Crust: Verifiable and Efficient Private Information Retrieval With Sublinear Online Time

Yinghao Wang
Zhejiang University
Hangzhou, Zhejiang, China

Xuanming Liu
Zhejiang University
Hangzhou, Zhejiang, China

Jiawen Zhang
Zhejiang University
Hangzhou, Zhejiang, China

Jian Liu
Zhejiang University
Hangzhou, Zhejiang, China

Xiaohu Yang
Zhejiang University
Hangzhou, Zhejiang, China

ABSTRACT

Private Information Retrieval (PIR) is a cryptographic primitive that allows a user to access data from a database without disclosing the specific information being requested, thereby safeguarding privacy. PIR schemes suffer from a significant computational burden. By running an offline preprocessing phase, PIR schemes can achieve sublinear online computation. While protocols for semi-honest servers have been well-studied in both single-server and multi-server scenarios, scant attention has been given to scenarios involving malicious servers. In this study, we introduce a straightforward yet efficient sublinear PIR scheme named Crust. The scheme is tailored for verifiability and ensures privacy and data integrity against malicious servers. Our proposal is designed to function under two configurations: (i) with two non-colluding servers, or (ii) with a standalone single server. Apart from its verifiability, our scheme demonstrates notable efficiency. Regarding online computation efficiency, our scheme outperforms state-of-the-art two-server schemes by a factor of 16 and single-server sublinear PIR schemes by a factor of 6. Furthermore, relative to leading verifiable PIR schemes, our scheme showcases approximately 1000 times greater efficiency. To the best of our knowledge, this is the first PIR scheme to achieve both verifiability and amortized $O(\sqrt{n})$ online computation.

1 INTRODUCTION

Private Information Retrieval (PIR) is a significant cryptographic primitive in which servers possess a database structured as an array of N records. A client seeks to retrieve the i-th element without revealing the index i to the servers. PIR finds application in various real-world scenarios such as private database search [22], private media consumption [12], and credential breach reporting [23].

PIR schemes with sublinear online time. To improve the efficiency of PIR, a series of studies have focused on developing more efficient PIR protocols where the online computation is sublinear in the database size. These sublinear PIR schemes, by providing improved asymptotic efficiency, enable the application of PIR to large databases. Corrigan-Gibbs and Kogan [7] (hereinafter referred to as CK20) introduced an *offline-online* sublinear PIR construction, which includes an offline preprocessing phase followed by an online query phase. In this design, a single query requires the servers to perform approximately $\tilde{O}(\sqrt{N})$ computation and communication. This offline-online model acts as a foundational architecture for sublinear PIR schemes, paving the way for a series of subsequent research efforts [6, 15, 16, 21, 27].

Verifiability. The advent of cloud computing and the growing trend of outsourcing computing tasks have led to a critical reevaluation of the security model foundational to PIR. Most current PIR schemes [7, 13, 17, 18, 21] operate under the assumption of semi-honest servers, where the servers execute the protocol faithfully, except for trying to deduce the queried index from the interactions. Nonetheless, there exists the risk of malicious servers either deliberately providing false responses or doing so accidentally due to system failures. The necessity for verifiability in PIR is as crucial as in conventional non-private database queries. For instance, the "Safe Browsing" service in Checklist [15] underscores a scenario where malicious servers could mislead clients with incorrect answers, potentially directing them towards harmful websites. Another scenario involves a PIR server managing private DNS queries [25], where a malevolent server could carry out a DNS poisoning attack [8], obstructing users' access to specific online services or redirecting them to fake sites operated by attackers.

Selective failure attack. Adapting PIR protocols to ensure verifiability against malicious servers presents non-trivial challenges. The existing research has identified that standard data authentication techniques are susceptible to a particular type of assault known as *selective failure attacks* [14]. In such attacks, a malicious server deliberately selects specific database records to respond to a query, aiming to infer the client's queried index i based on the outcome of the client's verification. For instance, even when records are protected by digital signatures, a malicious server might replace the first record in the database with an arbitrary item and a counterfeit signature. Should the client query the first record and reject the response due to the invalid signature, the server is then able to deduce that the first record was the target of the query. Therefore, to mitigate this type of attack, in this work, we will adopt a more robust privacy definition.

Previous research [5, 9] has introduced verifiable PIR schemes as measures against malicious servers and selective failure attacks. However, these schemes are limited by requiring linear online computation for the servers, which restricts their applicability to large databases. To the best of our knowledge, no existing verifiable PIR scheme simultaneously achieves *sublinear computation* and ensures *privacy against selective failure attacks*. Thus, in this work, we revisit the domain of sublinear PIR schemes with the objective of addressing this gap. Consequently, we present Crust, comprising two verifiable PIR schemes: one tailored for a single-server setting and another for a two-server setting, both of which offer sublinear

computation and robust privacy safeguards against selective failure attacks.

1.1 Our Contributions

Our contributions can be summarized as follows:

- We conduct an in-depth analysis of existing sublinear PIR constructions, pinpointing the obstacles that hinder the integration of verifiability. Specifically, we highlight the challenge of incorporating verifiability into existing sublinear PIR schemes without compromising privacy against selective failure attacks. To address this challenge, we introduce a novel construction of sublinear PIR against a semi-honest adversary in the two-server setting, which later can be smoothly transitioned into efficient verifiable PIR schemes.
- For potential malicious attacks, we present Crust, comprising two innovative and efficient verifiable PIR schemes: one tailored for the two-server setting and the other for the single-server setting. Notably, these schemes represent the first instance of verifiable PIR to concurrently achieve amortized $O(\sqrt{N})$ server computation and communication efficiency (where N is the size of the database) and provide privacy protection against selective failure attacks. We devise a *pluggable* verification procedure that ensures data integrity without adding to the asymptotic complexity. This procedure is integrated with the aforementioned new PIR scheme to accomplish Crust. Our two-server scheme provides verifiability in scenarios involving two non-colluding servers, with one being semi-honest and the other malicious. The single-server scheme ensures data integrity against a malicious server, assuming the client holds a database digest from a trusted database owner. A comparison for Crust and other advanced PIR schemes is provided in Table 1.
- We have implemented our schemes and conducted a comprehensive evaluation of the state-of-the-art schemes. In a semi-honest scenario (excluding verification), our construction surpasses the leading two-server scheme [16] by a factor of 16 and single-server schemes by 6 times in online computation efficiency. In terms of verifiability, the online computation efficiency only experiences a modest 2 to 3 times increase, leading to approximately 1000 times greater online computation efficiency compared to the leading single-server verifiable PIR scheme [9], thereby establishing our verifiable PIR schemes as a practical choice for real-world applications.

1.2 Technical Overview

Our construction targets two primary objectives: *verifiability* and *efficiency*. The aim is to incorporate verifiability into sublinear PIR schemes without compromising efficiency.

- **The problem of dummy queries.** Recent works [15, 16, 19] introduce additional fake queries, referred to as dummy queries, to conceal the queried index, thereby securing the privacy of the scheme. However, we have found that these dummy queries are susceptible to selective failure attacks and can lead to a significant increase in online computation, becoming a major obstacle in achieving efficient verifiable PIR. We discuss this issue in detail in Section 4.1.
- **An efficient PIR without dummy queries.** We propose a new sublinear PIR construction that is efficient and avoids the use of dummy queries. To enhance computational and communication efficiency, we incorporate the technique of dividing the database into blocks and using a pseudorandom function to compress queries and storage. Furthermore, we introduce a new concept termed "crumbs", retrieving extra records as hints to replace the dummy queries. Consequently, we achieve an even more efficient two-server PIR scheme against a semi-honest adversary, which is detailed in Section 4.2.
- **A pluggable verification procedure.** Turning to defense against malicious attacks, we design a pluggable verification procedure that ensures data integrity with slightly additional overhead. The main idea of this procedure is to utilize an extra hint to verify the result. We combine this plugin with our newly designed PIR scheme to achieve a sublinear two-server verifiable PIR, which is discussed in Section 5.2.
- **Adapting for single-server.** Finally, we utilize standard techniques [6, 27] to adapt our two-server scheme for single-server setting. We discuss this transition in detail in Section 5.3.

2 RELATED WORKS

2.1 Multi-server PIR

In schemes involving multiple servers (multi-server PIR), the client interacts with $k > 1$ servers, each holding a copy of the database. Traditional multi-server PIR schemes have low communication costs but involve linear computational overhead. Corrigan-Gibbs and Kogan introduced the first sublinear PIR scheme in [7]. This scheme, which consists of one client and two servers, is divided into an offline phase and an online phase. During the offline phase, the client obtains some additional information called hints from one server about the database. In the online phase, the client utilizes these hints at the other server to make queries. At the core of their construction is a primitive named *Puncturable Pseudorandom Function* (PPRF). The scheme needs parallel instances to ensure its correctness. As a result, the scheme performs a single online query in $\tilde{O}(\lambda\sqrt{N})$ online time (where λ is the security parameter), as well as incurring an $\tilde{O}(N^{5/6})$ client storage requirement, rendering it impractical for real-world applications. This framework is further followed by a series of works, including [6, 15, 16, 21, 27].

Some of these works aim to improve the efficiency of the original framework. The construction of hints in [7] does not support efficient membership testing. To search for a suitable hint while making an online query, the client either needs to perform approximately $O(N)$ computations or maintain a searching result cache with a size of $\tilde{O}(N^{5/6})$. Neither option is optimal. Hence, works like [15, 16, 21] focus on leveraging improved PPRFs to reduce online computations, communication costs, and client storage. For instance, [16] presents a more practical construction named TreePIR along with a new primitive called *Weakly Puncturable Pseudorandom Function*, which facilitates efficient membership testing without significant client storage requirements. However, this scheme introduces the complexity of "dummy queries", which will be elaborated upon in this paper. Nevertheless, the scheme remains loyal to the original PPRF-based approach, which, as demonstrated in our benchmark, is costly in practical scenarios.

Table 1: A comparison of Crust and other state-of-the-art PIR schemes. Green cells indicate verifiability and represent the best theoretical asymptotic complexity. The costs are averaged over $\sqrt{N}$ queries.

Scheme	Servers	Communication	Computation	Verifiable
TreePIR [16]	2	$O(\sqrt{N})$	$O(\sqrt{N}\log^2 N)$	✗
DPF-PIR [11]	2	$O(\log N)$	$O(N)$	✗
APIR(Merkle) [5]	2	$O(\sqrt{N}\log N)$	$O(N\log N)$	✓
Crust (Theorem 1)	2	$O(\sqrt{N})$	$O(\sqrt{N})$	✓
SimplePIR [13]	1	$O(\sqrt{N})$	$O(N)$	✗
VeriSimplePIR [9]	1	$O(\sqrt{N})$	$O(N)$	✓
Piano [27]	1	$O(\sqrt{N})$	$O(\sqrt{N}\log N)$	✗
APIR(LWE) [5]	1	$O(\sqrt{N})$	$O(N)$	✓
Crust (Theorem 2)	1	$O(\sqrt{N})$	$O(\lambda\sqrt{N})$	✓

2.2 Single-server PIR

There is another line of works [6, 27] focusing on building single-server PIR schemes based on [7]. The transition is non-trivial as hints cannot be retrieved and utilized on the same server. Corrigan-Gibbs et al. [6] applied the offline-online sublinear PIR framework to the single-server setting. However, the inefficiencies of the two-server scheme also apply to the single-server scheme, necessitating parallel instances for correctness and resulting in large client storage requirements. This work was extended by Piano [27], combining elements from [16] and [6] to create a straightforward and efficient single-server scheme with amortized computation complexity of $O(\lambda\sqrt{N})$ and communication complexity of $O(\sqrt{N})$. Although this scheme represents a step towards improved efficiency, it introduces dummy queries that impede the integration of verifiability into sublinear schemes, as will be discussed in Section 4.

