Authenticated private information retrieval

Simone Colombo *EPFL*
Kirill Nikitin *Cornell Tech*
Henry Corrigan-Gibbs *MIT*
David J. Wu *UT Austin*
Bryan Ford *EPFL*

Abstract. This paper introduces protocols for *authenticated* private information retrieval. These schemes enable a client to fetch a record from a remote database server such that (a) the server does not learn which record the client reads, and (b) the client either obtains the "authentic" record or detects server misbehavior and safely aborts. Both properties are crucial for many applications. Standard private-information-retrieval schemes either do not ensure this form of output authenticity, or they require multiple database replicas with an honest majority. In contrast, we offer multi-server schemes that protect security as long as at least one server is honest. Moreover, if the client can obtain a short digest of the database out of band, then our schemes require only a single server. Performing an authenticated private PGP-public-key lookup on an OpenPGP key server's database of 3.5 million keys (3 GiB), using two non-colluding servers, takes under 1.2 core-seconds of computation, essentially matching the time taken by unauthenticated private information retrieval. Our authenticated single-server schemes are 30-100× more costly than state-of-the-art unauthenticated single-server schemes, though they achieve incomparably stronger integrity properties.

1 Introduction

Private information retrieval (PIR) [31] enables a client to fetch a record from a database while hiding from the database server(s) which specific record(s) the client retrieves. PIR has numerous privacy-protection uses, such as in metadata-private messaging [5, 6], certificate transparency [63, 83], video streaming [51], password-breach alerting [4, 60, 88], retrieval of security updates [24], public-key directories [64], and private SQL-like queries on public data [74, 93].

Most PIR protocols, however, do not ensure data authenticity in the presence of malicious servers. In many multi-server PIR schemes [18, 31], a single adversarial server can flip any subset of bits in the client's recovered output. In all single-server PIR schemes we know of (c.f., [1, 4, 5, 19, 22, 32, 37, 46, 52, 57, 62, 66, 71, 76, 78] for a non-exhaustive list), a malicious server can choose the exact output that the client will receive by substituting all the database records with a chosen record before processing the client's request. In applications where data integrity matters, such as a PGP public-key directory, unauthenticated PIR is inadequate.

This is the full version of a paper with the same title appearing at USENIX Security 2023.

This paper introduces *authenticated private information retrieval*, which augments the standard privacy properties of classic PIR with strong authenticity guarantees. In the multi-server setting, we propose authenticated-PIR schemes for:

- *Point queries*, in which a client wants to fetch a particular database record. For example, "What is the public key for `user@usenix.org`?"
- *Predicate queries*, where a client wants to apply an aggregation operator – such as `COUNT`, `SUM`, or `AVG` – to all records matching a predicate. For example, "How many keys are registered for email addresses ending in `@usenix.org`?"

Our corresponding authenticated-PIR schemes guarantee integrity in the *anytrust* model [95]: as long as at least one of the PIR servers is honest. In contrast, prior work that deals with malicious or faulty PIR servers in the multi-server setting either requires a majority or supermajority of servers to be honest [11, 12, 39, 49] or requires expensive public-key cryptography operations [99]. Our schemes use only fast symmetric-key cryptography in the multi-server setting.

In the single-server setting, we offer authenticated-PIR schemes for point queries which provide authentication as long as the client can obtain a short digest of the database via out-of-band means (Fig. 1). Prior work for the single-server setting [57, 94, 100] ensures only that the server truthfully answers the query with respect to *some* database—not necessarily the database the client queried. Table 2 summarizes prior work and Section 8 gives the complete discussion.

New definitions. Our first contribution is a new definition of integrity for private information retrieval. In our multi-server PIR schemes, a client communicates with several database servers, and client privacy holds as long as at least one server is honest. In this multi-server setting, we say that a PIR scheme satisfies integrity if, whenever the client accepts the servers' answers, the client's output is consistent with an honest server's view of the database.

Defining integrity in the single-server setting is more tricky: If the single database server is malicious, who is to say what the "right" database is? Our approach assumes that the client can obtain a short digest of the database via some out-of-band means. A single-server PIR protocol satisfies integrity if the client accepts the protocol's output only if the output is consistent with the database that the digest represents. In some applications of PIR, the client could obtain this database digest via a gossip mechanism, as in CONIKS [65], or from a collective

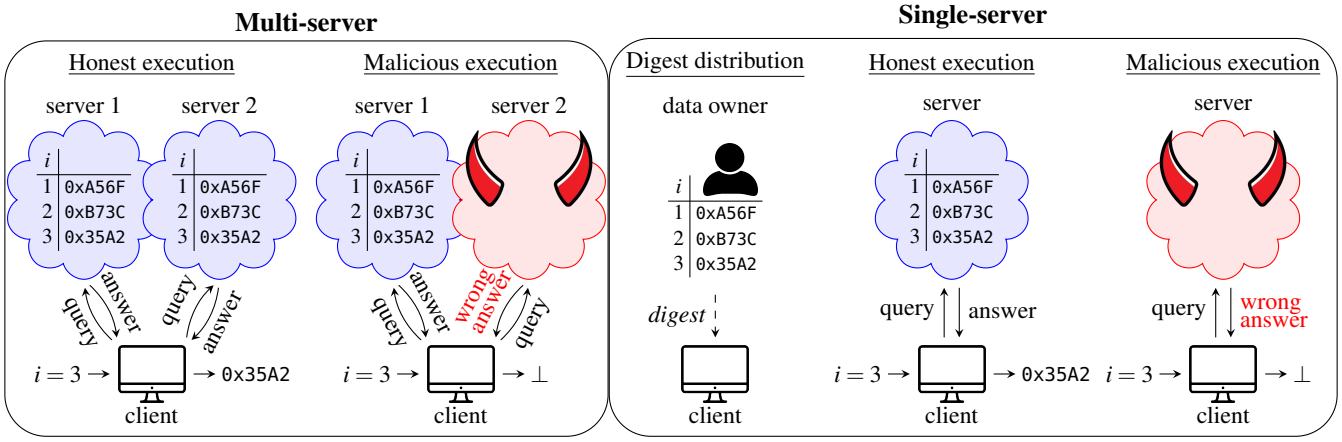

Figure 1: In multi-server authenticated PIR, $k \geq 2$ servers hold an exact replica of the database and the client's output is consistent with the honest server's view of the database. If at least one server is honest, the client detects any malicious behaviour from the other servers that reply with respect to an altered database, and rejects the answers. In the single-server setting, a potentially-malicious PIR server holds the database outsourced by the data owner. The client's output is consistent with a database digest that the client obtained from the honest data owner.

authority [86], or from a signature-producing blockchain [73]. In other applications of PIR such as video streaming [51], a database *owner*—distinct from the PIR servers—might produce, sign, and distribute this digest.

A subtle and important point is that our security definitions require protection against *selective-failure attacks* by malicious servers [53, 55, 57]. In this class of attacks, a malicious server answers the client's query with respect to a database that differs from the true database in a few rows. By observing whether the client accepts or rejects the resulting answer, the server can learn information about which rows the client had queried. To defend against these attacks, our security definitions require that *any* misbehavior on the part of a malicious server causes a client to reject the servers' response.

New constructions. We construct new authenticated-PIR schemes in the multi- and single-server settings.

Multiple servers, point queries. Our first multi-server PIR scheme allows the client to make only *point* queries—to fetch single records from the database. The scheme is simple to implement and has minimal performance overhead. In this scheme, the servers compute a Merkle tree over the database rows and send the client the Merkle root. The client aborts if the servers send different roots. The client then uses unauthenticated PIR to fetch its desired row and a Merkle inclusion proof with respect to the root. The scheme provides authentication when composed with certain—though not all—standard PIR schemes. (Kushilevitz and Ostrovsky suggested using Merkle trees in this setting [57], though we are the first to formalize the approach and identify the class of PIR schemes for which it is secure.) On a database containing N records of ℓ bits, and on security parameter λ, our two-server authenticated-PIR scheme for point queries has communication cost $O(\lambda \log N + \ell)$, which matches the cost of the best unauthenticated schemes. Experimentally, this form of authentication imposes less than $3\times$ computational and $1.6\times$ bandwidth overhead, compared with unauthenticated PIR.

Multiple servers, predicate queries. Our multi-server scheme for predicate queries starts with an existing unauthenticated scheme based on function secret sharing [17, 18, 93]. We cannot use Merkle trees for authentication: the space of possible queries is exponentially large, so the servers cannot precompute and authenticate each potential answer as before. The client instead uses an information-theoretic message-authentication code—common in malicious secure multi-party protocols [33, 35]—to detect whether a server has tampered with its answer. Asymptotically, the communication and computation of our authenticated-PIR scheme for predicate queries matches the costs of the corresponding unauthenticated scheme. Empirically, the authenticated scheme incurs a median overhead of less than $1.02\times$ for both user time and bandwidth. Our multi-server scheme for predicate queries is concretely more computationally expensive (at least $350\times$) than our scheme for point queries because the cost of evaluating the function secret shares is non-trivial. Thus, this scheme does not scale as well to a large number of servers compared to our specialized multi-server scheme for point queries.

Single server, point queries. Finally, we give two single-server authenticated-PIR protocols: one from the learning-with-errors assumption, and one from the decisional-Diffie-Hellman assumption. Like many recent single-server PIR protocols [1, 4, 5, 52], our schemes extend the classic Kushilevitz-Ostrovsky scheme based on additively homomorphic encryption [57, 75]. Our schemes incorporate additional randomness that the client uses to authenticate the server's response. The client verifies the server's reply using a short database digest that the client obtains via out-of-band means. Our schemes operate with single-bit records. We propose extensions for handling larger records, but they require increased

client computation: more efficient single-server, multi-bit authenticated PIR remains a promising area for future work. Over a database of size N and with security parameter λ, our single-server authenticated-PIR schemes have communication cost $\sqrt{N} \cdot \mathsf{poly}(\lambda)$. In contrast, unauthenticated schemes have communication cost as low as $\log N \cdot \mathsf{poly}(\lambda)$. Our fastest single-server scheme is 30-100× more computationally expensive than the fastest unauthenticated scheme.

An example application. To evaluate authenticated PIR in the context of a practical application, we design and build Keyd, a privacy-preserving PGP public-key directory deployed in the two-server setting. A Keyd client can query the servers for the PGP public key corresponding to a particular email address without leaking the queried email address to the servers. Moreover, a Keyd client can also query the servers for private analysis of the PGP public keys dataset by issuing conjunctive `COUNT`, `SUM` and `AVG` queries without leaking the parameter of the keys over which the predicate is computed. For example, a client can issue a query of the form `SELECT COUNT(*) FROM keys WHERE keyAlgorithm =` p, where p represents the hidden parameter of the predicate, e.g., RSA or ElGamal. Our new authenticated-PIR schemes provide the client with a strong integrity guarantee about the output of the protocols. When run on a recent dump of the SKS PGP key directory, including over 3.5 million keys, querying for a particular key takes the client 1.11 seconds, compared with 1.10 seconds with unauthenticated PIR. Issuing predicate queries with Keyd on the same database imposes an overhead of 1.01× on user time and of 1.05× on bandwidth compared with unauthenticated PIR.

2 Background and motivation

This section reviews classic PIR schemes, and why naïvely introducing integrity protection into them is unsafe.

2.1 Private information retrieval (PIR)

A PIR protocol [31] takes place between a client and one or more servers. Each server holds a copy of a database consisting of a set of equal-length records. The client wants to query the database without revealing the details of its query to the servers. Modern PIR protocols support two types of queries: (1) the client can fetch a single record from the database, without revealing *which record* it retrieved, or more generally, (2) the client can evaluate a function on all the database records, without revealing *which function* it evaluated. Non-trivial PIR schemes must also be communication efficient, requiring the client and servers to exchange a number of bits sublinear in the database size. Otherwise, the client could simply download the entire database and perform the query locally.

There are two main types of PIR protocols: multi-server and single-server. In multi-server PIR [31], the client communicates with $k > 1$ database replicas; correctness holds if all k servers are honest and privacy holds if at least one server is honest. Multi-server PIR schemes traditionally offer information-theoretic privacy. In single-server PIR schemes ($k = 1$) [57], correctness holds if the single server is honest and privacy holds against a dishonest server. Single-server PIR schemes require a computationally-bounded server and public-key cryptographic operations [34].

PIR scheme	*No. of honest servers needed*	*Malicious secure*	*Selective-failure secure*	*No public-key cryptography*	*Recovery*
Multi-server schemes					
Robust PIR [11, 12]	1	✗	✗	✓	✓
Byzantine PIR [11, 12, 39, 49, 56]	$>2k/3$	✓	✓	✓	✓
Fault-tolerant PIR [97]	$>k/2$	✓	✓	✓	✓
Verifiable PIR [99]	1	✓	✓	✗	✗
Authenticated PIR (§4, §5)	1	✓	✓	✓	✗
Single-server schemes					
KO97 [57]	0	✓	✗	✗	✗
Verifiable PIR [94, 100]	0	✓	✗	✗	✗
Authenticated PIR (§5)	0	✓	✓	✗	✗

Table 2: Summary of PIR schemes that tolerate dishonest servers. The multi-server schemes assume k servers in total. *Malicious* indicates schemes that resist malicious adversaries, as opposed to merely faulty servers. *Selective-failure secure* indicates schemes designed to resist selective-failure attacks [55]. *No public-key cryptography* indicates schemes that require only fast symmetric primitives; single-server schemes always require public-key operations [34]. *Recovery* indicates whether, in case of a server's misbehaviour, the client is able to recover the correct output or just aborts.

In many applications, the database is a list of (keyword, value) pairs; the PIR client holds a keyword and wants the associated value. In this paper, we construct authenticated PIR schemes for integer-indexed arrays, and we use off-the-shelf methods [29, 48] to convert these schemes into authenticated keyword-based PIR schemes.

2.2 Why integrity matters in PIR

Standard PIR schemes give the client *no integrity guarantees*. If any one of the servers in a single- or multi-server scheme deviates from the protocol, the malicious server can—in many PIR protocols—completely control the output that the client receives. In other words, classic PIR protocols do not ensure correctness against even just one malicious server.

This lack of integrity protection is extremely problematic in many applications of PIR:

- *Public-key server*: If a client uses PIR to query a PGP or Signal key server for a contact's public keys, a malicious server could cause the client to fetch a false public key for which the adversary controls the secret key.

