in existing TEE-based solutions [1, 128, 201] while designing the simulator for ROT. The following sections detail the design of the simulator. We also present the argument for indistinguishability, covering all possible combinations of corrupt and honest parties.

G.1 Sim Design - UsageApproval Stage:

There are four possible combinations of honest and corrupt parties during this stage.

A. When both DO and DU are honest: Since both parties are honest, they do not engage in any protocol steps; instead, they directly provide their inputs to $\mathcal{F}_{ROT}$, which handles all the operations. As a result, the $\mathcal{Z}$ may notice the lack of network messages. Consequently, Sim's primary objective is to simulate these network messages. Additionally, even if both parties are honest, $\mathcal{Z}$ can still instruct $\mathcal{A}$ to block or tamper with the network messages and observe whether the outcomes align with those of the real-world execution. Therefore, Sim must also simulate the effects of any tampered network messages as follows:

1. In the ideal world, any communication between the ideal functionality and the adversary passes through the simulator. Hence, Sim gets the notification ("UsageApproval"...) from $\mathcal{F}_{ROT}$ and starts after that. At this moment, $\mathcal{F}_{ROT}$ pauses its execution and waits for any further instruction from Sim.
2. Sim then simulates all network messages of the UsageApproval stage on behalf of both honest parties by utilizing the information received from the notification provided by

$\mathcal{F}_{ROT}$. It then delivers these network messages to the adversary one after another, mimicking the real-world protocol execution.

For instance, Sim first prepares a "GetDetails" message, addressed to DU, and sends that to the adversary, $\mathcal{A}$. The necessary information to create this message is already present in ("UsageApproval"...) notification obtained from $\mathcal{F}_{ROT}$. Since, within the UC framework, Sim has the capability to manipulate the sender's identity [129, Chapter 2.3] of the network message, the corrupt DU cannot detect that the message was actually sent by Sim.

3. After delivering each network message to the adversary, Sim observes the actions of $\mathcal{A}$ upon receiving the message. If $\mathcal{A}$ forwards the message to the intended recipient without alteration, then Sim generates a reply message on behalf of the receiving honest party and transmits that simulated message to the adversary, then moves on to the next step of the protocol.

4. If $\mathcal{A}$ permits all network communication unaltered, then Sim instructs $\mathcal{F}_{ROT}$ to proceed normally.

5. On the other hand, if Sim detects any tampering of the network communication, it simulates the corresponding effect of the real-world protocol execution and instructs $\mathcal{F}_{ROT}$ to abort.

6. One exception is that, if Sim detects tampering on the signature part of the signed enclave, during the transmission of a signed enclave, it notes this fact but instructs the $\mathcal{F}_{ROT}$ to continue operating normally. However, if Sim notices tampering in the content part, it instructs $\mathcal{F}_{ROT}$ to abort.

The main goal of the simulator is to ensure that the effects of network message manipulation are indistinguishable in both the real and ideal worlds; thus, $\mathcal{Z}$ cannot differentiate between them.

B. When DO is honest but DU is corrupted: In this case, Sim primarily detects any improper replies from DU and implements corresponding countermeasures to ensure that the execution in the ideal world matches the execution in the real world. Hence:

1. Since DO is honest, it will call $\mathcal{F}_{ROT}$. As a result, $\mathcal{F}_{ROT}$ will issue a notification to $\mathcal{A}$, which will first reach Sim.

2. Sim starts after getting this notification. On behalf of DO, it calls GetDetails entry point of DU with the SR received from $\mathcal{F}_{ROT}$.

3. Next, Sim observes what does the corrupted DU return to DO. If the returned values do not match with $Req(D)$, ID_{Enc} and $Cert_{DU}$ obtained from $\mathcal{F}_{ROT}$, Sim instructs $\mathcal{F}_{ROT}$ to abort.

4. Otherwise, Sim simulates the real-world DO's subsequent communication with $\mathsf{Lib}_{\mathsf{Enc}}$.

5. If nothing got blocked or tampered, Sim generates a random key-pair $(Pri_{sim} - Pub_{sim}) \leftarrow \mathcal{AE}.KeyGen(1^{\lambda})$

6. Sim then signs $Eclv_{\mathcal{C}}$ and produces $Eclv_{\mathcal{C}}^{sim}$

7. Subsequently, Sim sends $Eclv_{\mathcal{C}}^{sim}$ to DU and instructs $\mathcal{F}_{ROT}$, to continue normally.

Hybrid 1: It works similarly to the real-world scenario, except that the signed enclave, $Eclv_{\mathcal{C}}^{DO}$ is replaced with $Eclv_{\mathcal{C}}^{sim}$. Since the signature scheme, Σ, is EUF-CMA secure and the signature verification key is not revealed to anyone, *Hybrid 1* remains indistinguishable from the real-world.

This *Hybrid 1* is exactly the same as the ideal world.