2.3 Verifiable PIR

Some works have explored the concept of verifiable PIR, initially introduced by Chor et al. [4] and further investigated by [1, 10, 24, 26]. The most recent works include [5, 9]. In [5], Colombo et al. give a thorough definition of verifiable PIR (in their work, they refer to it as authenticated PIR). The authors propose a straightforward approach for multi-server verifiable PIR, leveraging precomputed Merkle Proofs, and a method for predicate queries utilizing function secret sharing [2, 3]. They also present constructions for single-server verifiable PIR. Additionally, they introduce the concept of selective failure attack in PIR queries. However, the proposed schemes exhibit inefficiencies in both computation and communication, making them impractical. In another work [9], verifiability is introduced to the single-server scheme SimplePIR [13] with minimal computation overhead and only a 1.1-1.5× communication overhead. However, these contributions build upon existing PIR constructions that involve online computation with a complexity of $O(N)$. To the best of our knowledge, there have been no previous endeavors toward imbuing sublinear PIR schemes with verifiability. This study aims to bridge this gap by extending the notion of verifiability to sublinear PIR schemes.

3 PRELIMINARIES

In this paper, we explore both two-server and single-server PIR schemes. We assume a data owner O has a database $\mathcal{D}$ with N records, where $\mathcal{D} = (\mathcal{D}_0, \ldots, \mathcal{D}_{N-1})$ and each record is an element in $\mathbb{F}_{2^p}$. Further, we assume that N is a perfect square, allowing for the expansion of the database if this condition is not initially met. For the two-server PIR scheme, our focus is on a client, C, and two non-colluded servers, $\mathcal{S}_{\text{hint}}$ and $\mathcal{S}_{\text{query}}$. In the single-server setting, our attention shifts to a single server $\mathcal{S}$. We adopt the notation $[k]$ to represent the set $\{0, 1, \ldots, k-1\}$. The notation $S[j]$ on a set $S = \{S_0, S_1, \ldots\}$ is utilized to signify a set where $S[j] = S_j$. The symbol λ is used to denote the security parameter, and $\text{negl}(\cdot)$ indicates the negligible function. We use ":=" to denote a value assignment.

3.1 Offline-online PIR

In this section, we present a formal definition of PIR schemes operating in an offline-online model, as introduced in [7].

Definition 3.1 (Offline-online PIR)**.** An *offline-online PIR* scheme Π allows a client to retrieve a record $\mathcal{D}_i$ from the database $\mathcal{D}$ without revealing the index i to any of the k servers. The scheme consists of algorithm tuple $\Pi = (Setup, Hint, Query, Answer, Reconstruct)$:

- $Setup(1^\lambda, N) \to (ck, q_h)$. It generates a client key ck and hint queries q_h given database size N and the security parameter λ.
- $Hint(\mathcal{D}, q_h) \to h$. It generates a hint h from the database $\mathcal{D}$ and hint queries q_h.
- $Query(ck, i) \to q$. It generates a query q given the hint h and the index to query i. Note that the query q may consist of multiple sub-queries.
- $Answer(\mathcal{D}, q) \to a$. It generates an answer a in response to the query q.
- $Reconstruct(h, a) \to \mathcal{D}_i$. It reconstructs the record $\mathcal{D}_i$ with the help of hint h, given an answer a.
- $Refresh(ck, h, a) \to (ck, h)$. It updates the hint h, given the answer a.

The scheme is required to adhere to two fundamental properties: **Correctness** and **Privacy**. The correctness guarantees that the client obtains the correct record $\mathcal{D}_i$ with overwhelmingly high probability, while privacy ensures that the servers gain no information about the index i during the query process. A more detailed definition is available in Appendix B.

This study concentrates primarily on the two-server setting (where $k = 2$) and the single-server setting (where $k = 1$). In the

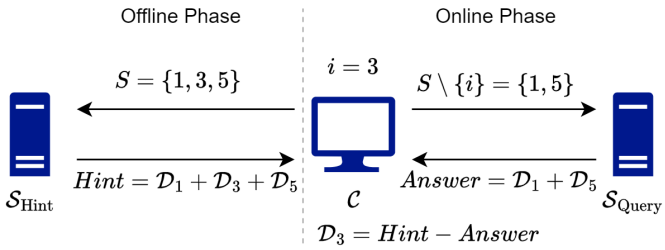


Figure 1: An illustration of CK20 [7] scheme, querying $i = 3$.

two-server scenario, we assume non-colluding servers throughout the scheme execution, consistent with prior works such as [5, 7]. Here, hints typically consist of a set containing the indexes of specific records within the database $\mathcal{D}$, combined with a parity computed from these records. The terms "parity" or "hint parity" are used to denote the parity included in a hint, while "set" or "hint set" refers to the set enclosed within the hint.

3.2 Selective Failure Attack

In verifiable PIR schemes [5, 9, 26], the integrity of the answer provided by the server(s) to the client is typically ensured by enabling the client to conduct additional verification protocols. However, revealing that the answer is rejected can potentially divulge information about the index being queried to an adversary. The concept of a *selective failure attack*, initially introduced in [14], exploits this information to the adversary's advantage. As discussed in the introduction, a malicious server could manipulate records in the database selectively, thus deducing information about the queried index i from the outcomes of the verification. Privacy against a selective failure attack can be formally defined as follows:

Definition 3.2 (Privacy Against Selective Failure Attack)**.** Given a security parameter λ, A PIR scheme Π is private against selective failure attack if there exists a probabilistic polynomial time simulator $Sim(1^\lambda, N)$ such that for an algorithm *serv* following the protocol in Π honestly, any probabilistic polynomial time adversaries $\mathcal{A}$ corrupting one of the k servers, $\mathcal{A}$ cannot distinguish the view in the following worlds except for probability negligible in λ.

- **World 0**: C interact with $\mathcal{A}$ who plays the role of the corrupted server and *serv* who plays the role of honest servers. At each step t, $\mathcal{A}$ adaptively chooses the next index i_t and C use i_t as its query. The client outputs 1 to $\mathcal{A}$ after each query if the answer is accepted, otherwise 0.
- **World 1**: *Sim* interact with $\mathcal{A}$ who plays the role of the corrupted server and *serv* who plays the role of honest servers. At each step t, $\mathcal{A}$ adaptively chooses the next index i_t and *Sim* runs without the knowledge of i_t. *Sim* output a bit b to $\mathcal{A}$ after each query.

  $\mathcal{A}$ is allowed to deviate from the protocol arbitrarily.

Prior work [5] attempted to mitigate this attack by associating the database with a digest and conducting linear computation on the database. However, their approach did not lead to a sublinear PIR scheme due to their reliance on PIR schemes that entail linear computations.

4 A CONSTRUCTION OF EFFICIENT TWO-SERVER PIR

We commence by presenting a construction for a two-server PIR scheme in a semi-honest setting. Our construction is based on the CK20 framework. We adopt a novel approach to constructing hints and queries. This approach is designed to facilitate efficient querying while ensuring that the construction does not hinder the integration of verifiability.

4.1 Base Framework for Two-server PIR

Since we have chosen to build verifiable PIR based on existing sublinear PIRs [7, 16], a foundational understanding of these schemes is imperative. We start with a "strawman" two-server PIR scheme, followed by a discussion on rectifying the limitations of the protocol. The strawman protocol is an offline-online PIR:

Offline phase:

- **Setup:** The client C samples M sets $sk_j, j \in [M]$, each contains s random indexes in range $[N]$.
- **Hint:** C forwards these sets to the hint server $\mathcal{S}_{\text{hint}}$. $\mathcal{S}_{\text{hint}}$ calculates parity for the sets as $h_j := \sum_{k \in sk_j} \mathcal{D}_k, j \in [M]$. C stores these sets and their corresponding parities as the hint.

Online phase:

- **Query:** C first identifies a set sk_t containing the desired index i and computes the set excluding this specific index, represented as $sk'_t := sk_t \setminus \{i\}$. Additionally, C samples a new set $\overline{sk}$ containing the index i and $\overline{sk}' := \overline{sk} \setminus \{i\}$. The set $\overline{sk}'$ is sent to $\mathcal{S}_{\text{hint}}$, while sk'_t is sent to $\mathcal{S}_{\text{query}}$.
- **Answer:** Upon receipt of these sets, the servers calculate the parities for the two sets as $a_{\text{query}} := \sum_{j \in sk'_t} \mathcal{D}_j$ and $a_{\text{hint}} := \sum_{j \in \overline{sk}'} \mathcal{D}_j$, respectively, and return these parities to C.
- **Reconstruct:** C reconstructs the desired record through the calculation $\mathcal{D}_i := h_t - a_{\text{query}}$.
- **Refresh:** C updates the used hints by setting $sk_t := \overline{sk}$ and $h_t := a_{\text{hint}} + \mathcal{D}_i$.

The correctness of the protocol is straightforward. Nonetheless, there are three critical issues inherent to the strawman protocol, which are listed as follows:

(1) **Efficient membership testing.** Searching for an appropriate set sk_t that contains the queried index i during the online phase is computationally expensive.
(2) **Space and communication efficiency.** It is inefficient for the clients to sample, store, and send the sets in plain, incurring both $O(N)$ storage and offline communication complexity for the client to finish the protocol.
(3) **Privacy.** The sets sk'_t and $\overline{sk}'$ never contains the queried index i, effectively leaking some information about the index to query.

In existing literature, various approaches have been proposed to tackle the challenges above. For example, [16] resolved the above problems and introduced a sublinear PIR construction, leveraging a primitive called *Weakly Puncturable Pseudorandom Function*. [27] proposed a similar algorithm they name *PossibleParities*. However, when it comes to verifiable PIR, the adaptability of these schemes encounters obstacles, primarily due to the incorporation of "dummy queries", which will be detailed and elaborated on in the following.

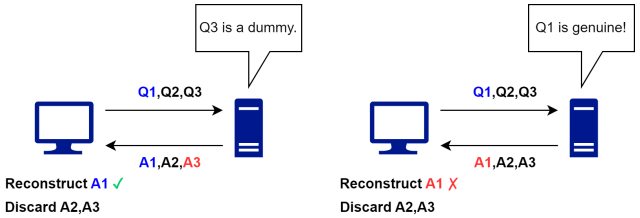


Figure 2: An illustration of dummy queries. The client sends 3 queries to the server, within which the first query is the genuine query and the other two are dummy queries. Since the client only rejects a false answer to the genuine query but accepts a false answer to a dummy query, the server can manipulate the records and infer information from the client's verification outcome.

The problem of dummy queries. Recent studies [15, 16, 19] following [7] have introduced the concept of "dummy queries". These dummy queries are randomly generated fake queries to obfuscate the genuine query. In these schemes, a single query sent from the client leaks information about the queried index i to the servers. It is the combination of the genuine query and the dummy queries that effectively conceals the index i. The servers respond to dummy queries, while the client subsequently disregards the answers to the dummy queries.

In the semi-honest setting, the scheme's privacy hinges on the assumption that the servers cannot distinguish the genuine query from the dummy ones. However, in the context of verifiable PIR and the consideration of selective failure attacks, dummy queries introduce complications. During the verification process, the client only rejects the servers' responses if the answer to the genuine query has been altered. This is due to the client's lack of knowledge about the randomly generated dummy query, preventing the client from reconstructing and verifying the answer to a dummy query. Given this scenario, the server can arbitrarily choose the answer it tampers with and can deduce from the client's verification result whether the altered answer was the genuine one. This vulnerability allows the malicious server to potentially learn which record the client is interested in, impeding the incorporation of verifiability into existing sublinear PIR schemes. An illustration of this problem is presented in Figure 2. Apart from privacy concerns, computing dummy answers also escalates online computational costs. Balancing efficiency and addressing selective failure attacks necessitates that we **eliminate dummy queries** in our scheme.