- *Domain name system*: If a client uses PIR to query a DNS resolver, a malicious PIR server could cause the client to recover the wrong IP address for a hostname and thus poison the client's DNS cache.
- *Online certificate status protocol (OCSP)*: If a client uses PIR to query the revocation status of a public key, a malicious PIR server could trick the client into trusting a certificate that was revoked by the CA after compromise.
- *Content library*: If a client uses PIR to fetch a movie [51] or a software update, a malicious PIR server could cause the client to recover a malware-infected file instead.

Non-private variants of these applications can already offer integrity. For example, CONIKS [65] provides integrity of key bindings for public-key directory servers and DNSSEC [7] ensures integrity of DNS data. The challenge is thus to ensure integrity in the *private* variants of these applications.

2.3 Selective failure and other attacks on PIR

We can always compose standard authentication mechanisms with PIR. For example, a *database owner* – the party responsible for its creation – can append to each database row a digital signature on the record under the database owner's key or a Merkle inclusion proof with respect to a known root. The database owner can then outsource the authenticated database to an untrusted PIR server. After performing a query, the client simply checks the authentication tag on the row it retrieved.

This attempt at authenticated PIR is insecure and vulnerable to *selective-failure attacks* [55]. In such attacks, a malicious PIR server selectively corrupts the database so that only targeted queries fail the integrity check. Suppose a malicious PIR server "guesses" that the client is likely to access a particular record, and corrupts *only* that record. The client's integrity check then fails only if the attacker's guess was correct. If the attacker can determine whether the client accepted or rejected the PIR protocol's output—e.g., via the client's subsequent behavior—the attacker can violate client privacy.

Naïve composition can yield other security and privacy hazards. For example, if authentication tags attached to database rows do not uniquely identify the database version and row number, then a malicious PIR server might undetectably swap or duplicate rows or replay old database versions.

Even in a multi-server setting where one malicious server cannot unilaterally corrupt database rows independently, but is limited to blindly flipping bits in its answer without knowing which row these bit-flips will affect, more subtle attacks on naïve compositions may be readily feasible. If rows are protected by malleable digital signatures [40], for example, then a malicious server might flip signature bits in the result so that the signature of a particular "guessed" database row becomes a *different* still-valid signature the client will accept, while the signatures on all other rows become invalid.

3 Defining authenticated PIR

We now define *authenticated PIR* in the multi- and single-server settings. In both models, we wish to ensure that the client either obtains "correct" (authentic) output, or else safely rejects the answer without leaking any private information. Privacy must hold even if the PIR servers learn whether the client has accepted or rejected the answer. Therefore, our protocols protect against selective-failure attacks (Section 2.3).

Notation. We use $\mathbb{N}$ to denote the set of natural numbers. For $N \in \mathbb{N}$, $[N] = \{1, \ldots, N\}$. We use $\mathsf{negl}(\cdot)$ to denote a negligible function and $\mathsf{poly}(\cdot)$ to denote a fixed polynomial. Throughout, we use $\mathbb{F}$ to denote a finite field. We will typically take $\mathbb{F}$ to be the set of integers modulo a prime p with addition and multiplication modulo p. For a finite set S, we write $x \xleftarrow{R} S$ to indicate that x is sampled independently and uniformly at random from S. The symbol $\perp$ is an output that indicates rejections. For a group $\mathbb{G}$, we use $1_{\mathbb{G}}$ to denote the identity element. For finite sets S and T, we use $\mathsf{Funs}[S,T]$ to denote the set of all functions from S to T. By "efficient algorithm" we refer to a probabilistic polynomial time algorithm. In some settings, we will also consider hardness against *non-uniform* adversaries (i.e., polynomial-time algorithms that can additionally take polynomial-size advice as input, see Remark 37).

3.1 Multi-server definition

We now define k-server authenticated PIR schemes, for $k \geq 2$. See Appendix B for the full formalism.

Our definition generalizes private information retrieval to *weighted functions* of the database rows: the client has a secret function f in mind, which must come from a particular class of functions $\mathcal{F}$. The servers hold a database $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ and public "weights" $(w_1, \ldots, w_N)$, one per database row. The client's goal is to get the weighted sum of its private function f applied to each of the rows: $\sum_{i \in [N]} w_i f(i, \mathbf{x}_i)$. When the function class $\mathcal{F}$ is expressive enough, this general syntax subsumes not only the usual definition of multi-server PIR, but also more expressive PIR schemes for predicate queries.

Definition 1 (k-server authenticated PIR for predicate queries)**.** *A k-server authenticated PIR scheme for function class $\mathcal{F} \subseteq \mathsf{Funs}[[N] \times \{0,1\}^{\ell}, \mathbb{F}]$, database size $N \in \mathbb{N}$, and weights $\mathbf{w} \in \mathbb{F}^N$, consists of three efficient algorithms:*

- $\mathsf{Query}(1^{\lambda}, f) \to (\mathsf{st}, q_1, \ldots, q_k)$. *Given a security parameter λ, expressed in unary, and a function $f \in \mathcal{F}$, return secret client state st and queries $q_1, \ldots, q_k$, one per server.*
- $\mathsf{Answer}(\mathbf{X}, \mathbf{w}, q) \to a$. *Apply query q to database $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N) \in (\{0,1\}^{\ell})^N$ together with weights $\mathbf{w} = (w_1, \ldots, w_N) \in \mathbb{F}^N$ and return answer a.*
- $\mathsf{Reconstruct}(\mathsf{st}, a_1, \ldots, a_k) \to \left\{ \sum_{i \in [N]} w_i f(i, \mathbf{x}_i), \perp \right\}$. *Take as input client state st and answers $a_1, \ldots, a_k$ and return*

the weighted output of the function f applied to the rows of database $\mathbf{X}$, or an error $\perp$.

A k-server authenticated-PIR protocol must satisfy the following properties. We state the properties here informally and give formal cryptographic definitions in Appendix B.

Correctness. Informally, an authenticated-PIR scheme is *correct* if, when an honest client interacts with honest servers, the client always recovers the weighted output of its chosen function applied to the database, i.e., $\sum_{i\in[N]} w_i f(i,\mathbf{x}_i)$.

Integrity. An authenticated-PIR scheme preserves *integrity with error* ε if, when an honest client interacts with a set of k servers, where at most $k-1$ can be malicious and might arbitrarily deviate from the protocol, the client either: outputs the sum of products of its desired function and weights applied to the database, or outputs the error symbol $\perp$, except with probability ε. If the scheme has negligible integrity error, we just say that it "preserves integrity." Classic PIR schemes do not ensure this integrity property.

Privacy (against malicious servers). An authenticated-PIR scheme satisfies *privacy* if any coalition of up to $k-1$ malicious servers "learns nothing"—in a strong cryptographic sense—about which function in the function class $\mathcal{F}$ the client wants to evaluate on the database, even if the servers learn whether the client's output was the error symbol $\perp$ during reconstruction. Standard PIR schemes do not necessarily satisfy our strong notion of privacy, since such schemes may be vulnerable to selective-failure attacks (Section 2.3); authenticated-PIR schemes that provide privacy are not.

We say that an authenticated-PIR scheme is *secure* if it satisfies both integrity and privacy. We define integrity and privacy separately because, as Section 3.3 shows, we can reduce the integrity error of a PIR scheme that provides privacy.

Example 2 (PIR for point queries—Standard PIR)**.** In authenticated-PIR schemes for point queries, as in a standard PIR scheme, a client privately fetches a single database row. We can recover this functionality from Definition 1, where we take the row length $\ell = 1$ for simplicity. The class of functions $\mathcal{F}$ is the class of point functions $\mathcal{F} = \{f^{(1)},\ldots,f^{(N)}\} \subseteq \mathsf{Funs}[[N]\times\{0,1\},\mathbb{F}]$, where $f^{(i)}(i,\cdot) = 1$ and $f^{(i)}(i',\cdot) = 0$ for all $i' \neq i$. The weights are the database entries themselves, i.e., $w_i = x_i \in \{0,1\} \subseteq \mathbb{F}$, for $i \in [N]$.

Example 3 (`COUNT` query)**.** A `COUNT` predicate query counts the database entries satisfying a predicate. A client can count the occurrences of a string $\sigma \in \{0,1\}^\ell$ in a database $\mathbf{x}_1,\ldots,\mathbf{x}_N \in \{0,1\}^\ell$ using the class of functions $\mathcal{F} \subseteq \mathsf{Funs}[[N]\times\{0,1\}^\ell,\mathbb{F}]$, where $f(\cdot,\mathbf{x}_i) = 1$ if $\mathbf{x}_i = \sigma$ and $f(\cdot,\mathbf{x}_i) = 0$ otherwise, with constant weights $w_i = 1_\mathbb{F}$, $i \in [N]$.

Remark 4 (Security against $k-1$ malicious servers)**.** The form of authenticated PIR we define above requires security to hold even against coalitions of up to $k-1$ malicious servers. This defines the minimal requirement for multi-server PIR schemes, which do not support complete collusion, and is a model frequently used in anonymous communication systems [6, 58, 95]. In particular, the colluding servers can share their queries with each other and agree on the answers. The protocols that we construct satisfy this strong notion of security. A weaker definition requires security to hold against only adversaries that control a lower threshold $t < k-1$ of the servers. Prior work [11, 12, 49] takes $t < k/2$ or $t < k/3$. We discuss these and other related approaches in Section 8.

3.2 Single-server definition

This section defines single-server authenticated PIR. One challenge to providing integrity in the single-server setting is that the client has no source of information about the database content other than the server itself. (In the multi-server setting, the honest server acts as a source of "ground truth.") A malicious server can answer the client's query with respect to a database of the server's choosing, and completely control the client's output. We address this problem by introducing a public database digest that cryptographically binds the server to a given database and serves as the ground truth in the scheme. In applications, the client must obtain this digest via out-of-band means, e.g., via gossip, as in CONIKS [65], or from the database owner if the latter is distinct from the PIR server.

We now give the formal definition of a single-server authenticated-PIR scheme, which differs from the multi-server definition in its use of a digest and in the absence of complex queries. We assume for simplicity that each database record consists of a single bit. The definition generalizes naturally to databases with longer rows.

Definition 5 (Single-server authenticated PIR for point queries)**.** *A* single-server authenticated PIR *scheme, for a database of size $N \in \mathbb{N}$, consists of the following algorithms:*

- $\mathsf{Digest}(1^\lambda,\mathbf{x}) \to d$. *Take a security parameter λ (in unary) and a database $\mathbf{x} \in \{0,1\}^N$ and return a digest d.*
- $\mathsf{Query}(d,i) \to (\mathsf{st},q)$. *Take as input a digest d and an index $i \in [N]$ and return a client state* st *and a query q.*
- $\mathsf{Answer}(d,\mathbf{x},q) \to a$. *Apply query q to database $\mathbf{x} \in \{0,1\}^N$ with digest d and return answer a.*
- $\mathsf{Reconstruct}(\mathsf{st},a) \to \{0,1,\perp\}$. *Take as input state* st *and answer a and return a database bit or an error $\perp$.*

A single-server authenticated-PIR scheme must satisfy analogous properties to those in the multi-server setting: correctness, integrity and privacy. If a scheme satisfies both integrity and privacy, we say that the scheme is secure. We present the formal definitions in Appendix E.

Malformed digest. Our schemes guarantee integrity for single-server authenticated PIR only when the client uses an honestly-generated digest. In all applications of single-server PIR that we envision, this security guarantee is sufficient—the client's goal is to check that a (possibly malicious) PIR

server's answer is consistent with the (correct) digest that the client has obtained out-of-band from the data owner. Stronger notions of security are possible, however. We could require that even if the digest is generated adversarially, the client is guaranteed to recover output that is consistent with *some* n-bit database. This stronger notion is related to that of simulatable adaptive oblivious transfer [23] and extends to other cryptographic primitives [45, 54].

3.3 Integrity amplification

The lattice-based single-server authenticated-PIR schemes that we construct in Section 5 have noticeable integrity error $\varepsilon = 1/\mathsf{poly}(\lambda)$ for some parameter settings. We show, in Appendix E.2, that if the authenticated-PIR schemes provide privacy, then it is possible to reduce the integrity error to a negligible quantity, in both the multi- and single-server settings. In particular, we prove:

Theorem 6 (Integrity amplification, informal)**.** *If Π is an authenticated-PIR scheme with privacy and with integrity error ε then, for every $t \in \mathbb{N}$, there is an authenticated-PIR scheme Π' with privacy and with integrity error ε^{t+1}, where Π' invokes Π at most $2t+1$ times.*

The integrity-amplification construction first encodes the database using an error-correcting code that can correct t errors. For instance, using the simple repetition code, we expand each database bit into $2t+1$ codeword bits. (When the database records are long, we can use better error-correcting codes.) Then, the client uses the base authenticated PIR scheme Π $2t+1$ times to fetch each of the $2t+1$ bits of the codeword corresponding to its desired database record.

If any of these $2t+1$ runs output $\perp$, the client outputs $\perp$. If none of the $2t+1$ runs output $\perp$, then either: (a) the client recovers at least $t+1$ correct bits of the codeword, in which case the client correctly recovers its desired output bit, or (b) the client recovers an incorrect bit on more than t of the protocol runs, which happens with probability at most ε^{t+1}, by the ε-integrity of the underlying PIR scheme.

4 Multi-server authenticated PIR

We give two constructions of multi-server authenticated PIR.

4.1 Point queries via Merkle trees

We first present a multi-server authenticated-PIR scheme for *point queries*. This scheme enables a client with a secret index $i \in [N]$ to retrieve the i^{th} record from a database of N records.

A natural way to construct an authenticated-PIR scheme is to combine a standard (unauthenticated) multi-server PIR scheme with a standard integrity-protection mechanism, such as Merkle trees [67]. While this composition is in general *insecure* under our definition, we show that it can be secure with a careful choice of the underlying primitives.

We sketch the construction here and formally present it in Appendix C (Construction 4). This construction uses a standard multi-server PIR scheme in which (a) the client sends a single message to each server and receives a single message in return and (b) client reconstructs its output by summing up (or XORing) the answers from the servers. Many standard PIR schemes have this form [18, 31, 32, 48] (see Definition 15).

In these schemes, if any of the servers deviate from the prescribed protocol, the worst they can do is to cause the client to recover the correct output shifted by a constant of the adversarial servers' choosing. Therefore, instead of recovering the message $m \in \{0,1\}^\ell$, the client recovers $m \oplus \Delta$, for some non-zero value $\Delta \in \{0,1\}^\ell$.