C. When DO is corrupt but DU is honest: In this context, Sim identifies any inappropriate responses from DO and enacts relevant counteractions to guarantee that the execution in the ideal world aligns with that in the real world. Hence,

1. Sim activates after observing a GetDetails call from DO. Sim can observe this call, since this is made to an external party by $\mathcal{A}$.

2. Sim then invokes the UsageApproval entry point of $\mathcal{F}_{ROT}$ with SR received from the corrupt DO.

3. Upon receiving back the notification from $\mathcal{F}_{ROT}$, Sim simulates the communication from the real-world DU to $\mathsf{Lib}_{\mathsf{Enc}}$ and returns the $Req(D)$, ID_{Enc} and $Cert_{DU}$ to DO, which Sim has already received from $\mathcal{F}_{ROT}$.

4. Then Sim waits for DO to return the signed enclave, $Eclv_{\mathcal{C}}^{DO}$.

5. If the enclave content (excluding the signature part) returned by DO exactly matches what Sim received from $\mathcal{F}_{ROT}$, then it instructs $\mathcal{F}_{ROT}$ to continue normally; otherwise, it sends an abort message.

D. When both DO and DU are corrupt: As both the parties are corrupt, the ideal functionality will not be used at all, requiring no simulation.

G.2 Sim Design - DataProcurement Stage:

A. When both DO and DU are honest: Since both parties are honest, $\mathcal{Z}$ can only observe the network messages. With both parties acting honestly, all essential operations are managed by $\mathcal{F}_{ROT}$, requiring no involvement from G_{att}. However, Sim is unable to produce a valid attestation of the public key on its own, leaving it vulnerable to detection by $\mathcal{Z}$. To mitigate this, Sim has to interact with G_{att} to obtain a public key epk^{sim} along with its attestation $\sigma_{TEE}(epk^{sim})$. However, $\mathcal{Z}$ cannot observe this interaction, as it is a local interaction [1]. Physically, this means Sim accesses its own TEE platform.

Specifically, Sim performs the followings:

1. When Sim receives a notification regarding (DataProcurement, ...) from $\mathcal{F}_{ROT}$, it checks whether it has recorded any tampering in the signature portion of the enclave during the previous stage. This corresponds to step 6 of Case A in the simulator design for the UsageApproval stage.
2. If there was a tampering, Sim sends an abort message to $\mathcal{F}_{ROT}$.
3. Otherwise, Sim interacts with G_{att} to install and subsequently invoke the "EncGetData" entry point. As a result, it obtains $eid^{sim}, (epk^{sim}, \sigma_{TEE}(epk^{sim}))$ and $(rand_S^{sim}, \sigma_{TEE}(rand_S^{sim}))$ from G_{att}. Sim utilizes $eid^{sim}, (epk^{sim}, \sigma_{TEE}(epk^{sim}))$ and $rand_S^{sim}$ in later stages of the simulation process.
4. Sim then creates network message $(eid^{sim}, (epk^{sim}, \sigma_{TEE}(epk^{sim})))$ destined for DO and delivers to $\mathcal{A}$ and then observes $\mathcal{A}$'s behavior.
5. If adversary blocks or tampers this communication, Sim instructs $\mathcal{F}_{ROT}$ to abort.
6. Otherwise, Sim generates a random message of the size $|(D|Cert_{DU}|\mathbb{T}_{\mathsf{L}})|$ (received from $\mathcal{F}_{ROT}$) and encrypts that with epk^{sim}.
7. Sim then hands over the ciphertext to $\mathcal{A}$, who is expected to deliver it to DU.
8. If $\mathcal{A}$ does not deliver that or tampers with that ciphertext before delivering, Sim instructs $\mathcal{F}_{ROT}$ to abort.
9. Otherwise, Sim writes $rand_S^{sim}$ to the blockchain followed by a read operation, similar to the real-world.
10. After reading, if Sim finds $rand_S^{sim}$ is present within the blockchain then instructs $\mathcal{F}_{ROT}$ to continue normally.

11. Otherwise, instructs $\mathcal{F}_{ROT}$ to abort.

Hybrid 1: It works similarly to a real-world scenario, with the exception that (eid, epk) is replaced by (eid^{sim}, epk^{sim}). Since an enclave generates both the enclave ID and key pair randomly, (eid, epk) will remain indistinguishable from (eid^{sim}, epk^{sim}). Consequently, *Hybrid 1* remains indistinguishable from the real-world scenario.

Hybrid 2: It works similarly to *Hybrid 1*, with a difference that $\sigma_{TEE}(epk)$ is replaced by $\sigma_{TEE}(epk^{sim})$. Since, G_{att} employs anonymous attestation [1]. By examining either $\sigma_{TEE}(epk^{sim})$ or $\sigma_{TEE}(epk)$, one can ascertain that *one of the* trusted TEE has issued this attestation, without identifying the exact entity responsible. Thus, *Hybrid 2* remains indistinguishable from *Hybrid 1*.