4.2 Efficient PIR Without Dummy Queries

In this subsection, we present a new design for an efficient two-server PIR scheme that avoids the use of dummy queries, yet preserves the efficiency of query performance. We follow the strawman protocol and try to address the aforementioned issues with three optimization techniques: (i) enabling efficient membership testing with a "divide-and-sample" technique, (ii) compressing the sets with a pseudorandom function to improve space and communication efficiency, and (iii) hiding the query index with a new concept called "crumbs". The techniques are detailed as follows:

Enabling efficient membership test. In the strawman protocol, the indexes in each set $sk_j, j \in [M]$ are generated randomly, which impedes efficient membership test. Drawing inspiration from existing literature [16, 27], we employ a strategy of dividing the database into $\sqrt{N}$ blocks, each comprising $\sqrt{N}$ elements. We generate each set S in the following way: within each block j, sample a random offset $x_j, j \in [\sqrt{N}]$, these offsets would map to the actual set $S = \{x_j + j \cdot \sqrt{N} \mid j \in [\sqrt{N}]\}$.

This method enables efficient membership testing. To determine if the queried index i is in a set S, the client evaluate the $\lfloor i/\sqrt{N} \rfloor$-th item of S and see if it equals $i - \sqrt{N} \cdot \lfloor i/\sqrt{N} \rfloor$. This results in the necessity of keeping the set **ordered**, in order to determine the block an offset belongs to. Henceforth, we assume that a set can be ordered and indexed similarly to a vector, which facilitates simplicity.

Compressing sets with pseudorandom function. Previous works [7, 16] choose to utilize a puncturable pseudorandom function to compress the sets, thereby decreasing the communication cost. However, the puncturable pseudorandom function primitive is costly in practice, as it incurs extra computation when evaluating the set. Contrastingly, we opt for a plain pseudorandom function. Concretely speaking, with a pseudorandom function parameterized by a key sk, i.e., $f_{sk} : [\sqrt{N}] \to [\sqrt{N}]$, a set S could be succinctly represented with a short λ-sized key sk: $S = \{f_{sk}(j) + j \cdot \sqrt{N} \mid j \in [\sqrt{N}]\}$ where λ is the security parameter. Thus, the membership testing can be expressed as:

$$f_{sk}(\lfloor i/\sqrt{N} \rfloor) = i - \sqrt{N} \cdot \lfloor i/\sqrt{N} \rfloor$$

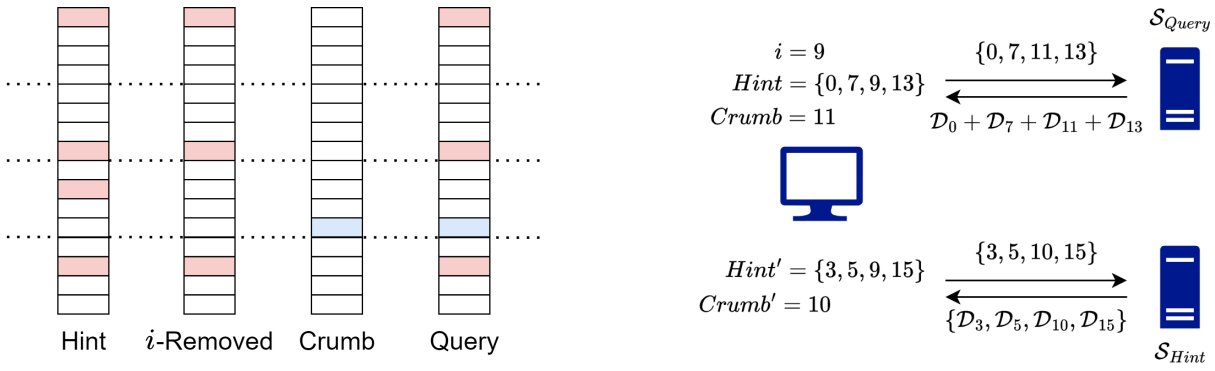
For reference convenience, we denote the process of evaluating f to a set S as $Expand$.

By abandoning the puncturable pseudorandom function primitive, we achieve a significant performance boost. However, this decision comes at the expense of losing the capability to remove an element from the set while maintaining a succinct representation. Consequently, the set needs to be depicted with a simple list of values. The client retains and transmits offline each hint in $O(\lambda)$ size, and sends online the plain set with $O(\sqrt{N})$ communication.

Guarding privacy with crumbs. The non-puncturable pseudorandom function primitive offers reasonable space and communication efficiency, albeit it raises a new challenge: concealing the removed indexes while preserving item order. To address this challenge, we propose the client retrieves additional hints, termed **crumbs**. A crumb $c = (u, o)$ consists of a random record u and its corresponding offsets in its block o. During the offline phase, the client fetches one random crumb from each block, resulting in a total of $\sqrt{N}$ crumbs.

The idea is to utilize crumbs to mask the removed item. Specifically, when the client seeks to query the i-th record, after discovering i from the set, it finds a crumb $c = (u, o)$ in the $\lfloor i/\sqrt{N} \rfloor$-th block, and replaces the index with o. This process is illustrated in Figure 3.

As each crumb contains merely one bare record, the true answer, which excludes the crumb, can be easily obtained by excluding the crumb's contribution from the answer. Concretely, with a as the answer from $\mathcal{S}$, u as the value of the used crumb, the client updates



(a) Query construction **(b) Query process**

Figure 3: An illustration of a query. The database contains $N = 16$ items and is divided into $\sqrt{N} = 4$ blocks. a) A hint contains one item in each block. The client wants to query record $i = 9$, which is in the third block. The index is removed from the set, and a crumb in the third block is added to the set. b) The query server answers with the sum of the set. The client samples a new hint set to refresh the hint and crumb. The hint server answers with individual records. The index are written in plain for illustration. They are actually represented with offsets in blocks.

the answer disregarding the crumb as $a := a - u$. Finally, the client can reconstruct the record $\mathcal{D}_i$ in the same way as in the strawman protocol.

The obfuscation of removed items by crumbs is evident. The set sent by the client to the query server now comprises one random offset in each block, unrelated to the queried index i. Our construction maintains sublinear efficiency: in the offline phase, the client receives $O(\lambda\sqrt{N})$ sized parities which are evaluated by the server in $O(\lambda N)$ time. When amortized over polynomial times queries in the online phase, each query incurs $O(\sqrt{N})$ communication and computation. Note that in our techniques, we do not use dummy queries. Jumping ahead, this approach allows us to implement a smooth verification procedure, which will be detailed in Section 5.

4.3 Hint Retrieving and Refreshing

Retrieving and refreshing hints are also critical aspects of the offline-online PIR scheme. It is essential for every record to be associated with at least one hint; without this, querying the record during the online phase would be unfeasible. Furthermore, the scheme must accommodate multiple queries and spread the significant offline costs over these queries to ensure efficiency. The client cannot use a hint twice as the server would learn about the indexes by comparing the queries. Therefore, hints must be refreshed after being consumed in one query.

The offline retrieval of hints and crumbs is straightforward. Given that the hint server $\mathcal{S}_{\text{hint}}$ is semi-honest, all the hint sets can be generated from one PRF key supplied by the client. The crumbs can be fetched in the same way, by asking the server to send all the records in one set rather than the sum of them.

The online refreshing of hints and crumbs is more complex. In the strawman protocol, refreshing is accomplished by the client sampling a new set that includes the queried index i and interacting with the hint server to acquire the hint for this set. However, in our protocol, crumbs derived from the hint server cannot be employed to obscure the removed item from the hint server. Otherwise, the hint server can compare the query with the crumbs it sent to the client, thus learning which block is replaced by a crumb. To resolve this discrepancy, the client selects a new random index j to substitute i within the block l of the newly sampled set. To inform the client about $\mathcal{D}_j$, the hint server responds to the query with $\sqrt{N}$ plain records, each corresponding to an individual index in the set. This allows the client to construct a new hint from the records along with $\mathcal{D}_i$ and to update the utilized crumb with $\mathcal{D}_j$. By adopting this method for refreshing hints and crumbs, our scheme is now capable of supporting a polynomial number of queries, in line with the recent advancements discussed in [16]. A comprehensive proof is provided in Appendix B.1. We give the full protocol in Figure 4.

Handling Changing Databases. Handling changing databases is a frequent situation in applications. The optimizations and techniques used in our approach integrate smoothly with current methods for managing database modifications, including Checklist [15].

5 EFFICIENT VERIFIABLE PIR

In this section, we describe a technique for embedding verifiability into a PIR scheme as a plugin. The concept of "plugin" signifies that verifiability can be incorporated without modifying the foundational structure of the original scheme. We will provide a comprehensive design for such an enhancement to the PIR scheme introduced in Section 4. Consequently, we introduce Crust, an efficient framework that includes both an efficient two-server verifiable PIR scheme and an efficient single-server verifiable PIR scheme.

5.1 Definition

We first formally present the concept of *Verifiable Private Information Retrieval* (verifiable PIR), which extends the notion of offline-online PIR as defined earlier.

Two-server Scheme of Crust

Notation: The scheme involves a client C, a query server $\mathcal{S}_{\text{query}}$ and a hint server $\mathcal{S}_{\text{hint}}$. Assume a hint consists of a tuple of elements: $h = (sk, h^s, sr, h^r)$. Assume a crumb consists of an offset and a value $c = (o, u)$. Assume $fk_{key} : \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a PRF that maps one PRF key to many keys. *Expand* a key sk involves calculating the set $\{f_{sk}(j) + j \cdot \sqrt{N} \mid j \in [\sqrt{N}]\}$. *Expand* a key sr involves calculating the set $\{fr_{sr}(j) \mid j \in [\sqrt{N}]\}$. Suppose i is the queried index.

Offline Phase:

- **Setup:** C samples one PRF key $mk \in \{0,1\}^\lambda$, initializes the hints to $h_j := (fk_{mk}(j), 0, fk_{mk}(j+M), 0), j \in [M]$, and the crumbs $c_j := (\perp, \perp), j \in [\sqrt{N}]$. C sends mk to the $\mathcal{S}_{\text{hint}}$.
- **Hint:**
  - $\mathcal{S}_{\text{hint}}$ evaluates mk to PRF keys $sk_j, j \in [M]$ and $sr_j, j \in [M]$. It further *Expand*s these keys to sets $S_j, j \in [M]$ and $SR_j, j \in [M]$.
  - $\mathcal{S}_{\text{hint}}$ calculates the parities $h_j^s := \sum_{k \in [\sqrt{N}]} \mathcal{D}_{S_j[k]}, j \in [M]$ and $h_j^r := \sum_{k \in [\sqrt{N}]} \mathcal{D}_{S_j[k]} \cdot SR_j[k], j \in [M]$ for each set. The parities are sent to C.
  - In the j-th $\sqrt{N}$ sized block ($j \in [\sqrt{N}]$), $\mathcal{S}_{\text{hint}}$ samples one record u_j as crumb, along with its corresponding offset o_j. It then sends $c_j := (o_j, u_j), j \in [\sqrt{N}]$ to C. C stores these crumbs.