Our approach then is to have the servers compute a Merkle tree over the N database entries along with their indices: $\{(1,\mathbf{x}_1),\ldots,(N,\mathbf{x}_N)\}$. Call the root of the tree R. Then for each entry, each server constructs a Merkle proof π_i of inclusion in the tree rooted at R and attaches this proof to each database record. The asymptotic complexity of this preprocessing phase is $O(N)$; we discuss concrete costs in Section 7 and Appendix C.3. Finally, the client and servers run the PIR protocol over the database $\{(1,\mathbf{x}_1,\pi_1),\ldots,(N,\mathbf{x}_N,\pi_N)\}$. Each of the servers also sends the Merkle root R to the client.

The client first checks that it received the same Merkle root R from all of the servers. Since at least one of the servers is honest, this ensures the client receives the *honestly-generated* root. If all the roots match, the client reconstructs the record and verifies the Merkle inclusion proof with respect to R. If a server misbehaves, the client will recover $(i',\mathbf{x}_i',\pi_i') = (i,\mathbf{x}_i,\pi_i) \oplus \Delta$ for some non-zero offset Δ. Whenever $\Delta \neq 0$, security of the Merkle proof ensures that π_i' will be an invalid proof of $(i,\mathbf{x}_i)$ with respect to R.

4.2 Predicate queries via function sharing

Recent work on function secret sharing [17, 18] in the multi-server PIR setting enables a client to compute a non-trivial function f over the database contents, without revealing this function f to the servers. For example, a client can count the number of database records that match a certain predicate, without revealing this predicate to the servers.

We design an authenticated-PIR protocol for predicate queries by extending classic PIR schemes based on function secret sharing [17, 18]. At a high level, the client makes two correlated PIR queries. The reconstructed answer to the first query should contain the value v that the client wants. The reconstructed answer to the second query should contain $v' = \alpha v$, where α is a random scalar known only to the client. To authenticate the servers' answers, the client checks that $\alpha v = v'$ and rejects if not. As we will show, if any server misbehaves, the client will be checking that $\alpha(v+\Delta) = v' + \Delta'$,

for some non-zero Δ and Δ'. Sampling α from a sufficiently large space of values ensures that the client catches a cheating server almost certainly.

This idea of using secret-shared random values for data authentication follows a long line of work on information-theoretic message authentication codes and malicious-secure multiparty computation [16, 33, 35, 38].

We now describe our construction in detail.

Preliminaries: Function secret sharing. We recall the definition of function secret sharing [17, 18]: A k-party *function secret-sharing* scheme is defined with respect to a function class $\mathcal{F}$. Each function $f \in \mathcal{F}$ maps elements in some input space to a finite group or field $\mathbb{F}$. Then a function secret-sharing scheme consists of two efficient algorithms:

- $\mathsf{Gen}(1^\lambda, f) \to (f_1, \ldots, f_k)$. Given a function $f \in \mathcal{F}$, output k function-secret-shares $f_1, \ldots, f_k$.
- $\mathsf{Eval}(f_i, x) \to f_i(x) \in \mathbb{F}$. Given a secret-share f_i and a function input x, output the evaluation of f_i on x.

A function secret-sharing scheme must satisfy the following informal properties, defined formally in Appendix A.3:

- **Correctness.** Given shares $(f_1, \ldots, f_k)$ of a function $f \in \mathcal{F}$, for all x in the domain of f, it holds that $\sum_{i \in [k]} \mathsf{Eval}(f_i, x) = f(x) \in \mathbb{F}$.
- **Security.** Given shares $(f_1, \ldots, f_k)$ of a function $f \in \mathcal{F}$, a computationally-bounded adversary that learns $k-1$ of the shares learns nothing about the shared function f, beyond the fact that $f \in \mathcal{F}$.

For the construction, we need the following definition:

Definition 7 (Function class closed under scalar multiplication)**.** *Let $\mathcal{F}$ be a class of functions whose codomain is a finite field $\mathbb{F}$. Then we say that the function class $\mathcal{F}$ is* closed under scalar multiplication *if, for all functions $f \in \mathcal{F}$ and for all scalars $\alpha \in \mathbb{F}$, it holds that the function $\alpha \cdot f \in \mathbb{F}$.*

Construction. Our scheme, presented in Construction 1, is defined with respect to a finite field $\mathbb{F}$, a record length $\ell \in \mathbb{N}$, a database size $N \in \mathbb{N}$, a function class $\mathcal{F} \subseteq \mathsf{Funs}[[N] \times \{0,1\}^\ell, \mathbb{F}]$ closed under scalar multiplication, and weights $w \in \mathbb{F}^N$. The $k \geq 2$ servers each hold a copy of a database of N ℓ-bit records. We write the n database records as $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \{0,1\}^\ell$. Given a predicate function $f \in \mathcal{F}$, the client samples a random non-zero field element $\alpha \in \mathbb{F}$ and secret-shares f together with a new function g defined as $g(i, x_i) = \alpha \cdot f(i, x_i) \in \mathbb{F}$ into k shares, i.e., f_j and g_j for $j \in [k]$. (Alternatively, if the underlying function-secret-sharing scheme supports it, the client can also secret share the single function $(f(i, x_i), g(i, x_i))$ whose image is in $\mathbb{F}^2$.)

Upon receiving the shares, each server $j \in [k]$ sets each element of its answer tuple to the sum of the function shares' evaluations on all the database records multiplied by the corresponding weights: i.e., $\mathbf{a}_j \leftarrow \left(\sum_{i \in [N]} w_i \cdot f(i, \mathbf{x}_i), \sum_{i \in [N]} w_i \cdot g(i, \mathbf{x}_i)\right) \in \mathbb{F}^2$. The servers directly evaluate the function shares on the database records. The client adds the answer vectors and reconstructs an intermediate value $\mathbf{a} \leftarrow \sum_{j \in [k]} \mathbf{a}_j \in \mathbb{F}^2$.

If all the servers are honest, the client-reconstructed value $\mathbf{a}$ equals $\mathbf{a} = (a_1, a_2) = \left(\sum_{i \in [N]} w_i \cdot f(i, \mathbf{x}_i), \alpha \cdot \sum_{i \in [N]} w_i \cdot f(i, \mathbf{x}_i)\right)$. The client then verifies that $\alpha \cdot a_1 = a_2$. As α is randomly generated and secret-shared among the servers, only the client knows its value. If $\alpha \cdot a_1 \neq a_2$, then the client rejects. Otherwise, the client accepts and outputs a_1.

Proof sketch. To explain how this approach protects integrity, we argue by contradiction. Say that server $j \in [k]$

Construction 1 (k-server authenticated PIR for predicate queries tolerating $k-1$ malicious servers)**.** The construction is parametrized by a number of servers $k \in \mathbb{N}$, a number of database rows $N \in \mathbb{N}$, a row length $\ell \in \mathbb{N}$, a finite field $\mathbb{F}$, a security parameter λ, a function class $\mathcal{F} \subseteq \mathsf{Funs}[[N] \times \{0,1\}^\ell, \mathbb{F}]$ that is closed under scalar multiplication, and a function-secret-sharing scheme $(\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ for the function class $\mathcal{F}$, parametrized by λ. We represent the database as N binary strings, each of length ℓ: $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \{0,1\}^\ell$.

$\underline{\mathsf{Query}\left(1^\lambda, f\right) \to (\mathsf{st}, \mathbf{q}_1, \ldots, \mathbf{q}_k)}$

1. Sample a random field element $\alpha \xleftarrow{R} \mathbb{F} \setminus \{0\}$.
2. Set the state $\mathsf{st} \leftarrow \alpha$.
3. Let $g \leftarrow \alpha \cdot f$. Such a g must exist since the function class $\mathcal{F}$ is closed under scalar multiplication, as in Definition 7.
4. Compute $q_1, \ldots, q_k \leftarrow \mathsf{FSS.Gen}(1^\lambda, f)$ together with $q'_1, \ldots, q'_k \leftarrow \mathsf{FSS.Gen}(1^\lambda, g)$.
5. Output $\left(\mathsf{st}, (q_1, q'_1), \ldots, (q_k, q'_k)\right)$.

$\underline{\mathsf{Answer}\left(\mathbf{x}_1, \ldots, \mathbf{x}_N \in \{0,1\}^\ell, \mathbf{w} \in \mathbb{F}^N, \mathbf{q}\right) \to \mathbf{a} \in \mathbb{F}^2}$

1. Parse $\mathbf{q}$ as (q_f, q_g).
2. Compute answer as $a_f \leftarrow \sum_{j \in [N]} w_j \cdot \mathsf{FSS.Eval}(q_f, \mathbf{x}_j)$ and $a_g \leftarrow \sum_{j \in [N]} w_j \cdot \mathsf{FSS.Eval}(q_g, \mathbf{x}_j)$.
3. Return $\mathbf{a} \leftarrow (a_f, a_g) \in \mathbb{F}^2$.

$\underline{\mathsf{Reconstruct}\left(\mathsf{st}, \mathbf{a}_1, \ldots, \mathbf{a}_k \in \mathbb{F}^2\right) \to \mathbb{F} \cup \{\perp\}}$

1. Parse the state st as $\alpha \in \mathbb{F}$.
2. Compute $\mathbf{a} \leftarrow \mathbf{a}_1 + \cdots + \mathbf{a}_k \in \mathbb{F}^2$.
3. Parse $\mathbf{a}$ as $(m, \tau) \in \mathbb{F}^2$.
4. Compute $\tau' \leftarrow m \cdot \alpha \in \mathbb{F}$.
5. If $\tau = \tau'$, output $m \in \mathbb{F}$. Otherwise, output $\perp$.

should have returned an answer $\mathbf{a}_j \in \mathbb{F}^2$ to the client. Suppose server j is malicious and returns an answer $\hat{\mathbf{a}}_j = \mathbf{a}_j + \Delta \in \mathbb{F}^2$ for some non-zero value $\Delta = (\Delta_m, \Delta_\tau) \in \mathbb{F}^2$. The client will reconstruct the answer as $\mathbf{a} + \Delta = \left(\sum_{i\in[N]} w_i \cdot f(i,\mathbf{x}_i) + \Delta_m, \alpha \cdot \sum_{i\in[N]} w_i \cdot f(i,\mathbf{x}_i) + \Delta_\tau\right) \in \mathbb{F}^2$. As server j has no information about α—due to the privacy guarantees of the function-secret-sharing scheme—the malicious server's choice of Δ is (computationally) independent of α. For the verification to pass, it must be that $\alpha \cdot \Delta_m = \Delta_\tau$. If $\Delta \neq 0$ and α is sampled independent of Δ, this happens with probability at most $1/(|\mathbb{F}| - 1)$ over the randomness of α. Next, the privacy of the client's queries is ensured by the underlying function secret-sharing scheme. In Appendix D.1, we formally prove that this construction is secure.

Theorem 8. *Suppose there exists a k-party function-secret-sharing scheme for a function class $\mathcal{F} \subseteq \mathsf{Funs}[[N] \times \{0,1\}^\ell, \mathbb{F}]$ that is closed under scalar multiplication (Definition 7), for database size $N \in \mathbb{N}$, which, on security parameter $\lambda \in \mathbb{N}$, outputs secret shares of length $L(\lambda)$. Then, there is a k-server authenticated-PIR scheme for function class $\mathcal{F}$ with query complexity $2L(\lambda)k$ bits and answer complexity $2k\lambda$ bits.*

By applying the two-party function-secret-sharing scheme of Boyle, Gilboa, and Ishai [18], we get:

Corollary 9. *Given a length-doubling pseudorandom generator with seed length λ, there is a two-server authenticated PIR scheme for point functions and interval functions with communication complexity $O(\lambda \log N)$, on security parameter λ and database size N.*

Handling functions with larger output. In some PIR applications, a client might want to evaluate a function whose output is larger than a single field element, e.g., geographical coordinates for route planners [93]. We hence extend our scheme to support multi-element authenticated output.

Here, we authenticate each output element of a function f with a separate function g_j, for $j \in [b]$, where b is the output length of f using an algebraic manipulation detection code [33]. In the query algorithm, the client generates a secret random scalar α as before but then computes $(g_1(i,\mathbf{x}_i), g_2(i,\mathbf{x}_i), \ldots, g_b(i,\mathbf{x}_i)) = (\alpha, \alpha^2, \ldots, \alpha^b) \odot f(i,\mathbf{x}_i)$, where $\odot$ represents the element-wise product, and sends secret-shared f and $g_1, \ldots, g_b$ to the servers. The servers then compute their answer as $\mathbf{a} \leftarrow (\mathbf{a}_f, \mathbf{a}_{g_1}, \ldots, \mathbf{a}_{g_b}) \in \mathbb{F}^{2b}$.

This already enables the client to validate integrity of the full output after the reconstruction by comparing it with $\mathbf{a}_{g_1}, \ldots, \mathbf{a}_{g_b}$. We further reduce the protocol's communication cost by setting the servers' answer to $(\mathbf{a}_f, \mathbf{a}_g = \sum_{i\in[b]} \mathbf{a}_{g_i}) \in \mathbb{F}^{b+1}$. The client re-computes this linear combination from the answer and compares it with the received value.

We show the full construction in Appendix D.2.

5 Single-server authenticated PIR

We now present a single-server authenticated-PIR scheme.

As depicted in Fig. 1, in this setting a data owner outsources the data to a single PIR server (e.g., an Amazon EC2 instance) and produces a database digest. This public digest serves as a commitment to the database contents. The client can fetch the digest from a distributed authority, or using a CONIKS-like gossip protocol [65], or out-of-band from the data owner.

It is possible in principle to construct single-server authenticated-PIR schemes by augmenting a standard single-server PIR scheme [5, 37, 52, 66, 71] with a succinct proof of correct server execution [77], but this would be orders of magnitude more costly in computation than our schemes are.

Preliminary: Rebalancing to get $\sqrt{N}$ communication. Our single-server authenticated-PIR schemes natively have a digest of size $\mathsf{poly}(\lambda)$ bits, upload $N \cdot \mathsf{poly}(\lambda)$ bits, and download $\mathsf{poly}(\lambda)$ bits. To reduce total communication to $\sqrt{N} \cdot \mathsf{poly}(\lambda)$ bits, we use a standard rebalancing trick [31].

The server first splits the database into $\sqrt{N}$ chunks, each of size $\sqrt{N}$. The digest then consists of the hash (with any collision-resistant hash function, e.g., SHA-256) of the $\sqrt{N}$ database digests. To query the database for the i^{th} row of the j^{th} chunk, the client issues a single query for row i. The server responds with the $\sqrt{N}$ chunk digests, and the answer computed against each chunk. The client checks that (1) the hash of the $\sqrt{N}$ chunk digests match the database digest and (2) all $\sqrt{N}$ chunk queries accept. If these checks pass, the client outputs the value of the j^{th} response as its answer.