Hybrid 3: Similar to the *Hybrid 2*, except that ct_{DO} is replaced with encryption of a dummy message of the same length. Since $\mathcal{AE}$ is IND-CPA secure, *Hybrid 3* is indistinguishable from *Hybrid 2*.

Hybrid 4: Similar to the *Hybrid 3*, except the value written to the blockchain, $rand_S$, is replaced with $rand_S^{sim}$. Since both $rand_S$ and $rand_S^{sim}$ are generated randomly, they remain indistinguishable. Notice, this *Hybrid 4* is exactly same as the ideal-world.

B. When DO is honest but DU is corrupt: Since DO is honest, all its functions will be handled by $\mathcal{F}_{ROT}$. To ensure that the scenario remains indistinguishable from real-world operations, Sim must replicate the corresponding protocol exchanges as would occur with a real-world DO. Specifically, Sim is required to supply the encrypted personal data to DU's enclave. However, according to the established privacy definition, $\mathcal{F}_{ROT}$ does not disclose personal data D, which means that Sim remains unaware of it. Consequently, Sim inputs simulated personal data, D_{Sim}, into the DU's enclave.

Following are the detailed steps performed by Sim:

1. After getting a (DataProcurement, ...) notification from $\mathcal{F}_{ROT}$, Sim invokes the SaveData entry-point of DU.

2. Although the communication between DU and G_{att} is local, DU is corrupt. As a result, Sim can monitor and intercept that communication leveraging the **extraction** method [1], and neither $\mathcal{A}$ nor $\mathcal{Z}$ can detect this interception.

3. Sim sends an abort message to $\mathcal{F}_{ROT}$ if it does not observe the following before DU replies to DO:

    - An install call from DU to G_{att} with $Eclv_{\mathcal{C}}^{DO}$.

- A resume call from DU to the EncGetData entry-point of G_{att}.

4. Whenever DU makes an (eid, "resume", ("EncGetData", DO)) call to G_{att}, Sim intercepts that call and replaces the input parameter DO with its own identity, Sim.

5. Subsequently, when G_{att} replies back, Sim notes down the response before forwarding that to the corrupted DU.

6. Sim again waits for a reply from DU to DO.

7. If DU does not return the exact same $(eid, epk, \sigma_{TEE}(epk))$ that DO noted down in Step 5, Sim sends an abort message to $\mathcal{F}_{ROT}$.

8. Otherwise, Sim generates a simulated message that includes a simulated data D_{Sim}, $Cert_{DU}$, and a random time limit, ensuring that the combined message corresponds to $|(|D|Cert_{DU}|\mathbb{T}_{\mathsf{L}})|$.

9. Sim then encrypts the simulated message using epk and delivers the ciphertext, ct_{Sim}, to DU.

Hybrid 1: It operates similarly to a real-world scenario, with the distinction that, ct_{DO} is changed to ct_{Sim}. Given that $\mathcal{AE}$ is IND-CPA secure and $|ct_{DO}| = |ct_{\mathsf{sim}}|$, *Hybrid 1* remains indistinguishable from the real world.

This *Hybrid 1* is precisely the view of $\mathcal{Z}$ in the ideal world execution.

C. When DO is corrupt but DU is honest: Since DU is honest, it will not perform any actions, including not launching an enclave. Therefore, in this case, to deceive $\mathcal{Z}$, Sim communicates with G_{att} and the blockchain to ensure that the protocol exchange mirrors the real-world scenario. Specifically:

1. Sim activates upon observing a "SaveData" call from the corrupted DO. Following this call, Sim communicates with G_{att}, similar to a real-world DU.

2. Specifically, Sim issues an "install" call to G_{att} with the $Eclv_{\mathcal{C}}^{DO}$, which it received during the UsageApproval stage, and then calls (eid^{sim}, "resume", ("EncGetData", DO)).

3. When G_{att} returns, Sim sends $eid^{sim}, epk^{sim}, \sigma_{TEE}(epk^{sim})$ to DO and waits for ct_{DO}.

4. After receiving ct_{DO}, Sim faithfully writes and reads the blockchain, similar to the real world.

5. Subsequently, it invokes G_{att} with (eid^{sim}, "resume", ("EncSaveData", $ct_{DO}, block$)).

6. If G_{att} returns normally, it indicates that DO has not provided any unauthorized data. In this case, Sim invokes the UsageApproval entry point of $\mathcal{F}_{ROT}$ and instructs to proceed normally.

7. Otherwise, Sim does not call $\mathcal{F}_{ROT}$ at all.

8. It is important to note that since the actual inputs from the data owner are not known to Sim, it invokes the UsageApproval entry point of $\mathcal{F}_{ROT}$ with dummy parameters. As a result, $\mathcal{F}_{ROT}$ does not store the real data; instead, it is stored in Sim's enclave.