Online Phase:

- **Query ($\mathcal{S}_{\text{query}}$):**
  - Denote the block that i is in as $l := \lfloor i/\sqrt{N} \rfloor$. C finds a hint $h_t = (sk_t, h_t^s, sr_t, h_t^r)$ satisfies that $f_{sk_t}(l) = i - \sqrt{N} \cdot l$. If no such hint exists, C samples a new hint with zero parities $(sk_t, h_t^s := 0, sr_t, h_t^r := 0)$.
  - C *Expand*s sk_t to a set S. It finds the crumb for this block $c_l = (o_l, u_l)$, and then replaces $S[l]$ with crumb offset o_l. C *Expand*s sr_t to set SR, and $SR[l]$ is replaced by a random element r. C sends S and SR to $\mathcal{S}_{\text{query}}$.
- **Answer ($\mathcal{S}_{\text{query}}$):** $\mathcal{S}_{\text{query}}$ calculates the parities $a^s := \sum_{k \in [\sqrt{N}]} \mathcal{D}_{S[k]}$ and $a^r := \sum_{k \in [\sqrt{N}]} \mathcal{D}_{S[k]} \cdot SR[k]$. It sends a^s and a^r to C.
- **Reconstruct:** C reconstructs the answer $\mathcal{D}_i := h_t^s - (a^s - u_l)$. It then verifies the answer by checking if $h_t^r - (a^r - r \cdot u_l) = fr_{sr_t}(l) \cdot \mathcal{D}_i$. If the verification fails, C outputs $\perp$. Otherwise, C outputs $\mathcal{D}_i$.
- **Query ($\mathcal{S}_{\text{hint}}$):**
  - C samples a new PRF key $\overline{sk}$ with $f_{\overline{sk}}(l) = i - \sqrt{N} \cdot l$. C samples a new PRF key $\overline{sr}$.
  - C *Expand*s $\overline{sk}$ to a set $\overline{S}$, *Expand* $\overline{sr}$ to a set $\overline{SR}$. It then replaces $\overline{S}[l]$ with a random offset $\overline{o}$ in block l. C sends $\overline{S}$ to $\mathcal{S}_{\text{hint}}$.
- **Answer ($\mathcal{S}_{\text{hint}}$):** $\mathcal{S}_{\text{hint}}$ responds with $\sqrt{N}$ entries $\mathcal{D}_{\overline{S}[j]}, j \in [\sqrt{N}]$.
- **Refresh:** C updates $c_l := (\overline{o}, \mathcal{D}_{\overline{S}[l]})$, updates $h_t := (\overline{sk}, \mathcal{D}_i + \sum_{k \in [\sqrt{N}], k \neq l} \mathcal{D}_{\overline{S}[k]}, \overline{sr}, SR[l] \cdot \mathcal{D}_i + \sum_{k \in [\sqrt{N}], k \neq l} SR[k] \cdot \mathcal{D}_{\overline{S}[k]})$

Figure 4: The two-server scheme of Crust. The red part is the verification procedure and is utilized as a plugin, and this part can be removed to achieve an even more efficient PIR scheme in a semi-honest setting.

Definition 5.1 (Verifiable Private Information Retrieval)**.** A *verifiable private information retrieval* scheme Π allows a client to retrieve a record $\mathcal{D}_i$ from the database $\mathcal{D}$ without revealing the index i to any of the k servers. The scheme consists of a tuple of algorithms $(Setup, Hint, Query, Answer, Reconstruct, Refresh)$:

- $Setup(1^\lambda, N) \to (ck, q_h)$. It generates a client key ck and hint queries q_h given database size N and the security parameter λ.
- $Hint(\mathcal{D}, q_h) \to \{h, \perp\}$. It refuses the database $\mathcal{D}$ and output $\perp$, or generates a hint h, given the database $\mathcal{D}$ and hint queries q_h.
- $Query(ck, i) \to q$. It generates a query q given the hint h and the index to query i. Note that the query q may consist of multiple sub-queries.
- $Answer(\mathcal{D}, q) \to a$. It generates an answer a in response to the query q.
- $Reconstruct(h, a) \to \{\mathcal{D}_i, \perp\}$. It reconstructs the record $\mathcal{D}_i$, given the answer a and the hint h, or rejects the answer and outputs $\perp$.
- $Refresh(ck, h, a) \to (ck, h)$. It updates the hint h, given the answer a.

In the two-server setting ($k = 2$), as discussed in the introduction, we assume a malicious query server and a semi-honest hint server. The hint server will always follow the protocol honestly, while the query server may deviate from the protocol. Prior work [5] adopts a similar setting, but they do not require the client to be aware of which server is semi-honest. The correct database is defined as the one accepted by the *Hint* algorithm. In the single-server setting ($k = 1$), the correct database is identified by a database digest provided by the data owner. According to the trusted digest, the client may reject a false database in the *Hint* algorithm. The scheme should satisfy the following properties:

- **Correctness.** Given a security parameter λ, for any sufficiently large N, any database $\mathcal{D}$ and index i, if both servers and client follow the protocol honestly, the client outputs $\mathcal{D}_i$ with overwhelmingly high probability.

- **Integrity.** Given a security parameter λ, for any sufficiently large N, any database $\mathcal{D}$ and index i, for any probabilistic polynomial time adversary $\mathcal{A}$, the probability that the client reconstructs an incorrect record $\widehat{\mathcal{D}_i} \neq \mathcal{D}_i$ is negligible in λ.
- **Privacy (Concerning Selective Failure Attack).** The servers learn nothing about the index i from the query, even with the knowledge of whether *Reconstruct* algorithm outputs $\perp$.

For a formal definition, please refer to Appendix B.

5.2 Verifiability as a Plugin

We first introduce the construction of our verifiable PIR scheme in a two-server setting. Recall that in numerous protocols for malicious-secure multiparty computation, verifying the outcome of computing a function $m = f(x)$ with potential malicious participants involves separately computing two values, $m = f(x)$ and $n = g(x) = \alpha f(x)$, where $\alpha \in \mathbb{F}_{2^p}$ represents a concealed coefficient. The verification process then entails checking whether $m\alpha = n$. Honest participants invariably succeed in this verification, whereas dishonest behavior has only a negligible probability of passing it. In our protocol, we adopt a similar approach to authenticate the query response a_{query}.

Recall that in the construction from Section 4.2, the client initially retrieves multiple hints from the hint server. Given our assumption of a semi-honest hint server and the presence of a malicious query server, our sole reliable resource is the honestly generated hints. Therefore, we opt to incorporate extra information within each hint to enable subsequent answer verification by the client. Rather than defining each hint as the sum of records within a specific set, each hint now includes two components: the sum parity h^s and the random parity h^r. For a specific set $S = \{x_j \mid j \in [s]\}$, the client first sample a set of random numbers $SR := \{r_j \mid j \in [s]\}$, then h^s, h^r are defined as following:

$$h^s := \sum_{j\in[s]} \mathcal{D}_{x_j},$$
$$h^r := \sum_{j\in[s]} r_j \cdot \mathcal{D}_{x_j}$$

Accordingly, the query process is divided into two distinct parts: a sum query q^s and a random query q^r. Suppose x_i is the queried index. The definitions of q^s and q^r are as follows:

$$q^s := S \setminus \{x_i\},$$
$$q^r := SR \setminus \{r_i\}$$

Upon receiving these two queries, the query server calculates the answers a^s and a^r using a similar summation method:

$$a^s := \sum_{j\in[s-1]} \mathcal{D}_{q^s_j},$$
$$a^r := \sum_{j\in[s-1]} q^r_j \cdot \mathcal{D}_{q^s_j}$$

Subsequently, the query server sends the responses to the client, who then reconstructs the answer $\mathcal{D}_{x_i} := h^s - a^s$ and verifies it by checking whether

$$h^r - a^r = r_i \cdot (h^s - a^s)$$

If the query server responds truthfully, the left-hand side of the equation simplifies to $r_i \cdot \mathcal{D}_{x_i}$. Given that $h^s - a^s = \mathcal{D}_{x_i}$, the equation should hold true. If the verification does not pass, the client outputs $\perp$ to indicate a discrepancy. A more detailed discussion on the integrity of this scheme is deferred to Appendix B.

In our framework, we further employ the optimization strategies discussed in Section 4.2 to improve efficiency. The set S is encoded using a PRF with the key sk. The additional set, SR, is similarly encoded using another PRF with the key sr: it has a different range: $fr_{sr} : [\sqrt{N}] \to \mathbb{F}_{2^p}$. Therefore, the set SR can be expanded using the short key sr:

$$sr \to SR = \{fr_{sr}(j) \mid j \in [\sqrt{N}]\}$$

Next, we delve into concealing the removed item in an online query. The removed item of the random set SR leaks the block that contains the queried index i. Recall that in the semi-honest scheme, a random crumb c conceals the removed index in S. Similarly, in the malicious setting, a random element $r \in \mathbb{F}_{2^p}$ is utilized to hide the removed item in SR. More precisely, let $l := \lfloor i/\sqrt{N} \rfloor$ be the block containing queried index i, the client samples r from $\mathbb{F}_{2^p}$ and replace $SR[l]$ with r. After the client receives the answer a^r, it can remove the crumb value u and update a^r as $a^r := a^r - r \cdot u$. Subsequently, the client carries out the verification by checking if $h^r - a^r = fr_{sr}(l) \cdot \mathcal{D}_i$. If the verification fails, the client outputs $\perp$.

Formally, we present our two-server verifiable PIR scheme in Figure 4, with the verification procedure highlighted in red color. The efficiency of our scheme is comparable to the setup described in Section 4. We propose the following theorem regarding security and efficiency:

Theorem 1. The scheme described in Figure 4 is a two-server offline-online verifiable private information retrieval scheme. On a database of size N, the efficiency of the scheme is as follows:

- Offline communication complexity: $O(\lambda\sqrt{N})$.
- Offline computation complexity: $O(\lambda N)$.
- Online communication complexity: $O(\sqrt{N})$.
- Online computation complexity: $O(\sqrt{N})$.
- Amortized computation complexity: $O(\sqrt{N})$.
- Supports a polynomial number of queries.

Proof sketch. The correctness and integrity of our scheme follow straightforwardly from the construction. The privacy of Crust is derived from the fact that the client's queries are indistinguishable from random sets containing one element in each block. The server learns nothing from the verification result, as it can fully anticipate it based on the answer, thus preventing selective failure attacks. A formal proof can be found in Appendix B.

Efficiency. In the offline phase, the client sends one PRF key to the hint server. The hint server in return sends M parities and $\sqrt{N}$ crumbs to the client. Assuming parities and crumbs are of the same size as one database record and setting $M = \Theta(\lambda\sqrt{N})$, the total offline communication would be $O(\lambda\sqrt{N})$, and the offline computation is $O(\lambda N)$. In the online phase, the client searches for a hint that contains the queried index i, which takes $O(\sqrt{N})$ computation in expectation. The successive processing of the hint, including evaluating the set, replacing the index with crumb, and sampling a new set, also costs $O(\sqrt{N})$ computation. One query with a size of $O(\sqrt{N})$ is then sent to both servers. Each server computes the parities in $O(\sqrt{N})$ time. The hint server answers the query with $\sqrt{N}$ records and the query server answers the query with 2 parities. The client then verifies the answer in $O(1)$ time. Therefore, the total online communication is $O(\sqrt{N})$ and the online computation is $O(\sqrt{N})$. □

5.3 Adapting for Single-server

We now transition to adapting our scheme for a single-server verifiable PIR. In this scenario, we assume a data owner outsourcing a database to a single server and producing a database digest which the client is aware of. To accomplish such a transition, there are two primary challenges:

(1) **Refreshing hints.** To facilitate the client's ability to perform multiple queries, an efficient mechanism for refreshing hints online is necessary. However, the absence of a dedicated hint server complicates the implementation of such a mechanism.
(2) **Retrieving hints.** Recall that in the two-server setting, the hint server is tasked with supplying hints to the client. However, this approach is not feasible in a single-server setting, as the query server, being aware of the hints, could potentially deduce the queried index by comparing the original hint with the query.

Additionally, it is pertinent to consider how the client ensures the correctness of the database managed by an untrusted server. We aim to tackle the above challenges in the following part.

Refresh hints with backup. To enable hint refreshing during the online phase, we adopt the concept of backup hints [6]. In this approach, the client acquires additional hints as "backup" during the offline phase. These backup hints are then utilized to refresh the hints that were consumed in the online phase.