5.1 From learning with errors

Our first single-server authenticated-PIR scheme builds on lattices and relies on the learning-with-errors assumption (LWE) [82] (see Definition 43 for a formal statement). The LWE assumption with parameters $n, q, m, s \in \mathbb{N}$, states that the two distributions $(\mathbf{A}, \mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T})$ and $(\mathbf{A}, \mathbf{u}^\mathsf{T})$ are computationally indistinguishable, where $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m}$, $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow D_{\mathbb{Z},s}^m \in \mathbb{Z}_q^m$, and $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^n$, and where $D_{\mathbb{Z},s}$ is the discrete-Gaussian distribution with width parameter s (cf. Appendix F.1).

Construction 2 describes our scheme, which is a twist on Regev's LWE-based encryption scheme [82] and is an authenticated analogue of the SimplePIR LWE-based PIR scheme [52]. (We compare against SimplePIR in Section 7.) Regev's scheme encrypts a vector $\mathbf{v} \in \{0,1\}^N \subseteq \mathbb{Z}_q^N$ by the pair $(\mathbf{A}, \mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T} + t \cdot \mathbf{v}^\mathsf{T})$, where $\mathbf{A} \in \mathbb{Z}_q^{n\times N}$ is the LWE matrix, $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$ is the LWE secret, $\mathbf{e} \leftarrow D_{\mathbb{Z},s}^N$ is the error vector, and $t \in \mathbb{Z}_q$ is some scaling factor (commonly set to $q/2$). Regev's scheme is linearly homomorphic: for any vector $\mathbf{x} \in \{0,1\}^N \subseteq \mathbb{Z}_q^N$, the ciphertext $(\mathbf{A}\mathbf{x}, (\mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T} + t \cdot \mathbf{v}^\mathsf{T}) \cdot \mathbf{x})$ decrypts to $\mathbf{v}^\mathsf{T}\mathbf{x}$ (provided the accumulated error $\mathbf{e}^\mathsf{T}\mathbf{x}$ is small compared to t).

In our scheme, the first portion of this ciphertext ($\mathbf{A} \cdot \mathbf{x}$, on

database $\mathbf{x} \in \{0,1\}^N \subseteq \mathbb{Z}_q^N$) becomes the digest. Finding two distinct databases that map to the same digest is as hard as solving the short integer solutions problem [2].

To query for database record $i \in [N]$, the client prepares the Regev encryption $\mathbf{q}^\mathsf{T}$ of the i^{th} basis vector $\eta_i \in \mathbb{Z}_q^N$ (i.e., η_i is the vector that is 0 everywhere and 1 at index i). The scaling factor $t \in \mathbb{Z}_q$ is sampled randomly (from an appropriate range), which is critical for the security analysis. To answer the query, the server homomorphically computes the encryption of the inner product of the client's query with the database: $\mathbf{q}^\mathsf{T}\mathbf{x} \in \mathbb{Z}_q$. The client checks that the decrypted value is either 0 (indicating a database bit of zero) or close to t (indicating a database bit of one). Otherwise, the client outputs $\perp$.

Finally, by rebalancing Construction 2, we have:

Theorem 10. *Under the LWE assumption, Construction 2 is a secure single-server authenticated-PIR scheme when instantiated with database size N, lattice parameters (n,q,s), random matrix $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n\times N}$, and bound $B = O(\sqrt{\lambda N}s)$. The digest size consists of $n\sqrt{N}$ elements of $\mathbb{Z}_q$ and the per-query communication cost is $2\sqrt{N}$ elements of $\mathbb{Z}_q$. The scheme has integrity error $\varepsilon < 2B/(q-4B)$.*

The most important difference between SimplePIR [52] and Construction 2 is in the choice of LWE parameters. Since the integrity error is roughly $\sqrt{N}/q$, on database size N and modulus q, we must take the modulus q to be at least 128 bits to achieve negligible integrity error. (Alternatively, we can use a smaller modulus and run the protocol many times to amplify integrity as per Section 3.3.) In contrast, SimplePIR uses a 32-bit modulus with no repetition.

Construction 2 (Single-server authenticated PIR from LWE)**.** The construction is parametrized by a database length $N \in \mathbb{N}$, a lattice dimension $n \in \mathbb{N}$, a modulus $q \in \mathbb{N}$, a Gaussian width parameter $s \in \mathbb{N}$, a bound $B \in \mathbb{N}$, and a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times N}$. The database is a vector $\mathbf{x} \in \{0,1\}^N$.

$\underline{\mathsf{Digest}(\mathbf{x} \in \{0,1\}^N) \to \mathbf{d} \in \mathbb{Z}_q^n}$

1. Output $\mathbf{d} \leftarrow \mathbf{A}\mathbf{x} \in \mathbb{Z}_q^n$.

$\underline{\mathsf{Query}\left(\mathbf{d} \in \mathbb{Z}_q^N, i \in [N]\right) \to (\mathsf{st}, \mathbf{q})}$

1. Sample $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow D_{\mathbb{Z},s}^N \in \mathbb{Z}_q^m$, and $t \xleftarrow{R} \left[2B, q-2B\right]$. (Here $D_{\mathbb{Z},s}$ denotes the discrete Gaussian distribution over $\mathbb{Z}$ with parameter s, as in Appendix F.1.)
2. Compute $\mathbf{q}^\mathsf{T} \leftarrow \mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T} + t \cdot \eta_i^\mathsf{T} \in \mathbb{Z}_q^m$, where $\eta_i \in \mathbb{Z}_q^N$ denotes the i^{th} standard basis vector (i.e., the vector that is 0 everywhere except 1 in index i).
3. Set $\mathsf{st} \leftarrow (\mathbf{d}, \mathbf{s}, t)$ and output $(\mathsf{st}, \mathbf{q})$.

$\underline{\mathsf{Answer}\left(\mathbf{d} \in \mathbb{Z}_q^n, \mathbf{x} \in \{0,1\}^N \subseteq \mathbb{Z}_q^N, \mathbf{q} \in \mathbb{Z}_q^N\right) \to a \in \mathbb{Z}_q}$

1. Output $a \leftarrow \mathbf{q}^\mathsf{T}\mathbf{x} \in \mathbb{Z}_q$

$\underline{\mathsf{Reconstruct}\left(\mathsf{st}, a\right) \to \{0,1,\perp\}}$

1. Parse the state st as $(\mathbf{d}, \mathbf{s}, t)$.
2. If there exists $k \in \{0,1\}$ such that $|a - \mathbf{s}^\mathsf{T}\mathbf{d} - kt| < B$, then output k. Otherwise, output $\perp$.

5.2 From decisional Diffie-Hellman

This second construction uses the decisional Diffie-Hellman assumption (DDH). DDH holds in a group $\mathbb{G}$ of prime order p generated by $g \in \mathbb{G}$, if for $x, y, z \xleftarrow{R} \mathbb{Z}_p$, the two distributions (g, g^x, g^y, g^{xy}) and (g, g^x, g^y, g^z) are computationally indistinguishable (see Appendix G.1 for a formal definition).

Construction 3 details our scheme, which uses a group $\mathbb{G}$ of large prime order p. The database is a vector of N bits $\mathbf{x} = (x_1, \ldots, x_N) \in \{0,1\}^N$. The public parameters of the scheme include group elements $h_1, \ldots, h_N \in \mathbb{G}$. The digest is the product $d \leftarrow \prod_{j=1}^N h_j^{x_j} \in \mathbb{G}$. Finding two distinct databases that map to the same digest is as hard as solving the discrete-log problem in $\mathbb{G}$ [79].

The protocol operates as follows. The client samples two random values $r, t \xleftarrow{R} \mathbb{Z}_p$. The client then prepares a vector of N group elements. Say the client wants to fetch the i^{th} database bit. For $j \in [N]$, the j^{th} component of this vector is $q_j \leftarrow h_j^{r+t}$ if $j = i$ and is $q_j \leftarrow h_j^r$ otherwise. Under DDH, the server cannot differentiate between q_i and q_j for $j \neq i$.

The client queries the server with the resulting blinded vector $(q_1, \ldots, q_N)$. The server exponentiates each vector element to the corresponding database bit and computes the product $a = \prod_{j\in[N]} q_j^{x_j}$. If the server honestly executes the protocol, the client receives back the product of the blinded digest d^r and (a) either the group identity (when the retrieved bit is zero) or (b) the blinding factor h^t associated with the element of interest (when the retrieved bit is one). If the server returns any answer apart from the one prescribed by the protocol, the client detects this and rejects with overwhelming probability.

We then have, by rebalancing Construction 3:

Theorem 11. *If the DDH assumption holds in group $\mathbb{G}$, then Construction 3 is a secure single-server authenticated-PIR scheme when instantiated with database size N and group $\mathbb{G}$. The digest size consists in $\sqrt{N}$ elements of $\mathbb{G}$ and the per-query communication cost is $2\sqrt{N}$ elements of $\mathbb{G}$. The scheme has negligible integrity error.*

The scheme could be extended to retrieve multi-bit database entries in two readily-apparent ways. The first and simplest approach is to run Construction 3 in parallel for each bit of the entry. The second approach requires the client to solve tractable discrete logarithms, as we describe in Appendix G.5.

Incremental digest maintenance. We envision that the data owner would generate the database digest and publish it on a client-accessible website or a tamper-resistant log. If a

Construction 3 (Single-server authenticated PIR from DDH)**.** The construction is parametrized by a database length $N \in \mathbb{N}$, a group $\mathbb{G}$ of prime order p, and group elements $h_1, \ldots, h_N \in \mathbb{G}$. The database is a vector $\mathbf{x} \in \{0,1\}^\ell \subseteq \mathbb{Z}_p^N$.

$\underline{\mathsf{Digest}(\mathbf{x} \in \{0,1\}^N) \to d \in \mathbb{G}}$

1. Output $d \leftarrow \prod_{j \in [N]} h_j^{x_j} \in \mathbb{G}$.

$\underline{\mathsf{Query}\,(d \in \mathbb{G}, i \in [N]) \to (\mathsf{st}, q)}$

1. Sample two random values $r,t \xleftarrow{R} \mathbb{Z}_p$.
2. For $j \in [N] \backslash \{i\}$, compute $q_j \leftarrow h_j^r \in \mathbb{G}$.
3. Compute $q_i \leftarrow h_i^{r+t} \in \mathbb{G}$.
4. Set $\mathsf{st} \leftarrow (i,d,r,t)$.
5. Set $\mathbf{q} \leftarrow (q_1, \ldots, q_N) \in \mathbb{G}^N$.
6. Output $(\mathsf{st}, \mathbf{q})$.

$\underline{\mathsf{Answer}\left(d \in \mathbb{G}, \mathbf{x} \in \{0,1\}^N \subseteq \mathbb{Z}_p^N, \mathbf{q}\right) \to a \in \mathbb{G}}$

1. Parse the query $\mathbf{q}$ as $(q_1, \ldots, q_N) \in \mathbb{G}^N$.
2. Output $a \leftarrow \prod_{j \in [N]} q_j^{x_j} \in \mathbb{G}$.

$\underline{\mathsf{Reconstruct}\,(\mathsf{st}, a) \to \{0,1,\perp\}}$

1. Parse the state st as (i,d,r,t).
2. Set $m \leftarrow d^{-r} \cdot a \in \mathbb{G}$.
3. If $m = 1_\mathbb{G}$, output "0." If $m = h_i^t$, output "1." Otherwise, output $\perp$.

database record changes, the data owner can update the digest in either construction incrementally. For example, in the lattice based construction given an old digest $\mathbf{d} = \mathbf{A}\mathbf{x}$ and a new database $\mathbf{x}'$, the new digest is $\mathbf{d}' = \mathbf{d} + \mathbf{A}(\mathbf{x}' - \mathbf{x})$. Given the old digest, the server can compute the new digest in time proportional to the cost of computing $\mathbf{A}(\mathbf{x}' - \mathbf{x})$. This matrix-vector product, in turn, takes time linear in the number of updates to the database, i.e., the Hamming weight of the difference $\mathbf{x}' - \mathbf{x}$. If the database itself is public, any third party can verify that the new digest correctly incorporates these updates. The DDH-based construction supports a similar style of incremental updates. A frequently changing database, however, requires a client to obtain a fresh and correct digest before making each PIR query. One possible solution to this is to use a public log and a timestamping service [86, 89].

6 Implementation

We implemented all of our authenticated-PIR schemes in roughly 4k lines of Go and 45 lines of C. Our function-secret-sharing implementations are based on the Function Secret Sharing (FSS) Library [92]. Our Merkle-tree implementation is based on the `go-merkletree` library [87]. We implemented group operations in our single-server scheme from the DDH assumption with the CIRCL library [44]. The single-server scheme built on the LWE assumption uses a plaintext modulus of 2^{128} and relies on the `uint128` library [26].

We also implemented multi-server *unauthenticated*-PIR schemes as baselines for comparison. The multi-server unauthenticated-PIR scheme, also used in the authenticated-PIR scheme for point queries, is over the binary field and uses `fastxor` [25]. We use the original implementation of SimplePIR [52] as our single-server PIR baseline.

Our implementation is available under open-source license at `https://github.com/dedis/apir-code`.

6.1 Privacy-preserving key directory

To evaluate the practicality of authenticated PIR, we built Keyd, a PGP public-key directory service that offers (1) classic key look-ups and (2) computation of statistics over keys. A key-directory service maps human-memorable identifiers, such as email addresses, to cryptographic identities (public keys). Examples of such directories are the MIT PGP Public Key Server [69], along with the public-key directories that secure-messaging solutions, such as Signal, implicitly offer.

We implement Keyd in the two-server model, where the security properties hold as long as at least one server is honest. The Keyd key service provides the following properties:

- **Privacy**: The client reveals no information to the servers about the content of its query.
- **Integrity**: The client is guaranteed to recover the *correct* result for the issued query, i.e., the output of the protocol is consistent with the honest server's view.

Prior key-server designs ensure only one of these two properties. It is possible to add privacy to a key server using conventional PIR and issue private complex queries using Splinter [93], *or* to add integrity as in CONIKS [65]. Prior to authenticated PIR, we are unaware of any approach that simultaneously solves both problems in the presence of malicious servers, without resorting to trusted hardware [64].

Keyd lays out public keys in the database using a hash table that maps public keys into fixed-size buckets. To retrieve a PGP public key, a client hashes the requested email to determine the corresponding bucket number, queries the servers for the contents of the bucket, reconstructs and validates the answers, and finally selects and outputs the key of interest.