In this simulation, the network messages being simulated ensure that the perspective of the external adversary is consistent in both the real and ideal worlds. The absence of stored actual data in $\mathcal{F}_{ROT}$ is addressed in the simulation of the subsequent stage.

D. When both DO and DU are corrupt: As both the parties are corrupt, the ideal functionality will not be used by any of them, requiring no simulation.

G.3 Sim Design - DataUsage Stage:

Since only DU is involved in this stage, there should be two combinations. However, the output produced in this phase depends on the data received from DO in the previous stage, resulting in a total of four combinations.

A. When DU is honest and DO was honest in the previous stages: In this case, DU produces the correct output by consulting $\mathcal{F}_{ROT}$. Sim accurately records and then retrieves from the blockchain to ensure the observable effect of this stage resembles real-world execution.

B. When DU is honest and DO was corrupt in the previous stages: If DO was corrupt but the protocol has reached this point, it means that $\mathcal{F}_{ROT}$ has not received any abort messages from Sim. This indicates that, despite being corrupt, DO followed all the protocol steps correctly. Therefore, Sim must simulate communication with the blockchain. Additionally, Sim must address another aspect: since DO was corrupt, as indicated in the previous simulation steps, $\mathcal{F}_{ROT}$ does not retain the actual data. Instead, the data is stored in Sim's enclave.

Therefore,

1. Sim starts after getting a ("DataUsage") notification from $\mathcal{F}_{ROT}$ and immediately sends abort message to $\mathcal{F}_{ROT}$.

2. Then, it first invokes G_{att} with (eid^{sim}, "resume", ("EncDataUsage", $\perp, \perp$)) to receive $rand_E$ from G_{att}.

3. Then just like a real-world DU, Sim faithfully writes and then reads *block* from the blockchain.

These simulation steps are sufficient to ensure that $\mathcal{Z}$ perceives the same outcomes in both real-world and ideal-world scenarios. However, for the sake of completeness, we describe a few additional simulation steps that will produce the correct output for the honest DU.

1. Then again invokes G_{att} with (eid^{sim}, "resume", ("EncDataUsage", $block, \perp$)).

2. As a result G_{att} returns $\mathcal{AE}.Enc(Pub_{DU}, \mathcal{C}(\mathcal{D}))$.

3. Sim then invokes DataUsage entry point of $\mathcal{F}_{ROT}$ with $y = \mathcal{AE}.Enc(Pub_{DU}, \mathcal{C}(\mathcal{D}))$.

4. As a result, honest DU receives the correct output from $\mathcal{F}_{ROT}$.

C. When DU is corrupt and DO was honest in the previous stages: In this scenario, DU's enclave had stored simulated data instead of the actual data from DO. Consequently, when performing computations, DU's enclave will not produce the correct output. To address this issue, Sim interacts with $\mathcal{F}_{ROT}$ and also utilizes the equivocation method [1] to align the situation with that of the real world.

Specifically:

1. Sim activates, when it observes a (eid, "resume",("EncDataUsage", $block, \perp$)) call to G_{att} from DU.

2. Sim intercepts that call and invokes DataUsage entry point of $\mathcal{F}_{ROT}$.

3. $\mathcal{F}_{ROT}$ then returns the correct computation output, $\mathcal{AE}.Enc(Pub_{DU}, \mathcal{C}(\mathcal{D}))$, to Sim.

4. Sim then employs the equivocation method to produce a valid attestation for the output returned by the ideal functionality. Specifically, Sim modifies the third parameter of the previously intercepted call to G_{att}. The original call (eid, "resume",("EncDataUsage", $block, \perp$)) is transformed into (eid, "resume",("EncDataUsage", $block, \mathcal{AE}.Enc(Pub_{DU}, \mathcal{C}(\mathcal{D}))$)). Consequently, the enclave produces the attestation of the result generated by the ideal functionality.

5. Then, Sim provides the output generated by $\mathcal{F}_{BPPM}$ together with the attestation obtained through the equivocation method to DU in response to the original intercepted call to G_{att}.

6. Since DU observes the correct output and its valid attestation, it cannot detect any difference from the real-world behavior.

D. When DU is corrupt and DO was corrupt in the previous stages: Nothing is performed by the ideal functionality, hence nothing to simulate.

Thus, we can conclude that $Prot_{ROT}$ UC-realizes $\mathcal{F}_{ROT}$. □

Appendix H

UC-proof of *BPPM*

In a manner similar to the proof for ROT, we aim to demonstrate the indistinguishability between the ideal world and the real world in this proof. The subsequent sections will outline the design of the simulator for all stages of *BPPM*. Additionally, we will present the argument for indistinguishability, addressing all possible combinations of corrupt and honest parties.