In this method, the hints are classified into two types: main hints and backup hints. The client retrieves the main hints as the hints in the two-server scheme. In addition to that, the client samples λ backup hints for each of the $\sqrt{N}$ blocks. A backup hint for block l is a hint with the l-th offset kept empty. In other words, the backup hint does not contain an index in block l. The client makes queries to the server as it does in the two-server scheme utilizing main hints. Once a query to an index in block l is completed, a backup hint for block l is obtained. The client incorporates the queried index into the backup hint and updates the parities, transforming the backup hint into a main hint. This transformation is achieved by including an additional index in the hint to serve as the extra index. In practice, the client represents a hint set with a tuple (sk, x). The client evaluates a set by first expanding sk into a set S. If $x \neq \perp$, the client replaces the entry $S[\lfloor x/\sqrt{N} \rfloor]$ with x.

The crumbs follow a similar pattern. The client samples λ additional crumbs for each block during the offline phase and acquires a new crumb from the block after issuing a query.

It can be shown that with λ backup hints and crumbs allocated for each block, the scheme enables a client to execute up to $\sqrt{N}$ queries post one resource-intensive offline phase, with an overwhelming probability of success. However, after that, the offline phase must be reactivated to replenish these resources.

Retrieving hints by streaming the database. From the literature, in the single-server setting, addressing the challenge of retrieving hints offline can be approached in two ways: (i) [6] proposed the use of homomorphic encryption for retrieving hints, whereas (ii) [20, 27] recommended streaming the entire database to the client.

We adopt the latter approach since it is more efficient in practice. The hint generation could be done in a block-wise streaming way. The client requests one of the $\sqrt{N}$ blocks from the server at one time and updates all local hints. Specifically, when processing the j-th block, for a hint with parity h_t and PRF key sk_t, the client updates h_t by $h_t := h_t + \mathcal{D}_{f_{sk_t}(j)}$. If the hint does not contain any index in the block, indicating that it is a backup hint, it is skipped. After all hints are updated, the block is replaced with a new block fetched from the server. This ensures that the client's storage remains sublinear to database size.

Introducing verifiability. Leveraging the streaming strategy, integrating verifiability into our scheme is seamless: alongside the additional parity, we require the trusted database owner to provide a digest of the database, similar to the assumption made in [5]. Prior to executing online queries, the client validates the streamed database against this digest, ensuring the authenticity and integrity of the database content.

So far, we have tackled the primary challenges associated with transitioning our two-server verifiable PIR scheme to a single-server context. Before presenting our single-server construction, it is crucial to underscore some advantageous properties that facilitate this construction, which are similarly utilized in [6, 27]:

- **A scheme support $O(\sqrt{N})$ queries implies a scheme support polynomial number of queries.** By replaying the offline phase after conducting approximately $Q := \Theta(\sqrt{N})$ queries and ensuring the scheme concludes its offline setup within $O_\lambda(N)$ time, which is a common characteristic in sublinear PIR schemes, the client can offset the costly offline setup across the Q queries. This iterative offline-to-online cycle facilitates a polynomial number of queries.
- **Uniformly random query distribution.** We assume there's no duplication in Q queries and the queried indexes are uniformly random in the database, as aligned to prior works [6, 27].

Put everything together, we present our single-server verifiable PIR scheme in Figure 5, where the verification procedure is distinctly marked in red color. Formally, we have the following theorem:

Theorem 2. The scheme described in Figure 5 is a single-server offline-online verifiable private information retrieval scheme. On a database of size N, the efficiency of the scheme is as follows:

- Offline communication complexity: $O(N)$.
- Offline computation complexity: $O(\lambda N)$.
- Online communication complexity: $O(\sqrt{N})$.
- Online computation complexity: $O(\sqrt{N})$.
- Amortized computation complexity: $O(\lambda\sqrt{N})$.
- Supports a polynomial number of queries.

Proof sketch. The security proof is similar to that of the two-server scheme. The intuition is that the view of a malicious query server is the same in both the two-server and single-server schemes. We refer readers to Appendix B for a formal proof.

Efficiency. Streaming the database to the client in the offline phase requires $O(N)$ communication and $O(\lambda N)$ computation. This is in line with the state-of-the-art single-server sublinear PIR scheme [27]. In the online phase, the cost is similar to that of the two-server scheme. However, unlike the two-server scheme, the offline phase must be rerun after Q queries. By setting $Q = \Theta(\sqrt{N})$, we achieve an amortized communication cost of $O(\sqrt{N})$ and an amortized computation cost of $O(\lambda\sqrt{N})$. □

Single-server Scheme of Crust

Notation: The scheme involves a client C and a query server $\mathcal{S}_{\text{query}}$. Assume a hint consists of a tuple of elements: $h = (sk, x, h^s, sr, h^r)$. Assume a crumb consists of an offset and a value $c = (u, o)$. *Expand* a key sk involves calculating the set $\{f_{sk}(j) + j \cdot \sqrt{N} \mid j \in [\sqrt{N}]\}$. *Expand* a key sr involves calculating the set $\{fr_{sr}(j) \mid j \in [\sqrt{N}]\}$. Suppose i is the queried index. A trusted database owner O calculates a digest d of the database $\mathcal{D}$ and makes it accessible to C.

Offline Phase:

- **Setup:** C samples main PRF keys $sk_j, j \in [M]$, backup keys $\overline{sk}_{k,j}, j \in [\lambda], k \in [\sqrt{N}]$ and $sr_j, j \in [M]$ with corresponding backup keys $\overline{sr}_{k,j}, j \in [\lambda], k \in [\sqrt{N}]$, initializes the main hints to $h_j := (sk_j, \perp, 0, sr_j, 0), j \in [M]$, backup hints to $h_{k,j}(\overline{sk}_{k,j}, \perp, 0, \overline{sr}_{k,j}, 0), j \in [\lambda], k \in [\sqrt{N}]$ and crumb storage to $c_{k,j} := (\perp, \perp), j \in [\lambda], k \in [\sqrt{N}]$.
- **Hint:** The server streams the database to C. C initializes a digest d'. While streaming the l-th block, C updates the hints:
  - Update main hints: $h^s_j := h^s_j + \mathcal{D}_{f_{sk_j}(l)}$ and $h^r_j := h^r_j + f_{sr_j}(l) \cdot \mathcal{D}_{f_{sk_j}(l)}$ for $j \in [M]$.
  - Update backup hints not in the block: $\overline{h^s}_{k,j}$ and $\overline{h^r}_{k,j}$ by $\overline{h^s}_{k,j} := \overline{h^s}_{k,j} + \mathcal{D}_{f_{\overline{sk}_{k,j}}(l)}$ and $\overline{h^r}_{k,j} := \overline{h^r}_{k,j} + f_{\overline{sr}_{k,j}}(l) \cdot \mathcal{D}_{fr_{\overline{sk}_{k,j}}(l)}$ for $j \in [\lambda], k \in [\sqrt{N}], k \neq l$.
  - Update crumbs: $c_{l,j}$ to be randomly selected offset and corresponding record $(u_{l,j}, o_{l,j}), j \in [\lambda]$ in the block.
  - C updates d' with this database block.
  - After that, C deletes the block from storage.
  - C checks if $d = d'$. If the equation does not hold, C aborts and outputs $\perp$.

Online Phase:

- **Query:**
  - Denote the block that i is in as $l := \lfloor i/\sqrt{N} \rfloor$. C finds a hint $h_t = (sk_t, x_t, h^s_t, sr_t, h^r_t)$ that contains i (either $x_t = i$ or $f_{sk_t}(l) + \sqrt{N} \cdot l = i$ and $x_t = \perp \vee \lfloor x_t/\sqrt{N} \rfloor \neq l$). If no such hint exists, C samples a new hint $(sk_t, x_t := \perp, h^s_t := 0, sr_t, h^r_t := 0)$
  - C *Expands* sk_t to a set S. $S[\lfloor x_t/\sqrt{N} \rfloor]$ is replaced with x_t if $x_t \neq \perp$. It finds an unused crumb $c_{l,e} = (u_{l,e}, o_{l,e})$ in the block l and replaces $S[l]$ with $o_{l,e}$. C *Expand* sr_t to set SR, $SR[l]$ is replaced by a random element r. C sends S and SR to $\mathcal{S}_{\text{query}}$.
- **Answer:** $\mathcal{S}_{\text{query}}$ calculates the parities $a^s := \sum_{k \in [\sqrt{N}]} \mathcal{D}_{S[k]}$ and $a^r := \sum_{k \in [\sqrt{N}]} \mathcal{D}_{S[k]} \cdot SR[k]$. It sends a^s and a^r to C.
- **Reconstruct:** C reconstructs the answer $\mathcal{D}_i := h^s_t - (a^s - u_{l,e})$. It then verifies the answer by checking if $h^r_q - (a^r - r \cdot u_{l,e}) = fr_{sr_t}(l) \cdot \mathcal{D}_i$. If the verification fails, C outputs $\perp$. Otherwise, C outputs $\mathcal{D}_i$.
- **Refresh:**
  - C finds an unused backup hint $h_{l,j}$ in the block l, parses $h_{l,j}$ as $(\overline{sk}, \perp, \overline{h^s}, \overline{sr}, \overline{h^r})$. If no such hint exists, C samples a new hint $(\overline{sk}, x := \perp, \overline{h^s} := 0, \overline{sr}, \overline{h^r} := 0)$
  - C updates $h_t := (\overline{sk}, i, \overline{h^s} + \mathcal{D}_i, \overline{sr}, \overline{h^r} + f_{\overline{sr}}(l) \cdot \mathcal{D}_i)$.

Figure 5: The single-server scheme of Crust. The red part is the verification procedure and is utilized as a plugin, and this part can be removed to achieve an even more efficient PIR scheme in a semi-honest setting.

A practical online-offline paradigm. The concept of refreshing backup hints can be applied in a two-server scheme. Concretely, the client retrieves backup hints and crumbs from the hint server as it does in the single-server scheme. However, rather than streaming the database, the client sends PRF keys to the hint server, similar to the approach used in the two-server scheme. This allows the client to circumvent online interactions with a semi-honest server while minimizing offline communication. This notion offers a practical framework for service providers, enabling a database owner to offer hints using cost-effective idle resources and delegate real-time query processing to potentially untrustworthy servers.