To evaluate a predicate query, the client sends the query to the servers, which apply it to the appropriate PGP key metadata. For example, to evaluate a `COUNT` query on the email addresses, the client sends `SELECT COUNT(*) FROM email WHERE email =` p, where p represents the query parameter hidden through secret sharing. The `AVG` query is implemented using a `SUM` and `COUNT` query. We use TLS to protect the communication between client and servers.

Our Keyd serves a snapshot of SKS PGP key directory [90] from 24 January 2021. We removed all public keys larger than

8 KiB, a limit that we found excluded only keys with large attachments, such as JPEG images. We also removed all keys that had been revoked, keys in an invalid format, and keys with no email address in their metadata. We kept only the primary key of each public key. If multiple keys were linked to the same email address, we kept only the most recent key. If a key included multiple emails, we indexed this key using the primary email. As a result, our Keyd serves a total of 3,557,164 unique PGP keys (≈3 GiB in total), which is more than half of the keys in the original dump.

7 Experimental evaluation

We experimentally evaluate all of our authenticated-PIR schemes and the Keyd public-key directory service.

Parameters. We instantiate our multi-server authenticated-PIR scheme for predicate queries using $\mathbb{F}_p^4$ with $p = 2^{32} - 1$, yielding a security parameter of approximately 124 bits. This approach is faster than using a full 128-bit field element, because of better-optimized libraries and CPU instructions for operating on 32-bit values. The Merkle-based scheme for point queries uses BLAKE3 as the hash function. The DDH-based single-server scheme (§5.2) uses the P256 elliptic curve as the group. We select the parameters for the LWE-based schemes (§5.1) to ensure 128-bit of privacy according to current estimate of concrete security against known attacks [3]. We present one scheme with integrity error 2^{-128}, and another one that uses integrity amplification (Section 3.3 and Construction 6), with integrity error 2^{-64}. The scheme with integrity error 2^{-128} uses modulus $q = 2^{128}$ and lattice dimension $n = 4800$; the scheme with integrity error 2^{-64} works with $q = 2^{32}$ and $n = 1100$. For both implementations, the error distribution is the discrete Gaussian distribution with standard deviation $\sigma = 6.4$. Integrity amplification uses the simple repetition code. We further discuss parameter selection for the scheme based on integrity amplification in Appendix H.

Experimental methodology. We perform all the experiments on machines equipped with two Intel Xeon E5-2680 v3 (Haswell) CPUs, each with 12 cores, 24 threads, and operating at 2.5 GHz. Each machine has 256 GB of RAM, and runs Ubuntu 20.04 and Go 1.17.5. Machines are connected with 10 Gigabit Ethernet. In the experiments for the multi-server schemes and Keyd (Sections 7.1, 7.2 and 7.4), the client and the servers run on separate machines. For single-server schemes we use a single machine that runs both client and server, as the single-server schemes are inherently sequential. We always report the time elapsed from query computation to record reconstruction as user time and the cumulative bandwidth from and to the server(s) as bandwidth. We execute all experiments 30 times and report the median result across executions. We run all the experiments using a single core for each physical machine. For consistency across experiments, we always download the same public-key when evaluating Keyd. We have published our experimental code and results in our source-code repository (see Section 6).

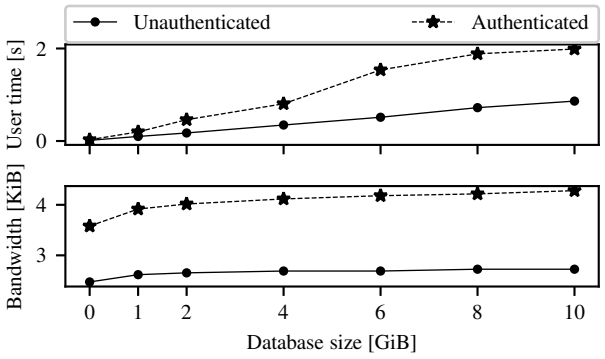

Figure 3: The cost of retrieving a 1 KiB record using classic ("Unauthenticated") and authenticated PIR for point queries (§4.1) from two servers. For this experiment we use a classic PIR scheme based on distributed point functions [17, 18, 48]. The Merkle proof attached to each record imposes the bandwidth and user time overheads.

7.1 Multi-server point queries

Fig. 3 presents user time and bandwidth overhead for our authenticated-PIR scheme for point queries, compared to classic unauthenticated PIR. Both the user time and the bandwidth overheads increase as the database size grows. This is due to each database record requiring an additional Merkle proof of size $O(\lambda \log N)$, which the client must fetch and verify. We measure a maximum overhead of $3\times$ for user time and of $1.6\times$ for bandwidth.

Fig. 4 shows the impact of the number of servers on user time and bandwidth. Since all servers answer in parallel, the user time increase is almost negligible. For authenticated PIR, the increase is due to Merkle proof verification. Bandwidth increases linearly for both schemes, since each server receives a query and sends an answer. The absolute bandwidth reported in Fig. 4 is significantly higher than that in Fig. 3, as the latter uses a state-of-the-art PIR scheme based on distributed point functions [17, 18, 48] as both unauthenticated scheme and the underlying PIR scheme for the authenticated version. Additionally, the database representation differs between the figures: Fig. 3 uses a vector representation, (one data block per database row), while Fig. 4 uses a matrix representation (with multiple blocks stored in a single row to trade off download complexity for query complexity).

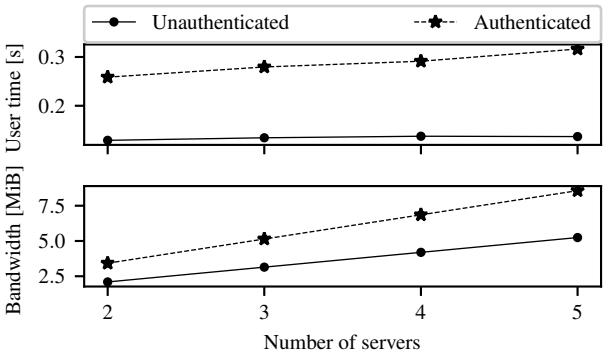

Figure 4: The cost of retrieving a 1 KiB record using unauthenticated and authenticated PIR for point queries (§4.1) from a variable number of servers holding a database of 1 GiB. For this experiment we use a classic PIR scheme based on classic secret-sharing over the binary field, as schemes based on distributed point functions impose a query bandwidth exponential in the number of servers and database length [17].

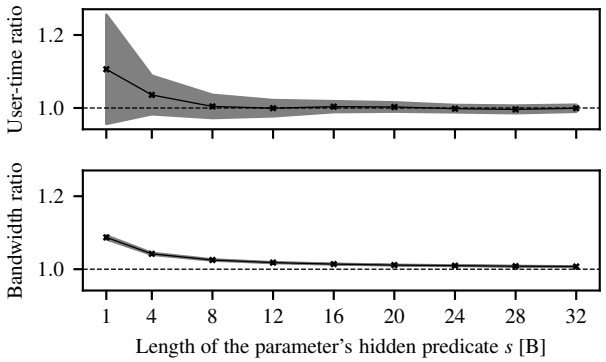

Figure 5: The user time and bandwidth ratios between unauthenticated and authenticated PIR (§4.2) for complex queries when querying two serves for the query `SELECT COUNT(*) FROM keys WHERE email LIKE "%s"` from a database composed of 100,000 random records. The median authentication overhead is less than 1.1× for both user time and bandwidth; the grey area shows the variance.

7.2 Multi-server complex queries

When comparing our multi-server authenticated-PIR scheme for complex queries with classic PIR (Fig. 5), we find that both the user time and bandwidth overheads of the authenticated scheme are less than 1.1×. The former comes from the longer output of the function-secret-sharing evaluation function—one $\mathbb{F}_{2^{31}-1}$ element versus five elements—and from the verification of the servers' answers, absent in the unauthenticated scheme. For bandwidth, the only difference is the so-called correction word in the function-secret-sharing key [17, 18], which is composed of a single field element in classic PIR and of five elements in authenticated PIR: one for the predicate evaluation's result and four for authentication. The servers' answers have the same ratio: a single field element in the unauthenticated scheme and five elements in the authenticated scheme. The bandwidth overhead is thus of a constant factor. Evaluation with $k \geq 3$ servers is infeasible as the length of the keys is $O(\lambda 2^{k/2} 2^{\ell/2})$, where ℓ is the input size in bits [17].

7.3 Single-server point queries

To evaluate our single-server authenticated-PIR schemes, we compare their performance against SimplePIR [52], the fastest classic single-server PIR scheme for small records to-date. We measure the costs of retrieving one data bit from the database.[1] We evaluate SimplePIR with its default configuration of 2048-bit database records. The client downloads a corresponding record and selects a desired bit from it. The offline bandwidth indicates the digest for authenticated schemes, and the hint for SimplePIR, as this scheme is a PIR-with-preprocessing scheme [10]. We show the results in Fig. 6.

The authenticated-PIR schemes from the decisional Diffie-Hellman assumption (DDH) and from the learning-with-errors assumption (LWE) have integrity error 2^{-128}. The DDH construction has a smaller digest, hence lower offline bandwidth, but has twice the online bandwidth of the LWE construction: both have the same asymptotic complexity, but LWE uses elements from $\mathbb{Z}_{2^{128}}$ and DDH from the elliptic curve P256, which encodes elements in 256 bits. The LWE construction is also faster (3-79×): arithmetic computations in $\mathbb{Z}_{2^{128}}$ are faster than elliptic-curve operations in P256.

The scheme with integrity amplification (LWE$^+$) has integrity error 2^{-64} and the same classic-PIR privacy as SimplePIR, except that SimplePIR does not provide privacy under selective-failure attacks. LWE$^+$ is faster than LWE for the 1 KiB and 1 MiB databases, but slower (1.4×) for the 1 GiB database: the repetition code requires repeating the protocol 15 times ($t = 7$). An error correcting code with higher rate, or parallel execution of the repetition code, could improve LWE$^+$. SimplePIR is 30-100× faster than LWE$^+$ due to its preprocessing for reducing online computation and exploiting a faster database representation through packing [52]. The asymptotic online and offline bandwidth overhead of SimplePIR and authenticated-PIR schemes from the LWE assumption are the same, but integrity amplification increases online bandwidth by $2t + 1\times$ (Section 3.3), whereas the client must download the digest only once. Concrete offline bandwidth is lower in SimplePIR due to database packing.

The current schemes are computationally costly, but we expect that future optimizations, such as multi-bit queries, as outlined in Appendix G.5, could reduce this cost.

[1] Other recent PIR schemes (e.g., [66, 71]) are competitive only in the large-record setting (where records are tens of kilobytes long).

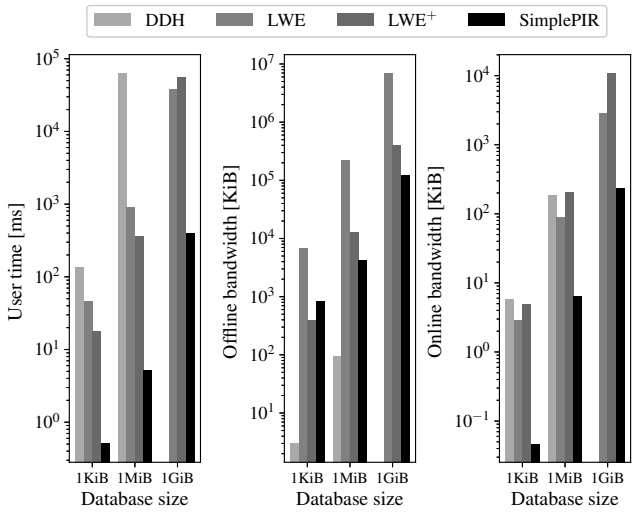

Figure 6: The cost of retrieving one data bit using our single-server authenticated PIR schemes and state-of-the-art classic single-server PIR scheme SimplePIR [52]. DDH indicates Construction 3 with 2^{-128}-integrity; LWE indicates Construction 2 ($q = 2^{128}$) with 2^{-128}-integrity; LWE$^+$ indicates Construction 6 (the base scheme is Construction 2 with $q = 2^{32}$) with 2^{-64}-integrity (see Section 3.3). DDH takes over an hour to retrieve a data bit from a 1 GiB database and we omit it from the figure.

7.4 Application: privacy-preserving key server

In this section, we evaluate our multi-server authenticated-PIR schemes in the context of the Keyd public-key server.

For classic key look-ups, which are point queries, we measure the wall-clock time needed to retrieve a PGP public-key with authenticated PIR (Section 4.1), classic PIR without authentication, and by direct download without privacy protection. To measure the latency of direct download, we download a PGP public-key from the OpenPGP key server using `wget`. Both PIR measurements include a manually-added RTT of 0.4 ms (the ping time to the nearest PGP key server). We perform all the measurements over the entire processed dataset of PGP keys (see Section 6). We measure 1.11 seconds for authenticated PIR, 1.10 seconds for unauthenticated PIR and 0.22 seconds for non-private direct look-up.

The authenticated scheme for point queries shows performance comparable to classic PIR without authentication. The Merkle-proof overhead in this case is smaller than in Fig. 3 due to a larger block size and hence less authentication data per data bit in Keyd. The OpenPGP key server maintainers informed us that their service typically handles around 3–10 public-key lookups per second, or less than 1 million requests per day [21]. A careful multithreaded implementation of our multi-server authenticated-PIR schemes for point queries can handle this load with 12 cores, just one more than the number of cores estimated for classic unauthenticated PIR (11 cores).

Query description	User time [s]			Bandwidth [KiB]		
	Unauth.	Auth.		Unauth.	Auth.	
`COUNT(*) WHERE email LIKE '%.edu'`	25.77	25.97	1.01×	1.8	1.9	1.06×
`type = 'ElGamal'`	7.52	7.66	1.02×	0.9	1.0	1.11×
`YEAR(created) = 2019 AND email LIKE '%.edu'`	48.28	48.32	1.00×	3.0	3.1	1.03×
`AVG(lifetime) WHERE email LIKE '%.edu'`	25.74	26.59	1.03×	1.8	1.9	1.05×

Table 7: Performance of different predicate queries on Keyd for unauthenticated and authenticated PIR (the two-server schemes for predicate queries). The median authentication overhead is 1.01× for user time and 1.05× for bandwidth.

To analyze the performance of Keyd in computing private statistics over keys, we measure user-perceived time and bandwidth of different predicate queries. Table 7 shows the results. For all the predicates, the overhead of authenticated PIR—in both user-perceived time and bandwidth—is upper bounded by a factor of 1.05×. This result matches the benchmark presented in Fig. 5 and is due to the latency being dominated by the function-secret-sharing evaluation, which is essentially equal for authenticated and unauthenticated PIR. For bandwidth overhead, the same reasoning as in Section 7.2 applies.