H.1 Sim Design - Setup Stage:

Our threat model assumes that CP is honest, while $DU_{i,j}$ can either be honest or corrupt. Thus, there are two possible combinations.

A. When $DU_{i,j}$ is honest: Since $DU_{i,j}$ is honest, all necessary operations are handled by $\mathcal{F}_{BPPM}$, requiring no actions from $DU_{i,j}$ or its enclave. However, an external adversary may detect the absence of network traffic. To address this, Sim simulates the network traffic. Furthermore, Sim interacts with G_{att} for the sake of generating attested messages.

Specifically Sim performs the following:

1. When Sim gets a notification (Setup, $DU_{i,j}$, $PDS_{i,j}[]$) from $\mathcal{F}_{BPPM}$, it interacts with G_{att} to install an enclave within its own TEE-platform.
2. Then Sim invokes the "EncSetKey" entry-point, and receives eid^{sim}, epk^{sim} and $\sigma_{TEE}(epk^{sim})$ from the enclave.
3. Next, Sim emulates the real-world interaction between $DU_{i,j}$, and CP.

4. To do this, Sim parses the received $PDS_{i,j}[]$ from $\mathcal{F}_{BPPM}$, and generates $|PDS_{i,j}[]|$-number of network messages of the format (SendCode, $PD.S$, eid^{sim}, epk^{sim}, $\sigma_{TEE}(epk^{sim})$).

5. Only after getting $|PDS_{i,j}[]|$-number of untampered replies from CP, Sim instructs $\mathcal{F}_{BPPM}$ to continue and also publishes eid^{sim}, epk^{sim} and $\sigma_{TEE}(epk^{sim})$.

   Note: Since Sim does not possess the execution permission for the requested code, CP will reply with an encrypted dummy code in all the responses. This will be handled in the later stage of simulation.

6. On the other hand, if Sim notices network message tampering, then it instructs $\mathcal{F}_{BPPM}$ to abort.

Hybrid 1: It works similarly to a real-world scenario, with the exception that (eid, epk) is replaced by (eid^{sim}, epk^{sim}). Since an enclave generates both the enclave ID and key pair randomly, (eid, epk) will remain indistinguishable from (eid^{sim}, epk^{sim}). Consequently, *Hybrid 1* remains indistinguishable from the real-world scenario.

Hybrid 2: It works similarly to *Hybrid 1*, with a difference that $\sigma_{TEE}(epk)$ is replaced by $\sigma_{TEE}(epk^{sim})$. Due to the anonymous attestation, *Hybrid 2* remains indistinguishable from *Hybrid 1*.

Hybrid 3: Similar to the *Hybrid 2*, except that all the replies from CP are replaced with encryption of dummy messages of the same length. Since $\mathcal{AE}$ is IND-CPA secure, *Hybrid 3* is indistinguishable from *Hybrid 2*.

Notice, this *Hybrid 3* is exactly same as the ideal-world.

B. When $DU_{i,j}$ is corrupt: In this instance, the ideal functionality will not perform any actions. Therefore, there is nothing to simulate.

H.2 Sim Design - SendOrigData Stage:

A. When both DO and $DU_{1,1}$ are honest: As both parties are honest, $\mathcal{A}$ can only see and tamper the network messages, which is all that Sim needs to emulate. To do that:

1. Sim starts after $\mathcal{F}_{BPPM}$ notifies with ("SendOrigData", *DO*, $|PDS_{DO}[]|, |DS_{DO}[]|$).

2. Then Sim utilizes $|PDS_{DO}[]|, |DS_{DO}[]|$ and by following the real-world protocol steps, prepares PC^{sim} and D^{sim} with placeholder content.

3. Then using epk^{sim}, it encrypts them to $PC_{ct}^{sim}, D_{ct}^{sim}$.

4. Next Sim forwards $PC_{ct}^{sim}, D_{ct}^{sim}$ to $\mathcal{A}$.

5. If $\mathcal{A}$ delivers them to $DU_{1,1}$ without tampering, then Sim instructs $\mathcal{F}_{BPPM}$, to continue normally.

6. On the other hand, if $\mathcal{A}$ tampers these messages, then in the real-world, the enclave identifies these and aborts. Hence, our designed Sim also returns an abort signal in that case.

Hybrid 1: It works similarly to a real-world scenario, with the exception that (PC_{ct}, D_{ct}) is replaced by $(PC_{ct}^{sim}, D_{ct}^{sim})$. Since $\mathcal{AE}$ is IND-CPA secure, *Hybrid 1* is indistinguishable from the real-world.

In the real-world, the enclave can detect any alteration of D_{ct} or PC_{ct} and notifies with abort. Setp 6 of the Sim design ensures the same behavior in the ideal-world as well. Hence, the real-world and ideal-world remain indistinguishable, without requiring any further hybrid arguments.