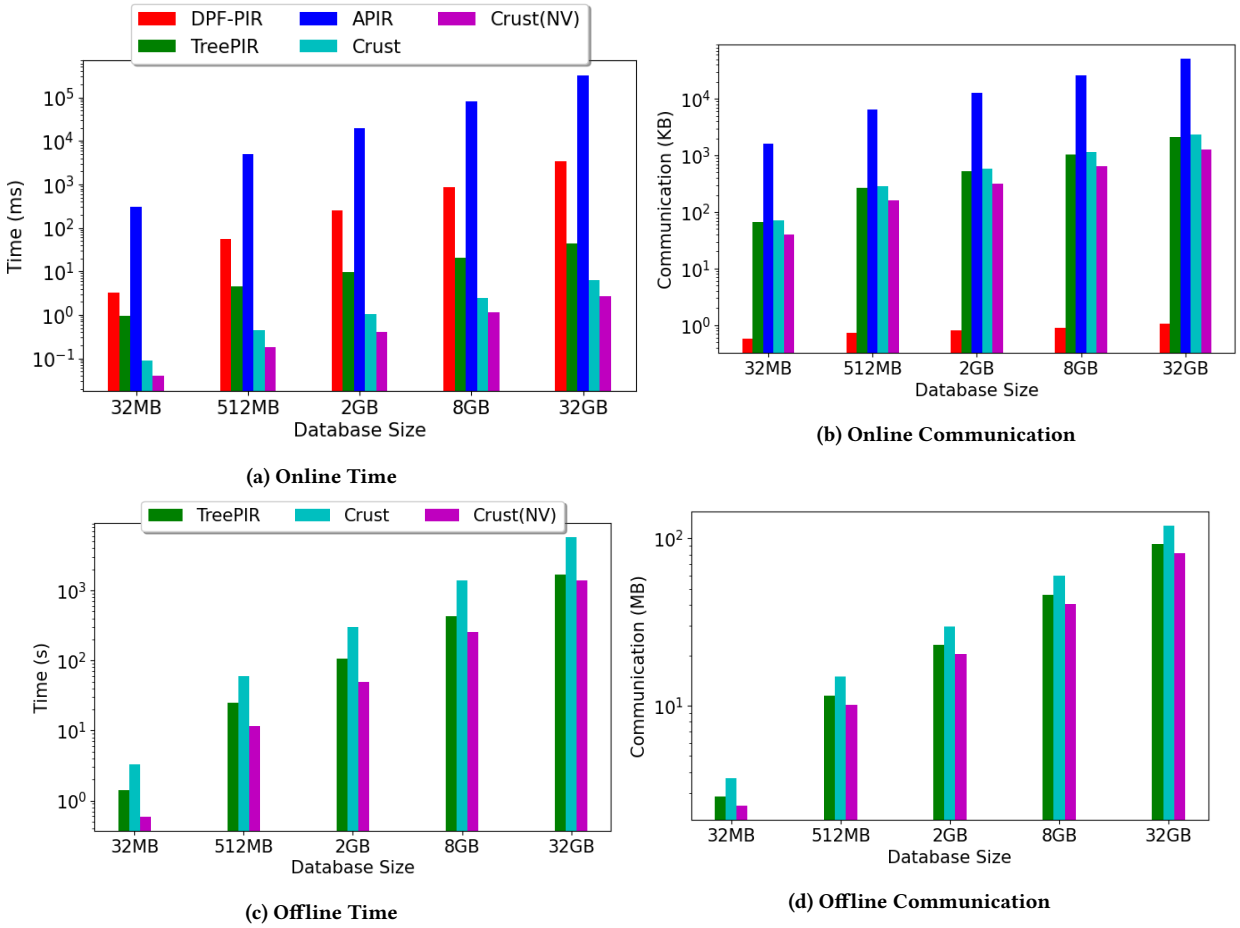
6 EVALUATION

6.1 Experiment Setup

We have implemented both of the two schemes for Crust. Our codebase is publicly accessible[1]. As for our construction, we carefully chose parameters to guarantee a minimum security level of 128 bits. Given that the verification procedure in Crust is designed to be pluggable, we evaluated two distinct versions of Crust: one with the verification feature activated and another without it. In the presentation of our evaluation results, "Crust(NV)" denotes the version of Crust where the verification procedure is not included.

We conducted a comprehensive evaluation of our construction's performance and benchmarked it against leading two-server PIR, single-server PIR, and verifiable PIR schemes. For the two-server PIR comparison, we selected DPF-PIR [11], TreePIR [16], and APIR

[1]The code is available at https://anonymous.4open.science/r/crust-D745.



(a) Online Time

(b) Online Communication

(c) Offline Time

(d) Offline Communication

Figure 6: Comparison of two-server PIR schemes. The DPF-PIR and APIR schemes do not employ an offline-online model, and only the computation time and communication overhead in the online phase are evaluated. The data of APIR on databases larger than 2GB is estimated because it consumes more than 128 GB RAM.

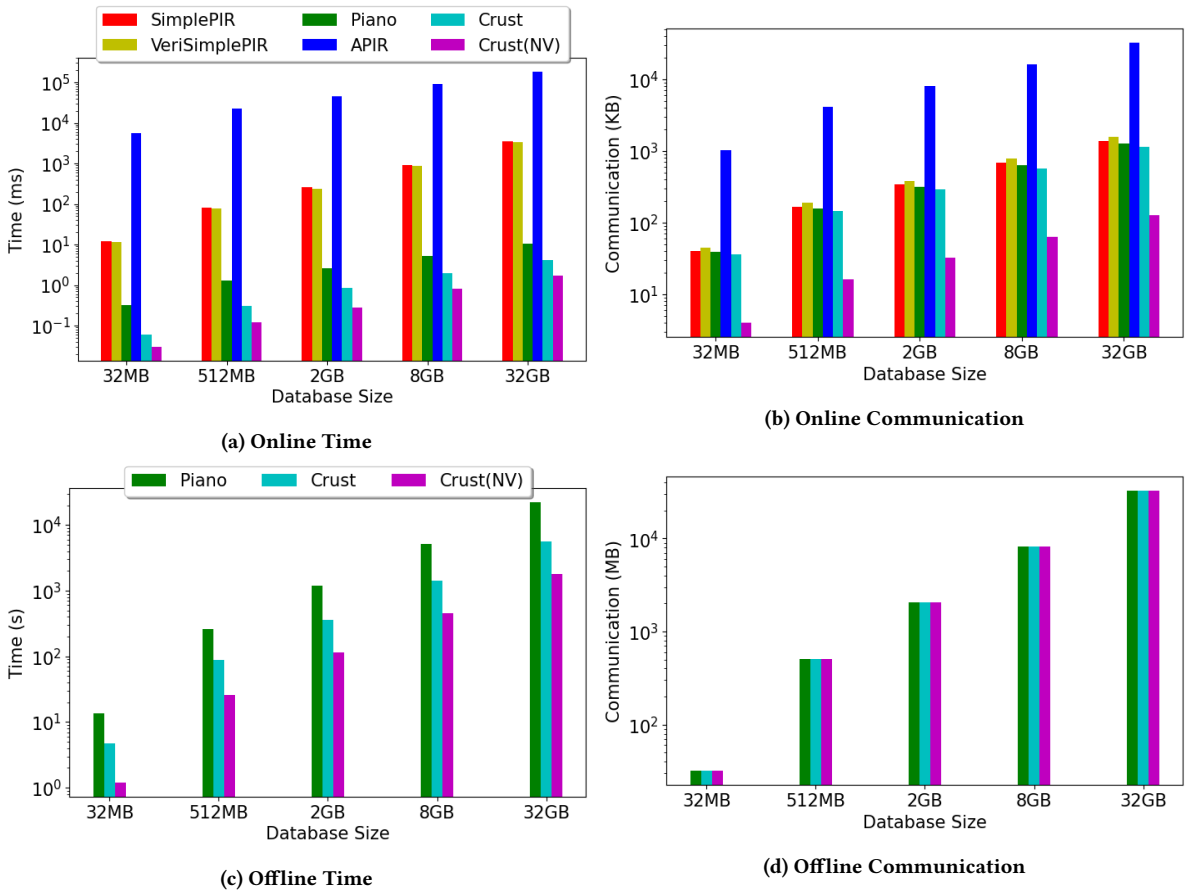
[5] as our baselines. In the context of single-server settings, our benchmarks include SimplePIR [13], VeriSimplePIR [9], Piano [27], and the learning-with-errors (LWE) variant of APIR. Among these, VeriSimplePIR and APIR are recognized as verifiable PIR schemes, indicating they are designed to ensure the integrity of responses in the presence of potentially malicious servers. The rest are predicated on the assumption of a semi-honest server model. All benchmarks were conducted on a single thread. For the two-server PIR schemes, the operations of the two servers were executed serially.

The benchmark tests were performed on a server equipped with a 32-core AMD EPYC 7K83 CPU and 128GB of RAM. These tests were conducted on five databases, with sizes ranging from 32MB to 32GB, each consisting of 32-byte records but differing in overall database size.

6.2 Two-server Schemes

The benchmark results for the two-server PIR schemes are depicted in Figure 6. Among these schemes, our schemes (both Crust and Crust(NV)) surpass the leading sublinear scheme, TreePIR [16], in both computation and communication efficiency. This superior performance primarily stems from our schemes' simplified query construction and the elimination of dummy queries. The removal of costly PPRF and dummy queries in the online phase significantly enhances performance, yielding an improvement of more than 16 times. Specifically, for a query on a 32 GB database, Crust(NV) requires only 2.72 milliseconds, while TreePIR takes 43.2 milliseconds. Even the "verifiable" version of Crust outperforms TreePIR in terms of online computation speed, running one query in 6.16 milliseconds. Nonetheless, it does entail marginally higher communication overhead. Regarding the offline phase, Crust(NV) incurs a lower cost compared to TreePIR. This reduction is primarily due to the simplified query construction.

DPF-PIR [11] is a linear PIR scheme, implying that its online computation time scales linearly with the size of the database. For a 32 GB database, DPF-PIR completes a single query in 3372 milliseconds. In contrast, our scheme requires less than 10 milliseconds to execute the same operation. This significant difference in performance underscores the critical need for a sublinear PIR scheme,



(a) Online Time

(b) Online Communication

(c) Offline Time

(d) Offline Communication

Figure 7: Comparison of single-server PIR schemes. SimplePIR, APIR, and VeriSimplePIR do not employ an offline-online model, and only the computation time and communication overhead in the online phase are evaluated.

especially when dealing with large databases, as it dramatically enhances query efficiency and reduces computation time.

Comparison with verifiable PIR. It can be concluded that Crust introduces only minimal overhead to achieve verifiability. APIR [5] provides a leading two-server verifiable PIR scheme. From the figure, our scheme surpasses APIR in computational efficiency across all database sizes. Furthermore, the Merkle proof-based APIR results in substantial database expansion, which hampers the ability to execute the scheme on databases exceeding 2^{28} records due to RAM limitations.

6.3 Single-server Schemes

Next, we present the benchmark results among single-server PIR schemes, as depicted in Figure 7. For this comparison, we select Piano [27], a state-of-the-art single-server sublinear scheme, as our benchmark counterpart. Piano necessitates the server to calculate $\sqrt{N}$ answers to conceal the queried index, thereby imposing additional computation and communication costs. Consequently, Crust outperforms Piano in both computation and communication, with 6 times faster online computation and 10 times smaller online communication.

In the offline phase, both Piano and Crust utilize a strategy that involves streaming the database to the client, resulting in similar communication costs. However, the parameter selection for Piano results in a higher offline computation burden compared to both versions of Crust. Specifically, Crust(NV) requires 90% less time than Piano for offline computation, while the verifiable version of Crust takes 75% less time than Piano.

Comparison with verifiable PIR. In comparison to the leading single-server verifiable PIR scheme VeriSimplePIR [9], Crust involves a 30% reduction in online communication costs and approximately 1000 times less online computation costs. When conducting a query on a 32 GB database, Crust only demands 4.23 milliseconds, showcasing the effectiveness of sublinear PIR schemes.

7 CONCLUSION

In this paper, we introduce Crust, a novel and efficient verifiable PIR scheme applicable in both two-server and single-server settings, where the verification procedure is pluggable. The efficiency of Crust surpasses that of existing verifiable PIR schemes and sublinear PIR schemes. Our scheme is poised to push the boundaries

of PIR and its applications, paving the way for enhanced privacy protections and data integrity assurances.