8 Related work

Authenticated PIR builds on diverse work on private information retrieval. Starting with the original proposal [31], improvements have reduced the communication cost of multi-server PIR with information-theoretic [8, 9, 42, 96, 98] or computational security [18, 30]. Kushilevitz and Ostrovsky [57] presented the first single-server PIR construction, and subsequent work reduced communication costs [22, 41, 47, 62, 75]. Recent advances introduced PIR for more complex (e.g., SQL-like) queries [74, 81, 93].

Kushilevitz and Ostrovsky [57] first noted that, in the single-server setting, the server could violate a client's privacy by manipulating database records and observing whether the client accepted the response as valid. Such attacks have come to be known as *selective-failure attacks* [53, 55, 61]. To our knowledge, we are the first to address selective-failure attacks in the multi-server setting.

In schemes that resist faulty servers (summarized in Table 2), a client can either *reconstruct* the correct database entry, or can *detect and abort*, when servers misbehave. Multiparty computation literature refers to the former approach as "full security" and the latter as "security with abort" [50].

Beimel and Stahl [11, 12] first consider malicious or crashing servers in the multi-server setting. Their approach focuses on ensuring data reconstruction, not detection of server misbehaviour, and it is further developed by concurrent and follow-up work [39, 43, 49, 56, 97]. Unlike authenticated PIR,

these approaches require an honest majority in the presence of malicious servers, with specific thresholds shown in Table 2.

Verifiable PIR in the multi-server setting [99] offers security properties similar to authenticated PIR, but requires expensive public-key cryptography. In the single-server setting [94, 100], verifiable PIR is not resistant to selective-failure attacks and offers a weaker property: it ensures that the server answer a query with respect to *some* database, but not necessarily the one intended. Our approach ensures that queries are answered with respect to a *specific* database, as determined by the honest server in the multi-server setting, or by the database digest in the single-server case. In concurrent work, Ben-David et al. [14] introduce another notion of verifiable PIR in the single-server setting, whose goal is to verify arbitrary properties on databases, but they do not consider selective-failure attacks.

Our multi-server scheme for point queries (Section 4.1) extends a Merkle-tree approach by Kushilevitz and Ostrovsky [57]. Our multi-server scheme for predicate queries builds on function secret-sharing [16, 17, 18, 38], information-theoretic message authentication codes [33], and malicious-secure multiparty computation protocols [15, 35].

Prior systems address integrity in private information retrieval [36, 70], but do not protect against selective manipulation in the single-server setting, and require additional assumptions in the multi-server setting.

Prior work has also considered privacy-preserving and integrity-assuring key directories [27, 28, 65, 68, 91]. In particular, CONIKS [65] and its improved version SEEMless [27], ensure consistency for the bindings thanks to ideas adapted from transparency log systems [59, 83], but do not address privacy of the client's queries.

9 Conclusion

Authenticated PIR enhances the strong privacy properties of classic PIR with strong data-authentication guarantees. We have presented formal definitions both in the dishonest-majority setting—where the security properties hold as long as at least one of the server is honest—and in the single-server setting. We suggest some avenues for further improvement:

- Can we construct single-server authenticated-PIR schemes for a *malicious* digest (i.e., the client's output is consistent with *some* n-bit database)?
- Can we construct single-server authenticated-PIR schemes whose performance matches that of the best unauthenticated schemes?

Acknowledgements. Brett Falk, Pratyush Mishra, and Matan Shtepel pointed out a flaw in the original proof of Theorem 28. We thank David Lazar for the initial discussions on selective-failure attacks against PIR protocols. We thank the anonymous reviewers of IEEE S&P and USENIX Security for their thoughtful comments and suggestions on how to improve this work. We thank Vincent Breitmoser for sharing with us the statistics of the `keys.openpgp.org` key server. We are grateful to Serge Vaudenay, Dima Kogan, Sylvain Chatel, Khashayar Barooti and Christian Mouchet for helpful conversations and feedback. We thank Pierluca Borsò-Tan and Noémien Kocher for help with the experiments and with the implementation of Keyd's clients. We thank Jean Viaene for help with the artifact evaluation. This work was supported in part by the AXA Research Fund, Handshake, ETH4D Humanitarian Action Challenges project PAIDIT, US Office of Naval Research (ONR) grant N000141912361, EU grant 825377, Swiss Reinsurance Corporation, DARPA Contract FA8750-19-C-0079, gifts from Google and Mozilla, a Facebook Research Award, MIT's Fintech@CSAIL Initiative, NSF CNS-2054869, CNS-2151131, CNS-2140975, a Microsoft Research Faculty Fellowship, and a Google Research Scholar award.

References

[1] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR: Private information retrieval for everyone. *PoPETs*, 2016.

[2] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, 1996.

[3] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 2015.

[4] Asra Ali, Tancrede Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. Communication–computation trade-offs in PIR. In *USENIX Security*, 2021.

[5] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *S&P*, 2018.

[6] Sebastian Angel and Srinath T. V. Setty. Unobservable Communication over Fully Untrusted Infrastructure. In *OSDI*, 2016.

[7] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. DNS Security Introduction and Requirements. RFC 4003, 2005.

[8] Amos Beimel and Yuval Ishai. Information-Theoretic Private Information Retrieval: A Unified Construction. In *ICALP*, 2001.

[9] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Jean-François Raymond. Breaking the $O(n^{1/(2k-1)})$ Barrier for Information-Theoretic Private Information Retrieval. In *FOCS*, 2002.

[10] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the Servers' Computation in Private Information Retrieval: PIR with Preprocessing. *J. Cryptol.*, 2004.

[11] Amos Beimel and Yoav Stahl. Robust Information-Theoretic Private Information Retrieval. In *SCN*, 2002.

[12] Amos Beimel and Yoav Stahl. Robust Information-

Theoretic Private Information Retrieval. *J. Cryptol.*, 2007.

[13] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, 1993.

[14] Shany Ben-David, Yael Tauman Kalai, and Omer Paneth. Verifiable Private Information Retrieval. In *TCC*, 2022.

[15] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic Encryption and Multiparty Computation. In *EUROCRYPT*, 2011.

[16] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *EUROCRYPT*, 2021.

[17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *EUROCRYPT*, 2015.

[18] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *CCS*, 2016.

[19] Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In *TCC*, 2017.

[20] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC*, 2013.

[21] Vincent Breitmoser. Private communication, 2021.

[22] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally Private Information Retrieval with Polylogarithmic Communication. In *EUROCRYPT*, 1999.

[23] Jan Camenisch, Gregory Neven, and Abhi Shelat. Simulatable adaptive oblivious transfer. In *EUROCRYPT*, 2007.

[24] Justin Cappos. Avoiding theoretical optimality to efficiently and privately retrieve security updates. In *FC*, 2013.

[25] Luke Champine. fastxor. `https://github.com/lukechampine/fastxor`, 2018.

[26] Luke Champine. uint128 for Go. `https://github.com/lukechampine/uint128`, 2022.

[27] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. SEEMless: Secure End-to-End Encrypted Messaging with less Trust. In *CCS*, 2019.

[28] Brian Chen, Yevgeniy Dodis, Esha Ghosh, Eli Goldin, Balachandar Kesavan, Antonio Marcedone, and Merry Ember Mou. Rotatable Zero Knowledge Sets: Post Compromise Secure Auditable Dictionaries with application to Key Transparency. In *ASIACRYPT*, 2022.

[29] Benny Chor, Nic Gilboa, and Mori Naor. Private Information Retrieval by Keywords. Cryptology ePrint Archive, Paper 1998/003, 1998.

[30] Benny Chor and Niv Gilboa. Computationally Private Information Retrieval (Extended Abstract). In *STOC*, 1997.

[31] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private Information Retrieval. In *FOCS*, 1995.

[32] Henry Corrigan-Gibbs and Dmitry Kogan. Private Information Retrieval with Sublinear Online Time. In *EUROCRYPT*, 2020.

[33] Ronald Cramer, Yevgeniy Dodis, Serge Fehr, Carles Padró, and Daniel Wichs. Detection of Algebraic Manipulation with Applications to Robust Secret Sharing and Fuzzy Extractors. In *EUROCRYPT*, 2008.

[34] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single Database Private Information Retrieval Implies Oblivious Transfer. In *EUROCRYPT*, 2000.

[35] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO*, 2012.

[36] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. DORY: An Encrypted Search System with Distributed Trust. In *OSDI*, 2020.

[37] Alex Davidson, Gonçalo Pestana, and Sofía Celi. FrodoPIR: Simple, Scalable, Single-Server Private Information Retrieval. In *PoPETs*, 2023.

[38] Leo de Castro and Antigoni Polychroniadou. Lightweight, Maliciously Secure Verifiable Function Secret Sharing. In *EUROCRYPT*, 2022.

[39] Casey Devet, Ian Goldberg, and Nadia Heninger. Optimally Robust Private Information Retrieval. In *USENIX Security*, 2012.

[40] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-Malleable Cryptography (Extended Abstract). In *STOC*, 1991.

[41] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor Hash Functions and Their Applications. In *CRYPTO*, 2019.

[42] Zeev Dvir and Sivakanth Gopi. 2-Server PIR with subpolynomial communication. *J. ACM*, 2016.

[43] Reo Eriguchi, Kaoru Kurosawa, and Koji Nuida. Multi-Server PIR with Full Error Detection and Limited Error Correction. In *ITC*, 2022.

[44] Armando Faz-Hernández and Kris Kwiatkowski. Introducing CIRCL: An advanced cryptographic library. `https://github.com/cloudflare/circl`, 2019.

[45] Marc Fischlin and Dominique Schröder. Security of blind signatures under aborts. In *PKC*, 2009.

[46] Craig Gentry and Shai Halevi. Compressible FHE with applications to PIR. In *TCC*, 2019.

[47] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communi-

cation rate. In *ICALP*, 2005.

[48] Niv Gilboa and Yuval Ishai. Distributed Point Functions and Their Applications. In *EUROCRYPT*, 2014.

[49] Ian Goldberg. Improving the Robustness of Private Information Retrieval. In *S&P*, 2007.

[50] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *STOC*, 1987.

[51] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath T. V. Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and Private Media Consumption with Popcorn. In *NSDI*, 2016.

[52] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In *USENIX Security*, 2023.

[53] Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *CRYPTO*, 2013.

[54] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *TCC*, 2009.

[55] Mehmet S. Kiraz and Berry Schoenmakers. A Protocol Issue for the Malicious Case of Yao's Garbled Circuit Construction. In *SITB*, 2006.

[56] Kaoru Kurosawa. How to Correct Errors in Multi-server PIR. In *ASIACRYPT*, 2019.

[57] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval. In *FOCS*, 1997.

[58] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle: An efficient communication system with strong anonymity. *PoPETs*, 2016.

[59] Ben Laurie. Certificate transparency. *Commun. ACM*, 2014.

[60] Lucy Li, Bijeeta Pal, Junade Ali, Nick Sullivan, Rahul Chatterjee, and Thomas Ristenpart. Protocols for checking compromised credentials. In *CCS*, 2019.

[61] Yehuda Lindell and Benny Pinkas. Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer. In *TCC*, 2011.

[62] Helger Lipmaa. An Oblivious Transfer Protocol with Log-Squared Communication. In *ISC*, 2005.

[63] Wouter Lueks and Ian Goldberg. Sublinear Scaling for Multi-Client Private Information Retrieval. In *FC*, 2015.

[64] Moxie Marlinspike. Private contact discovery. `https://signal.org/blog/private-contact-discovery/`, September 26, 2017. Accessed 11 April 2021.

[65] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: Bringing Key Transparency to End Users. In *USENIX Security*, 2015.

[66] Samir Jordan Menon and David J. Wu. SPIRAL: fast, high-rate single-server PIR via FHE composition. In *S&P*, 2022.

[67] Ralph C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In *CRYPTO*, 1987.

[68] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. Oblix: An efficient oblivious search index. In *S&P*, 2018.

[69] MIT PGP public key server. `https://pgp.mit.edu/`. Accessed 9 April 2021.

[70] Prateek Mittal, Femi G Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: Scalable anonymous communication using private information retrieval. In *USENIX Security*, 2011.

[71] Muhammad Haris Mughees, Hao Chen, and Ling Ren. OnionPIR: Response efficient single-server PIR. In *CCS*, 2021.

[72] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *FOCS*, 1997.

[73] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds. In *USENIX Security*, 2017.

[74] Femi G. Olumofin and Ian Goldberg. Privacy-preserving queries over relational databases. In *PoPETs*, 2010.

[75] Rafail Ostrovsky and William E Skeith. A survey of single-database private information retrieval: Techniques and applications. In *PKC*, 2007.

[76] Jeongeun Park and Mehdi Tibouchi. SHECS-PIR: somewhat homomorphic encryption-based compact and scalable private information retrieval. In *ESORICS*, 2020.

[77] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *S&P*, 2013.

[78] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Private stateful information retrieval. In *CCS*, 2018.

[79] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1991.

[80] John M. Pollard. Kangaroos, monopoly and discrete logarithms. *J. Cryptol.*, 2000.

[81] Joel Reardon, Jeffrey Pound, and Ian Goldberg. Relational-complete private information retrieval. *Technical Report CACR*, 2007.

[82] Oded Regev. On lattices, learning with errors, random

linear codes, and cryptography. In *STOC*, 2005.

[83] Mark Dermot Ryan. Enhanced Certificate Transparency and End-to-End Encrypted Mail. In *NDSS*, 2014.

[84] Matan Shtepel, Pratyush Mishra, and Brett Falk. Private communication, 2024.

[85] Markus Stadler. Publicly Verifiable Secret Sharing. In *EUROCRYPT*, 1996.

[86] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. In *S&P*, 2016.

[87] Weald Technology. go-merkletree. `https://github.com/wealdtech/go-merkletree`, 2019.

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

[89] Alin Tomescu and Srinivas Devadas. Catena: Efficient non-equivocation via Bitcoin. In *S&P*, 2017.

[90] Carles Tubio. SKS dump. `https://pgp.key-server.io/sks-dump`, 2021. Accessed 7 April 2021.

[91] Nirvan Tyagi, Ben Fisch, Joseph Bonneau, and Stefano Tessaro. Client-auditable verifiable registries. Cryptology ePrint Archive, 2021.

[92] Frank Wang. Function Secret Sharing (FSS) Library. `https://github.com/frankw2/libfss`, 2017.