B. When DO is honest but $DU_{1,1}$ is corrupted: Since DO is considered honest, all its functions will be managed by $\mathcal{F}_{BPPM}$. To maintain the scenario's indistinguishability from real-world operations, Sim must accurately replicate the corresponding protocol exchanges as they would occur with a real-world DO. Specifically, Sim is tasked with supplying the encrypted personal data-consent pair to the enclave of $DU_{1,1}$. However, according to the established privacy definition, $\mathcal{F}_{BPPM}$ does not disclose $DS_{DO}[]$, which means that Sim remains uninformed about it. As a result, Sim installs dummy personal data, into the enclave of $DU_{1,1}$. Hence:

1. Sim starts after getting notification from $\mathcal{F}_{BPPM}$.

2. Sim uses $|PDS_{DO}[]|$ and $|DS_{DO}[]|$ and prepares PC^{dummy} and D^{dummy} with dummy content.

3. Then using epk - the public key of $DU_{1,1}$'s enclave - Sim encrypts them to $PC_{ct}^{dummy}, D_{ct}^{dummy}$ and delivers those ciphertexts to the corrupted $DU_{1,1}$.

4. Sim then intercepts the communication between the corrupted $DU_{1,1}$ and G_{att}, by leveraging extraction method.

5. If Sim notices that $DU_{1,1}$ altered the input ciphertexts $(PC_{ct}^{dummy}, D_{ct}^{dummy})$ while calling EncSaveData entry point of G_{att}, Sim instructs $\mathcal{F}_{BPPM}$ to abort.

6. Otherwise, Sim instructs $\mathcal{F}_{BPPM}$, to continue normally.
7. As a result, $\mathcal{F}_{BPPM}$ stores the actual data-consent pair within its $DU_{1,1}$-specific data-structure, $DU_{1,1}.DList[]$.
8. Subsequently, Sim learns $PDS_{DO}[], did$ from $\mathcal{F}_{BPPM}$'s output.
9. Sim now utilizes the $PDS_{DO}[]$ to generate a simulated data-consent structure pair, denoted as PC^{sim} and D^{sim}, following the steps outlined in the real-world protocol. As no information regarding the public-key certificate of DO is made available to Sim, it depends on its own certificate, $Cert_{sim}$, and secret key, sk_{sim}, for this procedure. The objective is to establish a valid linkage between PC^{sim} and D^{sim}, ensuring they successfully pass the verification check conducted by the enclave. It is important to note, however, that the data attributes of $D^{sim}.DS[]$ still contains dummy values, as Sim does not possess knowledge of the actual $DS_{DO}[]$.
10. Next, using epk, Sim encrypts the pair into PC_{ct}^{sim} and D_{ct}^{sim}. Then, it allows the previously intercepted EncSaveData entry point call to G_{att}, along with these new ciphertexts.
11. Since PC^{sim} and D^{sim} are linked together, G_{att} will complete its execution normally and output the PC^{sim} (along with its valid attestation) containing $PDS_{DO}[]$ to $DU_{1,1}$.

Hybrid 1: It works similarly to the real-world scenario, except PC_{ct}, D_{ct} are replaced with $PC_{ct}^{dummy}, D_{ct}^{dummy}$. Since $\mathcal{AE}$ is IND-CPA secure, *Hybrid 1* remains indistinguishable from the real-world.

Hybrid 2: It works similarly to the *Hybrid 1*, except that the output of the enclave to $DU_{1,1}$ is replaced from PC to PC^{sim}.

$PC.PDS[]$ and $PC^{sim}.PDS[]$ are already the same. Since, both $PC.VK$ and $PC^{sim}.VK$ are generated randomly, making them indistinguishable. Due to the EU-CMA security of Σ, $PC.SC$ and $PC^{sim}.SC$ are also indistinguishable. Furthermore, since H is pre-image resistant *and* $DU_{1,1}$ never has access to the plaintext of D, the values $PC.DH$ and $PC^{sim}.DH$ effectively become indistinguishable. Consequently, both PC and PC^{sim} are indistinguishable overall, and *Hybrid 2* effectively mirrors the ideal world.

C. When DO is corrupted but $DU_{1,1}$ is honest: Since $DU_{1,1}$ is honest, it will not perform any actions, including not launching an enclave. Therefore, in this case, to deceive the adversary, Sim communicates with G_{att} and to ensure that the protocol exchange mirrors the real-world scenario. Specifically:

1. Sim activates upon receiving a ("SaveData", D_{ct}, PC_{ct}) call from the corrupted DO. Following this call, Sim communicates with G_{att}, similar to a real-world $DU_{1,1}$.

2. Specifically, Sim issues an (eid^{sim}, "resume", ("EncSaveData", D_{ct}, PC_{ct})) call to G_{att} for storing DO's data-consent pair within the simulator's enclave.