REFERENCES

[1] Amos Beimel and Yoav Stahl. 2003. Robust Information-Theoretic Private Information Retrieval. In *SCN 02 (LNCS, Vol. 2576)*, Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano (Eds.). Springer, Heidelberg, 326–341. https://doi.org/10.1007/3-540-36413-7_24

[2] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function Secret Sharing. In *EUROCRYPT 2015, Part II (LNCS, Vol. 9057)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, Heidelberg, 337–367. https://doi.org/10.1007/978-3-662-46803-6_12

[3] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2016. Function Secret Sharing: Improvements and Extensions. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 1292–1303. https://doi.org/10.1145/2976749.2978429

[4] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. 1995. Private Information Retrieval. In *36th FOCS*. IEEE Computer Society Press, 41–50. https://doi.org/10.1109/SFCS.1995.492461

[5] Simone Colombo, Kirill Nikitin, Henry Corrigan-Gibbs, David J. Wu, and Bryan Ford. 2023. Authenticated private information retrieval. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 3835–3851. https://www.usenix.org/conference/usenixsecurity23/presentation/colombo

[6] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. 2022. Single-Server Private Information Retrieval with Sublinear Amortized Time. In *EUROCRYPT 2022, Part II (LNCS, Vol. 13276)*, Orr Dunkelman and Stefan Dziembowski (Eds.). Springer, Heidelberg, 3–33. https://doi.org/10.1007/978-3-031-07085-3_1

[7] Henry Corrigan-Gibbs and Dmitry Kogan. 2020. Private Information Retrieval with Sublinear Online Time. In *EUROCRYPT 2020, Part I (LNCS, Vol. 12105)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, Heidelberg, 44–75. https://doi.org/10.1007/978-3-030-45721-1_3

[8] David Dagon, Manos Antonakakis, Kevin Day, Xiapu Luo, Christopher P Lee, and Wenke Lee. 2009. Recursive DNS Architectures and Vulnerability Implications.. In *NDSS*.

[9] Leo de Castro and Keewoo Lee. 2024. VeriSimplePIR: Verifiability in SimplePIR at No Online Cost for Honest Servers. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA. https://www.usenix.org/conference/usenixsecurity24/presentation/de-castro

[10] Casey Devet, Ian Goldberg, and Nadia Heninger. 2012. Optimally Robust Private Information Retrieval. In *USENIX Security 2012*, Tadayoshi Kohno (Ed.). USENIX Association, 269–283.

[11] Niv Gilboa and Yuval Ishai. 2014. Distributed Point Functions and Their Applications. In *EUROCRYPT 2014 (LNCS, Vol. 8441)*, Phong Q. Nguyen and Elisabeth Oswald (Eds.). Springer, Heidelberg, 640–658. https://doi.org/10.1007/978-3-642-55220-5_35

[12] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi, and Michael Walfish. 2016. Scalable and Private Media Consumption with Popcorn. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA, 91–107. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/gupta-trinabh

[13] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2023. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 3889–3905. https://www.usenix.org/conference/usenixsecurity23/presentation/henzinger

[14] Mehmet Sabir Kiraz and Berry Schoenmakers. 2006. A protocol issue for the malicious case of Yao's garbled circuit construction. https://api.semanticscholar.org/CorpusID:9024240

[15] Dmitry Kogan and Henry Corrigan-Gibbs. 2021. Private Blocklist Lookups with Checklist. In *USENIX Security 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 875–892.

[16] Arthur Lazzaretti and Charalampos Papamanthou. 2023. TreePIR: Sublinear-Time and Polylog-Bandwidth Private Information Retrieval from DDH. In *CRYPTO 2023, Part II (LNCS, Vol. 14082)*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer, Heidelberg, 284–314. https://doi.org/10.1007/978-3-031-38545-2_10

[17] Rasoul Akhavan Mahdavi and Florian Kerschbaum. 2022. Constant-weight PIR: Single-round Keyword PIR via Constant-weight Equality Operators. In *USENIX Security 2022*, Kevin R. B. Butler and Kurt Thomas (Eds.). USENIX Association, 1723–1740.

[18] Samir Jordan Menon and David J. Wu. 2022. SPIRAL: Fast, High-Rate Single-Server PIR via FHE Composition. In *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 930–947. https://doi.org/10.1109/SP46214.2022.9833700

[19] Muhammad Haris Mughees, Sun I, and Ling Ren. 2023. Simple and Practical Amortized Sublinear Private Information Retrieval. Cryptology ePrint Archive, Paper 2023/1072. https://eprint.iacr.org/2023/1072

[20] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. 2018. Private Stateful Information Retrieval. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 1002–1019. https://doi.org/10.1145/3243734.3243821

[21] Elaine Shi, Waqar Aqeel, Balakrishnan Chandrasekaran, and Bruce M. Maggs. 2021. Puncturable Pseudorandom Sets and Private Information Retrieval with Near-Optimal Online Bandwidth and Time. In *CRYPTO 2021, Part IV (LNCS, Vol. 12828)*, Tal Malkin and Chris Peikert (Eds.). Springer, Heidelberg, Virtual Event, 641–669. https://doi.org/10.1007/978-3-030-84259-8_22

[22] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. 2017. Splinter: Practical Private Queries on Public Data. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 299–313. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/wang-frank

[23] Ke Coby Wang and Michael K. Reiter. 2020. Detecting Stuffing of a User's Credentials at Her Own Accounts. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2201–2218. https://www.usenix.org/conference/usenixsecurity20/presentation/wang

[24] Liang Feng Zhang and Reihaneh Safavi-Naini. 2014. Verifiable Multi-server Private Information Retrieval. In *ACNS 14 (LNCS, Vol. 8479)*, Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay (Eds.). Springer, Heidelberg, 62–79. https://doi.org/10.1007/978-3-319-07536-5_5

[25] Fangming Zhao, Yoshiaki Hori, and Kouichi Sakurai. 2007. Two-servers PIR based DNS query scheme with privacy-preserving. 299–302. https://doi.org/10.1109/IPC.2007.27

[26] Liang Zhao, Xingfeng Wang, and Xinyi Huang. 2018. Verifiable single-server private information retrieval from LWE with binary errors. 546 (2018), 897–923. https://doi.org/10.1016/j.ins.2020.08.071

[27] Mingxun Zhou, Andrew Park, Elaine Shi, and Wenting Zheng. 2023. Piano: Extremely Simple, Single-Server PIR with Sublinear Server Computation. Cryptology ePrint Archive, Paper 2023/452. https://eprint.iacr.org/2023/452

A HANDLING A MALICIOUS HINT SERVER IN THE TWO-SERVER SETTING

A sublinear PIR scheme derived from the framework in [7] encounters challenges in maintaining integrity when the hint server behaves maliciously. Essentially, the roles of $\mathcal{S}_{\text{hint}}$ and $\mathcal{S}_{\text{query}}$ differ in that the information acquired by client C from $\mathcal{S}_{\text{query}}$ undergoes a cross-validation process with the knowledge from S_{hint}, but not the other way around. Specifically, there are two crucial points:

(1) The offline hint-retrieving phase only engages the hint server and the client. The query server is not involved at all.
(2) A part of the hint, namely the removed items, is not sent to the query server. These values are not cross-validated.

We demonstrate a possible strategy for the hint server to cheat the client. We assume there's a predefined genuine database $\mathcal{D}$ that is sent to the two servers. So the concept of correctness is clear.

In the online phase, suppose the client has chosen a hint $h_t = (sk_t, h_t^s, sr_t, h_t^r)$ to query $\mathcal{D}_i$, in which case $i \in Expand(sk_t)$. The client retrieves the hint from a malicious hint server. If h_t^s, h_t^r are generated with a false $\widehat{\mathcal{D}_i} \neq \mathcal{D}_i$, the client would reconstruct $\widehat{\mathcal{D}_i}$.

The verification we introduced is not involved. Notice that i will be removed in the query, so the only chance that the client can detect the cheating is when the client happens to use a crumb with index i in the query. The probability is $\frac{1}{\sqrt{N}}$.

From a similar argument, the scheme is prone to selective failure attacks when the hint server is malicious. Although the queries to the hint server are independent from the queried index. The mere fact that the client fails a query is informative enough for the hint server. That means the client just used a "poisoned" hint and that narrows down the queried index to possibly one hint set.

In such concerns, the scheme only tolerates one semi-honest hint server and a malicious query server. It will be valuable future work

to design a sublinear PIR scheme that can tolerate two malicious servers.

B SECURITY PROOF

We will focus on the two-server protocol. As is introduced in the brief analysis part, the single-server protocol can be easily reduced to the two-server protocol, with minimal modification to the proof.

B.1 Extension to Multiple Queries

To show the scheme can handle polynomially many queries, it is sufficient to show the distribution of hints remains the same after each query.

Lemma 1. At the end of the offline phase, and after each query, the client-side hints h follow the same distribution, which is indistinguishable from M random set with one random element in each block.

Proof. At the end of the offline phase, the argument follows directly from the scheme and the security of PRF. In the online phase, it is sufficient to show that during each query, the consumed hint and the refreshed hint follow the same distribution. Recall that in a query to i which falls into the l-th block, the found query set sk_t and the newly sampled set $\overline{sk}$ are both uniformly random in every other block than l and contains i in block l, which directly gives us the lemma. □

From lemma 1, we are confident to say the proof for the first query can be extended to the subsequent queries. In the rest of the proof, unless otherwise specified, we assume the query in the proof is the first one in the online phase.

B.2 Correctness

We first provide a formal definition of correctness.

Definition B.1 (Correctness for two-server scheme)**.** Given a security parameter λ for any sufficiently large N, any database $\mathcal{D}$ and index i, if both servers and client follow the protocol in Figure 4 honestly, the client outputs $\mathcal{D}_i$ with probability $PC_i \geq 1 - \mathsf{negl}(\lambda)$.

There's only one scenario in which the client cannot get a correct correctness, namely in the *Query* algorithm the client cannot find a hint containing the queried index. We show that such probability is negligible by the following lemma:

Lemma 2. Each hint contains any specific index i with probability $\frac{1}{\sqrt{N}}$.

Proof. Each hint contains exactly one random element in each block of size $\sqrt{N}$. Any specific index falls into one block and is selected with probability $\frac{1}{\sqrt{N}}$. □

From Lemma 2, the probability is at most $(1 - \frac{1}{\sqrt{N}})^M$. With $M = \Theta(\lambda\sqrt{N})$, $(1 - \frac{1}{\sqrt{N}})^{\lambda\sqrt{N}} < e^{-\lambda}$.

Definition B.2 (Correctness for single-server scheme)**.** Given a security parameter λ for any sufficiently large N, any database $\mathcal{D}$ and index i, if server and client follow the protocol in Figure 5 honestly, the client outputs $\mathcal{D}_i$ with probability $PC_i \geq 1 - \mathsf{negl}(\lambda)$.

For the single-server scheme, an extra risk is that a block runs out of crumbs or backup hints if more than λ queries fall into it. With $M = \Theta(\lambda\sqrt{N})$, we prove the following lemma

Lemma 3. The probability that more than λ queries fall into one block is negligible.

Proof. The proof can be found in **Claim A.1.** in [6]. □

The rest of single-server correctness follows the same argument as the two-server scheme.

B.3 Integrity

We first provide a formal definition of integrity.

Definition B.3 (Two-server Integrity)**.** Given a security parameter λ, for all probabilistic polynomial time adversary $\mathcal{A}_{\text{query}}$ corrupting the query server, define the following probability as $PI_i(\mathcal{A}_{\text{query}})$.

$$Pr\left[\begin{array}{rcl} (ck, q_h) & \leftarrow & Setup(1^\lambda, N) \\ h & \leftarrow & Hint(\mathcal{D}, q_h) \\ c \notin \{\mathcal{D}_i, \perp\} : \quad (q_{\text{hint}}, q_{\text{query}}) & \leftarrow & Query(ck, i) \\ a'_{\text{query}} & \leftarrow & \mathcal{A}_{\text{query}}(\mathcal{D}, q_{\text{query}}) \\ c & \leftarrow & Reconstruct(h, a'_{\text{query}}) \end{array}\right]$$

On all choices of i, $PI_i(\mathcal{A}_{\text{query}}) \leq \mathsf{negl}(\lambda)$.

The general idea of proving integrity has been shown in the security analysis part. We will give formal proof here. We first prove a useful lemma.