[93] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *NSDI*, 2017.

[94] Xingfeng Wang and Liang Zhao. Verifiable Single-Server Private Information Retrieval. In *ICICS*, 2018.

[95] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Scalable anonymous group communication in the anytrust model. In *EuroSec*, 2012.

[96] David Woodruff and Sergey Yekhanin. A geometric approach to information-theoretic private information retrieval. In *CCC*, 2005.

[97] Erica Y. Yang, Jie Xu, and Keith H. Bennett. Private Information Retrieval in the Presence of Malicious Failures. In *COMPSAC*, 2002.

[98] Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. In *STOC*, 2007.

[99] Liang Feng Zhang and Reihaneh Safavi-Naini. Verifiable Multi-server Private Information Retrieval. In *ACNS*, 2014.

[100] Liang Zhao, Xingfeng Wang, and Xinyi Huang. Verifiable single-server private information retrieval from LWE with binary errors. *Inf. Sci.*, 2021.

A Building blocks for authenticated PIR

In this section, we formally introduce the primitives used by the different authenticated-PIR schemes: classic multi-server PIR, Merkle-tree and function secret sharing.

Additional notation. We use $\mathsf{SD}(\cdot,\cdot)$ to denote the statistical distance between two distributions. The empty string is denoted with ε. We write $\mathcal{D}_0 \approx_c \mathcal{D}_1$ to denote that the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are computational indistinguishable.

A.1 Classic multi-server PIR for point queries

In this section we define standard k-server unauthenticated-PIR schemes, for $k \geq 2$.

Definition 12 (k-server PIR for point queries)**.** *A k-server* unauthenticated-PIR *scheme for point queries parametrized by a database length $N \in \mathbb{N}$, consists of three efficient, and possibly randomized, algorithms:*

- $\mathsf{Query}(1^\lambda, i) \to (\mathsf{st}, q_1, \ldots, q_k)$. *Given a security parameter λ, expressed in unary, and an index $i \in [N]$, return client state* st *and queries $q_1, \ldots, q_k$.*
- $\mathsf{Answer}(\mathbf{x}, q) \to a$. *Apply query q to database $\mathbf{x} \in \{0,1\}^N$ and return answer a.*
- $\mathsf{Reconstruct}(\mathsf{st}, a_1, \ldots, a_k) \to x_i$. *Take as input client state* st *and answers $a_1, \ldots, a_k$ and return the i^{th} record of the database x_i.*

A k-server unauthenticated-PIR scheme is required to satisfy the following properties.

Definition 13 (PIR correctness)**.** *An unauthenticated-PIR scheme* $\mathsf{PIR} = (\mathsf{PIR.Query}, \mathsf{PIR.Answer}, \mathsf{PIR.Reconstruct})$*, parametrized by a number of servers $k \in \mathbb{N}$ and a database size $N \in \mathbb{N}$ satisfies* correctness *if for every $\mathbf{x} \in \{0,1\}^N$, the following holds:*

$$\Pr\left[x_i' = x_i : \begin{array}{c}(\mathsf{st}, \{q_i\}_{i\in[k]}) \leftarrow \mathsf{PIR.Query}(i) \\ a_j \leftarrow \mathsf{PIR.Answer}(\mathbf{x}, q_j) \quad \forall j \in [k] \\ x_i' \leftarrow \mathsf{PIR.Reconstruct}(\mathsf{st}, a_1, \ldots, a_k)\end{array}\right] = 1,$$

where the probability is computed over all the random coins used by the algorithms of the scheme.

Definition 14 (PIR security)**.** *Let* $\mathsf{PIR} = (\mathsf{PIR.Query}, \mathsf{PIR.Answer}, \mathsf{PIR.Reconstruct})$ *be an unauthenticated-PIR scheme for point queries parametrized by a number of servers $k \in \mathbb{N}$ and a database size $N \in \mathbb{N}$. Let S be any subset of $k-1$ elements from $[k]$. For $i \in [N]$ let the distribution*

$$\mathrm{REAL}_i = \left\{ \bigcup_{j\in S} q_j : (\mathsf{st}, q_1, \ldots, q_k) \leftarrow \mathsf{PIR.Query}(i) \right\}.$$

Similarly, for a simulator $\mathcal{S}$, let the distribution

$$\mathrm{IDEAL}_{\mathcal{S}} = \left\{ \{q_j\}_{j\in S} \leftarrow \mathcal{S} \right\}.$$

A classic unauthenticated-PIR scheme $\mathsf{PIR} = (\mathsf{PIR.Query}, \mathsf{PIR.Answer}, \mathsf{PIR.Reconstruct})$ *parametrized by a database length* $N \in \mathbb{N}$ *and a number of servers* $k \in \mathbb{N}$ *is* secure *if for every* $i \in [N]$*, the following holds:*

$$\mathrm{REAL}_i \approx_c \mathrm{IDEAL}_{\mathcal{S}}.$$

In this work, we consider only *linear* classic PIR schemes. Many standard PIR schemes are linear [18, 31, 32, 48].

Definition 15 (Linear PIR)**.** *Let* $\mathsf{PIR} = (\mathsf{PIR.Query}, \mathsf{PIR.Answer}, \mathsf{PIR.Reconstruct})$ *be a classic PIR scheme for point queries parametrized by a number of servers* $k \in \mathbb{N}$ *and a database size* $N \in \mathbb{N}$*. We say that* PIR *is a* linear *PIR scheme if the* Reconstruct *algorithm is simply the sum of the individual severs' answers.*

A.2 Merkle tree

In this section we formally define a Merkle-tree scheme and we introduce its security properties.

Definition 16. *A* Merkle-tree scheme $\mathsf{M} = (\mathsf{Digest}, \mathsf{ProveIncludes}, \mathsf{VerifyIncludes})$*, parametrized by a digest length* $\ell_{\mathsf{dig}} \in \mathbb{N}$ *and a inclusion proof length* $\ell_\pi \in \mathbb{N}$*, for a database* $\mathbf{x} \in \{0,1\}^N$*,* $N \in \mathbb{N}$*, consists of two possibly randomized algorithms and one deterministic algorithm:*

- $\mathsf{Digest}(1^\lambda, \mathbf{x}) \to d$*. Given a security parameter* λ*, expressed in unary, and a database* $x \in \{0,1\}^N$*, returns a database digest* $d \in \{0,1\}^{\ell_{\mathsf{dig}}}$*.*
- $\mathsf{ProveIncludes}(1^\lambda, \mathbf{x}, i, x_i) \to \{\pi_i, \perp\}$*. This deterministic algorithm, on input a security parameter* λ *expressed in unary, a database* $\mathbf{x} \in \{0,1\}^N$*, a index* $i \in [N]$ *and a database record* $x_i \in \{0,1\}$*, outputs a unique proof* $\pi_i \in \{0,1\}^{\ell_\pi}$ *if* $x_i \in \mathbf{x}$ *and* $\perp$ *otherwise.*
- $\mathsf{VerifyIncludes}(d, i, x_i, \pi_i) \to \{0,1\}$*. Given a digest* $d \in \{0,1\}^{\ell_{\mathsf{dig}}}$*, a index* $i \in [N]$*, a database entry* $x_i \in \{0,1\}$ *and a proof* $\pi_i \in \{0,1\}^{\ell_\pi}$*, outputs* 1 *if* π_i *proves that the database represented by the digest* d *contains the record* x_i *at position* i *and* 0 *otherwise.*

A Merkle-tree scheme defined in Definition 16 is required to satisfy the following properties.

Definition 17 (Merkle tree correctness)**.** *Let* $\mathsf{M} = (\mathsf{Digest}, \mathsf{ProveIncludes}, \mathsf{VerifyIncludes})$ *be a Merkle-tree scheme as defined in Definition 16, parametrized by a digest length* $\ell_{\mathsf{dig}} \in \mathbb{N}$ *and a inclusion proof length* $\ell_\pi \in \mathbb{N}$*, for a database* $\mathbf{x} \in \{0,1\}^N$*,* $N \in \mathbb{N}$*. We say that* M *satisfies* correctness *if, for all* $i \in [N]$*, the following holds:*

$$\Pr\left[b = 1 : \begin{array}{l} d \leftarrow \mathsf{Digest}(\mathbf{x}) \\ \pi \leftarrow \mathsf{ProveIncludes}(\mathbf{x}, i, x_i) \\ b \leftarrow \mathsf{VerifyIncludes}(d, i, x_i, \pi) \end{array}\right] = 1.$$

Definition 18 (Merkle tree uniqueness)**.** *Let* $\mathsf{M} = (\mathsf{Digest}, \mathsf{ProveIncludes}, \mathsf{VerifyIncludes})$ *be a Merkle-tree scheme as defined in Definition 16, parametrized by a digest length* $\ell_{\mathsf{dig}} \in \mathbb{N}$ *and a inclusion proof length* $\ell_\pi \in \mathbb{N}$*, for a database* $\mathbf{x} \in \{0,1\}^N$*,* $N \in \mathbb{N}$*. Let* $\mathcal{A}$ *be an efficient adversary.* M *ensures* uniqueness *if the following holds:*

$$\Pr\left[b = b' = 1 : \begin{array}{l} (\mathbf{x}, i, x_i, \pi_i, \pi_i') \leftarrow \mathcal{A}(1^\lambda, N) \\ \text{if } \pi_i = \pi_i' \text{ then abort} \\ d \leftarrow \mathsf{Digest}(\mathbf{x}) \\ b \leftarrow \mathsf{VerifyIncludes}(d, i, x_i, \pi_i) \\ b' \leftarrow \mathsf{VerifyIncludes}(d, i, x_i, \pi_i') \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Definition 19 (Merkle tree soundness)**.** *Let* $\mathsf{M} = (\mathsf{Digest}, \mathsf{ProveIncludes}, \mathsf{VerifyIncludes})$ *be a Merkle-tree scheme as defined in Definition 16, parametrized by a digest length* $\ell_{\mathsf{dig}} \in \mathbb{N}$ *and a inclusion proof length* $\ell_\pi \in \mathbb{N}$*, for a database* $\mathbf{x} \in \{0,1\}^N$*,* $N \in \mathbb{N}$*. Let* $\mathcal{A}$ *be an efficient adversary.* M *satisfies* soundness *if the following holds:*

$$\Pr\left[b = 1 : \begin{array}{l} (\mathbf{x}, i, x_i^*, \pi_i) \leftarrow \mathcal{A}(1^\lambda, N) \\ \text{if } x_i = x_i^* \text{ then abort} \\ d \leftarrow \mathsf{Digest}(\mathbf{x}) \\ b \leftarrow \mathsf{VerifyIncludes}(d, i, x_i^*, \pi_i) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

A.3 Function secret sharing

In this section we formally define the properties of function-secret-sharing (FSS) schemes [17, 18]. We present the syntax in Section 4.2.

Definition 20 (FSS correctness)**.** *A k-party function secret-sharing scheme* $\mathsf{FSS} = (\mathsf{Gen}, \mathsf{Eval})$ *for a function class* $\mathcal{F}$ *defined over a field* $\mathbb{F}$ *satisfies* correctness *if for every* x *in the domain of* f*, the following holds:*

$$\Pr\left[\sum_{i \in [k]} \mathsf{Eval}(f_i, x) = f(x) \in \mathbb{F} : (f_1, \ldots, f_k) \leftarrow \mathsf{Gen}(1^\lambda, f)\right] = 1.$$

Definition 21 (FSS security)**.** *Let* $\mathsf{FSS} = (\mathsf{Gen}, \mathsf{Eval})$ *be a k-party function secret-sharing scheme for a function class* $\mathcal{F}$*. Let* S *be any subset of* $k-1$ *elements from* $[k]$*. For a security parameter* $\lambda \in \mathbb{N}$ *and a function* $f \in \mathcal{F}$ *let the distribution*

$$\mathrm{REAL}_{\lambda,f} = \left\{\bigcup_{i \in S} f_i : (f_1, \ldots, f_k) \leftarrow \mathsf{Gen}(1^\lambda, f)\right\}.$$

Similarly, for a simulator $\mathcal{S}$ *let the distribution*

$$\mathrm{IDEAL}_{\mathcal{S},\lambda,\mathcal{F}} = \left\{\{f_i\}_{i \in S} \leftarrow \mathcal{S}(1^\lambda, \mathcal{F})\right\}$$

A k-party function secret-sharing scheme $\mathsf{FSS} = (\mathsf{Gen}, \mathsf{Eval})$ *for a function class* $\mathcal{F}$ *is* secure *if there exists a simulator* $\mathcal{S}$ *such that for every security parameter* $\lambda \in \mathbb{N}$ *and every function* $f \in \mathcal{F}$*, the following holds:*

$$\mathrm{REAL}_{\lambda,f} \approx_c \mathrm{IDEAL}_{\mathcal{S},\lambda,\mathcal{F}}.$$

B Multi-server authenticated PIR definitions

In this section, we present the formal definitions for multi-server authenticated PIR.

Definition 22 (Authenticated PIR correctness)**.** *A k-server authenticated-PIR scheme* $\Pi = (\mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ *for function class* $\mathcal{F} \subseteq \mathsf{Funs}[[N] \times \{0,1\}^\ell, \mathbb{F}]$ *and database size* $N \in \mathbb{N}$ *satisfies* correctness *if for every* $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_N \in \{0,1\}^\ell$, $\ell \in \mathbb{N}$, $\mathbf{w} \in \mathbb{F}^N$, $\lambda \in \mathbb{N}$, $f \in \mathcal{F}$, *the following holds:*

$$\Pr\left[\begin{array}{l} y = \sum_{i\in[n]} w_i f(i, \mathbf{x}_i) : \\ \quad (\mathsf{st}, q_1, \ldots, q_k) \leftarrow \mathsf{Query}(1^\lambda, f) \\ \qquad\qquad a_j \leftarrow \mathsf{Answer}(\mathbf{X}, \mathbf{w}, q_j) \quad \forall j \in [k] \\ \qquad\qquad y \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a_1, \ldots, a_k) \end{array}\right] = 1,$$

where the probability is computed over all the random coins used by the algorithms of the scheme.