3. If G_{att} returns normally, it indicates that DO has not provided any unauthorized data. In this case, Sim learns did, PC and then invokes the SendOrigData entry point of $\mathcal{F}_{BPPM}$ and instructs to proceed normally.

   It is important to note that Sim requires four input parameters for invoking the SendOrigData entry point of $\mathcal{F}_{BPPM}$. However, since G_{att} returned (did, PC), it has only one input parameter $PDS_{DO}[] = PC.PDS[]$. Consequently, Sim generates a simulated data $DS^{sim}_{DO}[]$ by analyzing the size of the ciphertext, D_{ct}, observed in Step 1. Additionally, it uses a dummy ($Cert_{dummy}$) and secret key (sk_{dummy}) when calling the SendOrigData entry point of the ideal functionality.

   Consequently, $\mathcal{F}_{BPPM}$ is only able to store the actual approved data processing statements, $PDS_{DO}[]$, while the actual data is stored within Sim's enclave, identified with eid^{sim}. The fact that $\mathcal{F}_{BPPM}$ does not contain the actual data is addressed during the simulation of the ensuing stages.

4. On the other hand, if G_{att} aborts, Sim does not call $\mathcal{F}_{BPPM}$ at all.

D. When both DO and $DU_{1,1}$ are corrupt: Since both parties are corrupt, the ideal functionality will not be utilized by either party, eliminating the need for simulation at this stage. However, for the purposes of simulating the subsequent stages, Sim records the output PC from G_{att}. Then, akin to Step 3 of Case C, it invokes the SendOrigData entry point to install $PC.PDS[]$ within $\mathcal{F}_{BPPM}$. Nevertheless, the actual data part, $DS_{DO}[]$, remains secured solely within the enclave of the simulator.

Overall, due to the effects of simulation, the real world and the ideal world remain indistinguishable in all cases during the SendOrigData stage. $\mathcal{F}_{BPPM}$ consistently stores $PDS_{DO}[]$ for all the cases. However, $DS_{DO}[]$ is stored in $\mathcal{F}_{BPPM}$ only if DO acts honestly. If DO is not honest, $DS_{DO}[]$ is stored within the enclave of the simulator.

H.3 Sim Design - ForwardData Stage:

The situation in this stage has some alikenesses that of the SendOrigData stage. In particular, the role of cld in this stage is equivalent to the role of $DU_{i,j}$ in the SendOrigData

stage. Additionally, the operations of $DU_{i,j}$ share several similarities with those performed by DO in the SendOrigData stage. Consequently, the simulation steps in this stage exhibit significant resemblances to those of the SendOrigData stage.

A. When both $DU_{i,j}$ and cld are honest: Similar to Case A of SendOrigData stage, Sim generates indistinguishable network messages by utilizing $|DList[did].PDS[] - PDS_{rem}[]|$ and $|DList[did].DS[]|$, as notified by $\mathcal{F}_{BPPM}$.

B. When $DU_{i,j}$ is honest but cld is corrupted: Similar to Case B of SendOrigData stage, Sim installs simulated data-consent pair ($D^{sim}\&FPC^{sim}$) into the cld's enclave and generates network messages. Likewise, the corrupt cld acquires FPC^{sim} by the end of this process, which is indistinguishable from the real FPC.

C. When $DU_{i,j}$ is corrupted but cld is honest: It differs from the simulation to Case C of SendOrigData stage. Since $DU_{i,j}$ is corrupted, $\mathcal{F}_{BPPM}$ does not store the actual data; it is actually stored within the Sim's enclave (refer to Case B of SendOrigData and ForwardData stage). Specifically Sim performs the following:

1. Sim activates upon observing a ("EncPrepFrd",$did, PDS_{rem}[], ...$) call from the corrupt $DU_{i,j}$ to G_{att}.

   Notice: Since cld is honest, it itself did not install any enclave during the SetUp stage. Instead, Sim had installed an enclave on behalf of the honest cld (refer to Sim Design - Setup Case A) and published corresponding $eid, epk, \sigma_{TEE}(epk)$. Consequently, $DU_{i,j}$ uses those $epk, \sigma_{TEE}(epk)$ as the input parameters while calling EncPrepFrd entry point.

2. Sim notes down the first two input parameters, $did, PDS_{rem}[]$, of this call and allows the call unmodified to $DU_{i,j}$'s enclave.

3. If $DU_{i,j}$ performs any malicious action, G_{att} will abort, just like the real-world.

4. Otherwise, G_{att} will respond with ciphertexts. However, note that since $DU_{i,j}$ is corrupt, its enclave does not contain real data. As a result, the ciphertexts will represent mock data, which we handle in the simulation.

5. Then Sim waits for a (SaveData, D_{ct}, FPC_{ct}) call from the corrupt $DU_{i,j}$ to cld. When this call occurs, Sim records the identity of cld.