Lemma 4. For any non-zero offset $\Delta = (\Delta_s, \Delta_r) \in \mathbb{F}_{2^p}^2, \Delta \neq (0, 0)$, the following probability gives the chance that the client accepts a false answer $a' = a + \Delta$ from the query server. Denote the probability as $P_{\Delta,i}$.

$$Pr\left[\begin{array}{rcl} (ck, q_h) & \leftarrow & Setup(1^\lambda, N) \\ h & \leftarrow & Hint(\mathcal{D}, q_h) \\ c \neq \perp : \quad (q_{\text{hint}}, q_{\text{query}}) & \leftarrow & Query(ck, i) \\ a_{\text{query}} & \leftarrow & AnswerQuery(\mathcal{D}, q_{\text{query}}) \\ c & \leftarrow & Reconstruct(h, a_{\text{query}} + \Delta) \end{array}\right]$$

On all choices of Δ, i, $P_{\Delta,i} \leq \frac{1}{2^p - 1}$.

Proof. Let r be the random element in $\mathbb{F}_{2^p}$ that is removed from the random hint. In short, the reconstruction part can be rewritten as

$$Pr\left[r \cdot (h^s - a^s - \Delta_s) = h^r - a^r - \Delta_r\right]$$

Within which a_s, a_r are the answers, h_s, h_r are the hint parities. Since the true answers always pass the verification, we have

$$r \cdot (h^s - a^s) = h^r - a^r$$

So the probability can be further simplified as

$$Pr\left[r\Delta_s - \Delta_r = 0\right]$$

Note that r is a random element from PPRS and is indistinguishable from a hidden random value in $\mathbb{F}_{2^p}$. The equation is the evaluation of a non-zero degree-1 polynomial at a random point r. Since a degree-1 polynomial has at most 1 root in $\mathbb{F}_{2^p}$, the probability is at most $\frac{1}{2^p - 1}$. □

With lemma 4 at hand, integrity follows directly.

Remark 1 (Database will small entries)**.** On a database with small entries $2^p < 2^\lambda$, having a random parity equal in size to one database record is insecure. In practice, database entries can be aggregated into larger sizes to ensure security.

The definition of the single-server integrity is similar.

Definition B.4 (Single-server Integrity)**.** Given a security parameter λ, for all probabilistic polynomial time adversary $\mathcal{A}$ corrupting the server, define the following probability as $PI_i(\mathcal{A})$.

$$Pr\left[e \notin \{\mathcal{D}_i, \perp\} : \begin{array}{rcl} d & \leftarrow & Digest(1^\lambda, N, \mathcal{D}) \\ (ck, q_h) & \leftarrow & Setup(1^\lambda, N) \\ \{h, \perp\} & \leftarrow & Hint(\mathcal{A}(\mathcal{D}), q_h, d) \\ q & \leftarrow & Query(ck, i) \\ a' & \leftarrow & \mathcal{A}(\mathcal{D}, q) \\ e & \leftarrow & Reconstruct(h, a') \end{array}\right]$$

On all choices of i, $PI_i(\mathcal{A}) \leq \mathsf{negl}(\lambda)$.

The single-server version shares the same proof, with the only difference being we need to introduce an extra part for the offline setup to ensure the hints are correct. The proof can be done via a basic argument on cryptographic hash functions and we omit it here.

B.4 Privacy

There are two steps in the privacy proof. We first prove that the scheme satisfies privacy, not putting selective failure into consideration. This corresponds to the non-verifiable version of our scheme. Then we show that the scheme satisfies selective failure privacy.

Definition B.5 (Privacy)**.** Given a security parameter λ, the scheme is private with respect to $\mathcal{S}_{\text{query}}$ if there exists a probabilistic polynomial time simulator $Sim(1^\lambda, N)$ such that for an algorithm $serv_{\text{hint}}$ following the protocol of $\mathcal{S}_{\text{hint}}$ honestly, any probabilistic polynomial time adversaries $\mathcal{A}$ corrupting the query server $\mathcal{S}_{\text{query}}$, $\mathcal{A}$ cannot distinguish the view in the following worlds with probability non-negligible in λ.

- **World 0**: C interact with $\mathcal{A}$ who plays the role of $\mathcal{S}_{\text{query}}$ and $serv_{\text{hint}}$ who plays the roll of $\mathcal{S}_{\text{hint}}$. At each step t in the online phase, $\mathcal{A}$ adaptively chooses the next index i_t, and C uses i_t as its query.
- **World 1**: Sim interact with $\mathcal{A}$ who plays the role of $\mathcal{S}_{\text{query}}$ and $serv_{\text{hint}}$ who plays the roll of $\mathcal{S}_{\text{hint}}$. At each step t in the online phase, $\mathcal{A}$ adaptively chooses the next index i_t, and Sim runs without the knowledge of i_t.

$\mathcal{A}$ is allowed to deviate from the protocol arbitrarily. The privacy with respect to $\mathcal{S}_{\text{hint}}$ is defined symmetrically.

Intuitively, We are proving that the client's queries are indistinguishable from queries to random indexes.

We prove the following lemma.

Lemma 5. For any query q received by one of the two servers and any two queried indexes i, i',

$$Pr[q|i] = Pr[q|i']$$

Proof. q can be divided into an index set q^s and a random number set q^r. From the security of PRF, q^s is uniformly random in every block except the block that contains i or i'. When constructing q, i or i' is replaced by a random item in the block (a crumb). Therefore, q^s is indistinguishable from a random set with one element in each $\sqrt{N}$ sized block.

q^r is merely $\sqrt{N}$ random elements in $\mathbb{F}_{2^p}$, its content is irrelevant to the queried index i or i'. The lemma follows. □

With Lemma 5 at hand, we can prove the privacy of the scheme. We first prove the privacy with respect to $\mathcal{S}_{\text{query}}$. In the offline phase, $\mathcal{S}_{\text{query}}$ does not interact with the C. So we only need to consider the online phase. Consider the following hybrid:

- **Hyb 0**: C interact with $\mathcal{A}$ who plays the role of $\mathcal{S}_{\text{query}}$ and $serv_{\text{hint}}$ who plays the roll of $\mathcal{S}_{\text{hint}}$. At each step t in the online phase, $\mathcal{A}$ adaptively chooses the next index i_t. C drops i_t and samples a random index i'_t as its query.

From Lemma 5, **Hyb 0** is indistinguishable from **World 0**. The simulator Sim can be constructed as exactly what C does in **Hyb 0**. **Hyb 0** is indistinguishable from **World 1** because C never makes use of the knowledge of i_t. Therefore, the scheme satisfies privacy with respect to $\mathcal{S}_{\text{query}}$.

Privacy with respect to $\mathcal{S}_{\text{query}}$ in the offline phase follows simply from the fact that the offline phase happens before any query is made. The proof for the online phase is symmetric to the argument of privacy with respect to $\mathcal{S}_{\text{query}}$. This completes the proof of privacy in the semi-honest model.

Then we move on to the malicious model, where selective failure is introduced. Privacy is defined differently.

Definition B.6 (Privacy Against Selective Failure)**.** Given a security parameter λ, the scheme is private against selective failure with respect to $\mathcal{S}_{\text{query}}$ if there exists a probabilistic polynomial time simulator $Sim(1^\lambda, N)$ such that for an algorithm $serv_{\text{hint}}$ follow the protocol of $\mathcal{S}_{\text{hint}}$ honestly, any probabilistic polynomial time adversaries $\mathcal{A}$ corrupting the query server $\mathcal{S}_{\text{query}}$, $\mathcal{A}$ cannot distinguish the view in the following world with probability non-negligible in λ.

- **World 0**: C interact with $\mathcal{A}$ who plays the role of $\mathcal{S}_{\text{query}}$ and $serv_{\text{hint}}$ who plays the roll of $\mathcal{S}_{\text{hint}}$. At each step t in the online phase, $\mathcal{A}$ adaptively chooses the next index i_t, and C uses i_t as its query. The client outputs 1 to $\mathcal{A}$ after each query if the answer is accepted, otherwise 0.
- **World 1**: Sim interact with $\mathcal{A}$ who plays the role of $\mathcal{S}_{\text{query}}$ and $serv_{\text{hint}}$ who plays the roll of $\mathcal{S}_{\text{hint}}$. At each step t in the online phase, $\mathcal{A}$ adaptively chooses the next index i_t, and Sim runs without the knowledge of i. Sim output a bit b to $\mathcal{A}$ after each query.

$\mathcal{A}$ is allowed to deviate from the protocol arbitrarily.

In the offline phase, $\mathcal{S}_{\text{query}}$ does not interact with the C. So we only need to consider the online phase. Consider the following hybrid:

- **Hyb 0**: C interact with $\mathcal{A}$ who plays the role of $\mathcal{S}_{\text{query}}$ and $serv_{\text{hint}}$ who plays the roll of $\mathcal{S}_{\text{hint}}$. At each step t in the online phase, $\mathcal{A}$ adaptively chooses the next index i_t. C drops i_t and

samples a random index i'_t as its query. C outputs 1 to $\mathcal{A}$ after each query if the answer is accepted, otherwise 0.

Hyb 0 is indistinguishable from **World 0**. A query to i_t is indistinguishable from a query to i'_t, thus, we only need to prove that the extra bit is indistinguishable in the two worlds. We argue the bit has nothing to do concerning the input of C, but solely depends on $\mathcal{A}$. $\mathcal{A}$ could answer the query in two ways:

- $\mathcal{A}$ gives the correct answer a. From correctness, C output 1.
- $\mathcal{A}$ gives a false answer a'. From integrity, C output 0 with overwhelming probability.

We see that the input of C does not affect if the bit is 1 or 0. Therefore, the view of $\mathcal{A}$ in **Hyb 0** and **World 0** is indistinguishable. The simulator Sim can be constructed as exactly what the C do in **Hyb 0**. **Hyb 0** is indistinguishable from **World 1** because C never makes use of the knowledge of i_t. Therefore, the scheme satisfies privacy with respect to $\mathcal{S}_{\text{query}}$.

The privacy with respect to $\mathcal{S}_{\text{hint}}$ is defined separately.

Definition B.7 (Privacy Against Selective Failure)**.** Given a security parameter λ, the scheme is private against selective failure with respect to $\mathcal{S}_{\text{hint}}$ if there exists a probabilistic polynomial time simulator $Sim(1^\lambda, N)$ such that for any algorithm $serv_{\text{query}}$, any probabilistic polynomial time adversaries $\mathcal{A}$ passively corrupting the hint server $\mathcal{S}_{\text{hint}}$, $\mathcal{A}$ cannot distinguish the view in the following world with probability non-negligible in λ.

- **World 0**: C interact with $\mathcal{A}$ who plays the role of $\mathcal{S}_{\text{hint}}$ and $serv_{\text{query}}$ who plays the roll of $\mathcal{S}_{\text{hint}}$. At each step t in the online phase, $\mathcal{A}$ adaptively chooses the next index i_t, and C uses i_t as its query. C output 1 to $\mathcal{A}$ after each query if the answer is accepted, otherwise 0.
- **World 1**: Sim interact with $\mathcal{A}$ who plays the role of $\mathcal{S}_{\text{hint}}$ and $serv_{\text{query}}$ who plays the roll of $\mathcal{S}_{\text{query}}$. At each step t in the online phase, $\mathcal{A}$ adaptively chooses the next index i_t, and Sim runs without the knowledge of i. Sim output a bit b to $\mathcal{A}$ after each query.

  We stress that $\mathcal{A}$ should follow the protocol honestly.

Privacy with respect to $\mathcal{S}_{\text{hint}}$ in the offline phase follows simply from the fact that the offline phase happens before any query is made. The proof for the online phase is symmetric to the argument with respect to $\mathcal{S}_{\text{query}}$. This completes the proof of privacy in the malicious model.

Privacy in the single-server setting is defined identically to the two-server privacy with respect to $\mathcal{S}_{\text{query}}$. We see that the only difference is the $serv_{\text{hint}}$ algorithm being replaced by a real interaction with $\mathcal{A}$, which happens before any query is made and does not affect privacy.