6. Sim also checks whether the ciphertexts returned by G_{att} in Step 4 are the same as those provided in the SaveData call. Any discrepancy between them indicates that $DU_{i,j}$ is attempting to feed malicious data to cld. Consequently, Sim aborts without invoking $\mathcal{F}_{BPPM}$.

7. Otherwise, Sim invokes the ForwardData entry point of $\mathcal{F}_{BPPM}$ using the parameters noted in Step 2 and Step 5 (the other parameters can remain as dummy values).

8. Subsequently, Sim instructs $\mathcal{F}_{BPPM}$ to continue normally.

   Consequently, $\mathcal{F}_{BPPM}$ copies the actual data processing statements relevant to cld into its internal data structure associated with cld. However, it's important to note that even if the data part is copied by the ideal functionality into the structure related to cld, whether the copied data is real or simulated depends on the honesty of DO.

Hybrid 1: It works similarly to the real-world scenario, with the distinction that the ciphertexts returned by G_{att} to $DU_{i,j}$ are substituted with ciphertexts representing random dummy content of same size. Given that $\mathcal{AE}$ is IND-CPA secure, *Hybrid 1* remains indistinguishable from the real-world situation. This *Hybrid 1* is same as the ideal-world.

D. When both $DU_{i,j}$ and cld are corrupt: Sim's operation is similar to Case D of the SendOrigData stage.

H.4 Sim Design - DataUsage Stage:

In this stage, only $DU_{i,j}$ is involved, and since there is no observable effect on the outside world, the simulator must only ensure that corrupt $DU_{i,j}$ receives the intended computation result along with its attestation, just as it would in a real-world scenario. Sim can invoke $\mathcal{F}_{BPPM}$ and subsequently utilize the equivocation method for that purpose. However, a crucial point in this stage is that if DO was not honest, then $\mathcal{F}_{BPPM}$ does not store the actual personal data at all. As a result, the simulation of this stage is influenced by the honesty of DO as well.

A. When $DU_{i,j}$ is honest and DO was honest: In this case, $DU_{i.j}$ achieves the correct output by consulting $\mathcal{F}_{BPPM}$. Since there is nothing observable to an external observer, there is nothing to simulate.

B. When $DU_{i,j}$ is honest and DO was corrupt: Given that $DU_{i,j}$ is honest and there is no observable impact for an external party, $\mathcal{Z}$ cannot observe this situation, rendering a simulation unnecessary. Nonetheless, for the sake of completeness, we demonstrate how the honest $DU_{i,j}$ receives the correct computation result.

If DO was corrupt, $\mathcal{F}_{BPPM}$ does not retain the actual data; rather, it is stored within the enclave of the simulator.

Therefore,

1. Sim starts after getting a ("Process", did, S) notification from $\mathcal{F}_{BPPM}$ and immidiately send abort message to $\mathcal{F}_{BPPM}$.

2. Then, Sim interacts with its own enclave by invoking (eid^{sim}, "resume", ("EncProcess", $did, S, \perp$)), which stores the actual data.

3. As a result, the enclave returns computation $result$ on the actual data.

4. Sim then invokes Process entry point of $\mathcal{F}_{BPPM}$ with ($did, S, y = result$).

5. Consequently, $\mathcal{F}_{BPPM}$ notifies the correct output $result$ to $DU_{i,j}$.

C. When $DU_{i,j}$ is corrupt and DO was honest: In this scenario, simulation is straightforward since $\mathcal{F}_{BPPM}$ possesses the actual personal data. Sim interacts with $\mathcal{F}_{BPPM}$ to obtain the computation result and then uses the equivocation method to generate the attestation of the output. Finally, Sim provides the computation result and the received attestation to $DU_{i,j}$.

D. When $DU_{i,j}$ is corrupt and DO was corrupt: In this case as well, neither $\mathcal{F}_{BPPM}$ nor $DU_{i,j}$'s enclave stores the actual data. Instead, simulator's enclave stores that.

Therefore,

1. Sim starts after observing a ("Process", $did, S, \perp$) call to G_{att} from $DU_{i,j}$.

2. Sim intercepts that call and invokes the simulator's enclave with the same parameter set.

3. As a result, the simulator's enclave returns the actual computation $result$, along with the attestation. However, that attestation is not tied with the identity of $DU_{i,j}$'s enclave.

4. Hence, Sim now changes $DU_{i,j}$'s original call to G_{att} from ("Process", $did, S, \perp$) to ("Process", $did, S, result$).

5. As a result, the $DU_{i,j}$'s enclave returns expected attestation on the actual computation $result$. Note: $DU_{i,j}$'s enclave also verifies, whether the requested processing is allowed or not. Hence, $DU_{i,j}$ see not difference.

Thus, we can conclude that $Prot_{BPPM}$ UC-realizes $\mathcal{F}_{BPPM}$. □