Definition 23 (Authenticated PIR integrity)**.** *A k-server authenticated-PIR scheme* $\Pi = (\mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ *for function class* $\mathcal{F} \subseteq \mathsf{Funs}[[N] \times \{0,1\}^\ell, \mathbb{F}]$ *and database size* $N \in \mathbb{N}$ *preserves* integrity with error ε *if for every efficient adversary* $\mathcal{A}$*, and for every* $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_N \in \{0,1\}^\ell$, $\ell \in \mathbb{N}$, $\mathbf{w} \in \mathbb{F}^N$, $\lambda \in \mathbb{N}$, $f \in \mathcal{F}$, $j_{\mathsf{good}} \in [k]$*, the following probability is negligible in the security parameter* λ*:*

$$\Pr\left[\begin{array}{l} y \notin \left\{\sum_{i\in[N]} w_i f(i, \mathbf{x}_i), \perp\right\} : \\ \quad (\mathsf{st}, q_1, \ldots, q_k) \leftarrow \mathsf{Query}(1^\lambda, f) \\ \quad \{a_j\}_{j\neq j_{\mathsf{good}}} \leftarrow \mathcal{A}(\mathbf{X}, \mathbf{w}, \{q_j\}_{j\neq j_{\mathsf{good}}}) \\ \quad a_{j_{\mathsf{good}}} \leftarrow \mathsf{Answer}(\mathbf{X}, \mathbf{w}, q_{j_{\mathsf{good}}}) \\ \quad y \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a_1, \ldots, a_k) \end{array}\right] \leq \varepsilon,$$

where the probability is computed over all the random coins used by the algorithms of the scheme.

Definition 24 (Authenticated PIR privacy)**.** *Let* $\Pi = (\mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ *be a k-server authenticated-PIR scheme for function class* $\mathcal{F} \subseteq \mathsf{Funs}[[N] \times \{0,1\}^\ell, \mathbb{F}]$ *and database size* $N \in \mathbb{N}$*. For* $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_N \in \{0,1\}^\ell$, $\ell \in \mathbb{N}$, $\mathbf{w} \in \mathbb{F}^N$, $\lambda \in \mathbb{N}$, $f \in \mathcal{F}$, $j_{\mathsf{good}} \in [k]$*, and an adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*, define the distribution*

$$\mathrm{REAL}_{\mathcal{A}, j_{\mathsf{good}}, f, \lambda, \mathbf{X}, \mathbf{w}} = \left\{ \hat{\beta} : \begin{array}{l} (\mathsf{st}, q_1, \ldots, q_k) \leftarrow \mathsf{Query}(1^\lambda, f) \\ a_{j_{\mathsf{good}}} \leftarrow \mathsf{Answer}(\mathbf{X}, \mathbf{w}, q_{j_{\mathsf{good}}}) \\ (\mathsf{st}_{\mathcal{A}}, \{a_j\}_{j\neq j_{\mathsf{good}}}) \leftarrow \mathcal{A}_0(\mathbf{X}, \mathbf{w}, \{q_j\}_{j\neq j_{\mathsf{good}}}) \\ y \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a_1, \ldots, a_k) \\ b \leftarrow \mathbb{1}\{y \neq \perp\} \\ \hat{\beta} \leftarrow \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, b) \end{array} \right\}.$$

Similarly, for a simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$*, define the distribution*

$$\mathrm{IDEAL}_{\mathcal{A}, \mathcal{S}, \mathcal{F}, \lambda, \mathbf{X}, \mathbf{w}} = \left\{ \beta : \begin{array}{l} (\mathsf{st}_{\mathcal{S}}, Q) \leftarrow \mathcal{S}_0(1^\lambda, \mathcal{F}, \mathbf{X}, \mathbf{w}) \\ (\mathsf{st}_{\mathcal{A}}, A) \leftarrow \mathcal{A}_0(\mathbf{X}, \mathbf{w}, Q) \\ b \leftarrow \mathcal{S}_1(\mathsf{st}_{\mathcal{S}}, A) \\ \beta \leftarrow \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, b) \end{array} \right\}.$$

We say Π *is* private *if for every efficient adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*, and for every* $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N) \in (\{0,1\}^\ell)^N$, $\mathbf{w} \in \mathbb{F}^N$*, there exists a simulator* $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ *such that for all* $\lambda \in \mathbb{N}$*,* $f \in \mathcal{F}$*,* $j_{\mathsf{good}} \in [k]$*, the following holds:*

$$\mathrm{REAL}_{\mathcal{A}, j_{\mathsf{good}}, f, \lambda, \mathbf{X}, \mathbf{w}} \approx_c \mathrm{IDEAL}_{\mathcal{A}, \mathcal{S}, \mathcal{F}, \lambda, \mathbf{X}, \mathbf{w}}$$

Remark 25 (Selective-failure attacks)**.** The inclusion of the acceptance bit in the adversary's view ensures protection against selective failure attacks where whether a client accepts or not leaks information about the client's query. For example, in an actual execution of an authenticated-PIR scheme, a malicious server could replace a single record i in the database with garbage. Now, if the client's query does not depend on the value of record i, then everything proceeds normally. However, if the query does depend on the value of record i, then it receives a garbage value. Depending on the application, receiving a garbage value could cause the client to abort the protocol prematurely, or retry the protocol; in both of these cases, if the client engages in some kind of recovery mechanism, the server immediately learns information about the client's chosen index i. Definition 24 captures security against selective failure attacks by requiring that the probability of whether the client's response is valid or not (i.e., whether $y \neq \perp$) is *not* correlated with the client's query (since the *same* simulator works for all functions f and moreover, the simulator is not provided f as input). In this way, a malicious server that learns whether the protocol completed successfully or not still cannot learn anything about the client's query.

C Multi-server authenticated PIR for point queries

In this section we present the formal definition of the multi-server authenticated-PIR scheme for point queries based on a classic multi-server linear PIR scheme and a Merkle-tree scheme. In Construction 4, we give the scheme. In the remainder of this section, we give a overview of the strategy that we use to prove that Construction 4 satisfies integrity and privacy.

C.1 Overview of the proof strategy

In this section e give a overview of the strategy that we use to prove integrity and privacy for Construction 4. We describe the construction with $k = 2$ servers, but the same intuition generalizes to $k > 2$ servers. This overview is inspired by a private discussion with Brett Falk, Pratyush Mishra, and Matan Shtepel [84], which pointed out a flaw in the proof of Theorem 28.

Construction 4 uses a *linear* classic PIR scheme (Definition 15), i.e., the PIR.Reconstruct algorithm is the sum of the individual servers' answers. In other words, by denoting

Construction 4 (k-server authenticated PIR for point queries tolerating $k-1$ malicious servers)**.** The construction is parametrized by a number of servers $k \in \mathbb{N}$, a number of database rows $N \in \mathbb{N}$, a row length $\ell \in \mathbb{N}$, a security parameter $\lambda \in \mathbb{N}$, a Merkle-tree scheme M (Definition 16), and a linear PIR scheme PIR (Definition 15). Weights are ignored in this scheme. We represent the database as N binary strings of length ℓ each: $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \{0,1\}^\ell$. The Query algorithm inputs the security parameter 1^λ and an index $i \in [N]$; $\mathsf{Reconstruct}$ outputs either a vector $\mathbf{x}_i \in \{0,1\}^\ell$ or the rejection symbol $\perp$ (see Example 2 to recover this functionality from Definition 1). The servers execute the first three steps of the Answer procedure only when the database changes; we show the entire procedure for completeness.

$\underline{\mathsf{Query}(1^\lambda, i \in [N]) \to (\mathsf{st}, q_1, \ldots, q_k)}$

1. $(\mathsf{st}_{\mathsf{PIR}}, q_1, \ldots, q_k) \leftarrow \mathsf{PIR.Query}(i)$.
2. Set the state $\mathsf{st} \leftarrow (i, \mathsf{st}_{\mathsf{PIR}})$.
3. Output $(\mathsf{st}, q_1, \ldots, q_k)$.

$\underline{\mathsf{Answer}(\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_N \in \{0,1\}^\ell, q) \to a}$

1. Compute the digest $\mathsf{root} \leftarrow \mathsf{M.Digest}(\mathbf{X})$.
2. For $j \in [n]$, compute $\pi_i \leftarrow \mathsf{M.ProveIncludes}(\mathbf{X}, j, x_j)$.
3. Enlarge the database with the proofs for all the records as $\mathbf{X}' \leftarrow ((\mathbf{x}_1, \pi_1), \ldots, (\mathbf{x}_N, \pi_N))$.
4. Output $(\mathsf{root}, \mathsf{PIR.Answer}(\mathbf{X}', q))$.

$\underline{\mathsf{Reconstruct}(\mathsf{st}, a_1, \ldots, a_k) \to \{\{0,1\}^\ell, \perp\}}$

1. Parse the state st as $(i, \mathsf{st}_{\mathsf{PIR}})$.
2. For $j \in [k]$, parse a_k as (root_k, a'_k).
3. If the k roots $\{\mathsf{root}_j\}_{j \in [k]}$ are not all equal, return $\perp$.
4. Run the classic PIR reconstruction algorithm and parse $r_i \leftarrow \mathsf{PIR.Reconstruct}(\mathsf{st}_{\mathsf{PIR}}, a'_1, \ldots, a'_k)$ as $(\mathbf{x}_i, \pi_i)$.
5. If $\mathsf{M.VerifyIncludes}(\mathsf{root}_1, i, \mathbf{x}_i, \pi_i) = \perp$, then output $\perp$. Otherwise output $\mathbf{x}_i$.

the sum operation with $\oplus$, we can rewrite the reconstruction algorithm as

$$\mathsf{PIR.Reconstruct}(\mathsf{st}_{\mathsf{PIR}}, a'_1, \ldots, a'_k) \to a'_1 \oplus \cdots \oplus a'_k.$$

Construction 4 uses $\mathsf{PIR.Reconstruct}$ in line 4 of the $\mathsf{Reconstruct}$ procedure.

For the sake of this overview, we consider that the servers hold a copy of a two-record database $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2) \in (\{0,1\}^\ell)^2$, for row length $\ell \in \mathbb{N}$. Suppose that server 1 is malicious and that server 2 is honest, and suppose that server 1 mounts a selective-failure attack by replacing the record $\mathbf{x}_2$ with a bogus record $\mathbf{x}_2^* \in \{0,1\}^\ell$, i.e., server 1 uses the bogus database $\mathbf{X}^* = (\mathbf{x}_1, \mathbf{x}_2^*) \in (\{0,1\}^\ell)^2$. After the first three steps of the Answer procedure in Construction 4, the malicious server 1 has the following bogus enlarged database:

$$\mathbf{X}'_1 \leftarrow ((\mathbf{x}_1, \pi_1), (\mathbf{x}_2^*, \pi_2^*));$$

the honest server 2 has the correct enlarged database:

$$\mathbf{X}'_2 \leftarrow ((\mathbf{x}_1, \pi_1), (\mathbf{x}_2, \pi_2)).$$

Assume that server 1 sets a honest Merkle root, i.e., server 1 computes $\mathsf{root} \leftarrow \mathsf{M.Digest}(\mathbf{X})$; otherwise the client immediately rejects (line 3 of the $\mathsf{Reconstruct}$ procedure).

Given an index $i \in \{1,2\}$, the client computes queries q_1, q_2 using the Query procedure. The two servers compute the answers as follows:

$$a_1 \leftarrow (\mathsf{root}, \mathsf{PIR.Answer}(\mathbf{X}'_1, q_1))$$
$$a_2 \leftarrow (\mathsf{root}, \mathsf{PIR.Answer}(\mathbf{X}'_2, q_2))$$

Since the roots are equal, the client runs the classic PIR reconstruction procedure an gets $r_i \leftarrow \mathsf{PIR.Reconstruct}(\mathsf{st}_{\mathsf{PIR}}, a'_1, \ldots, a'_k)$. By the linearity of the classic PIR scheme and by setting $\Delta \leftarrow \mathsf{PIR.Answer}(\mathbf{X}'_1, q_1) \oplus \mathsf{PIR.Answer}(\mathbf{X}'_2, q_1)$, we have

$$\begin{aligned} r_i = a'_1 \oplus a'_2 &= \mathsf{PIR.Answer}(\mathbf{X}'_1, q_1) \oplus \mathsf{PIR.Answer}(\mathbf{X}'_2, q_2) \\ &= \mathsf{PIR.Answer}(\mathbf{X}'_2, q_1) \oplus \Delta \oplus \mathsf{PIR.Answer}(\mathbf{X}'_2, q_2) \\ &= (\mathbf{x}_i, \pi_i) \oplus \Delta, \end{aligned}$$

as $\mathbf{X}'_2$ is the honest enlarged database. By assumption the malicious server 1 feeds $\mathsf{PIR.Answer}$ with a bogus database, which implies that $\Delta = \mathsf{PIR.Answer}(\mathbf{X}'_1, q_1) \oplus \mathsf{PIR.Answer}(\mathbf{X}'_2, q_1) \neq 0$ and therefore that $(\mathbf{x}_i, \pi_i) \neq (\mathbf{x}_i, \pi_i) \oplus \Delta$. This in turn implies, by soundness and/or uniqueness of the Merkle-tree scheme (Definition 19 and Definition 18) that the client rejects the answers in step 5 of the $\mathsf{Reconstruct}$ procedure of Construction 4.

The crux is that the argument holds *regardless* of whether the client queried for index $i = 1$ or $i = 2$: we do not assume a specific index in the argument that leads to client's rejection. In other words, the client rejects, except with negligible probability, whenever one of the servers replies with respect to a bogus database independently from the index that the client inputs to the Query procedure.

C.2 Security proofs

We prove security for the case of $k = 2$ servers. All the arguments generalize naturally to the k-server setting with $k > 2$.

Correctness of the scheme introduced in Construction 4 can be verified by inspection. To prove both integrity and security, we find it useful to first prove Lemma 26, which

informally states that if a malicious server deviates from the prescribed protocol, the Reconstruct algorithm rejects with high probability.

Lemma 26. *Consider the authenticated-PIR scheme in Construction 4, on record size $\ell \in \mathbb{N}$ and with $k = 2$ servers for the sake of the proof. Recall that Construction 4 uses a linear PIR scheme (Definition 15). Then, for every $\lambda \in \mathbb{N}$, every non-zero $\Delta \in \{0,1\}^{\ell_{\mathsf{dig}}+\ell+\ell_\pi}$, where ℓ_{dig} is the length of the digest and ℓ_π is the length of a Merkle inclusion proof as per Definition 16, every database $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_N \in \{0,1\}^\ell$, and every index $i \in [N]$, the following holds:*

$$\Pr\left[\begin{array}{rc} & (\mathsf{st}, q_1, q_2) \leftarrow \mathsf{Query}(1^\lambda, i) \\ y \neq \perp : & a_1 \leftarrow \mathsf{Answer}(\mathbf{X}, q_1) \\ & a_2 \leftarrow \mathsf{Answer}(\mathbf{X}, q_2) \\ & y \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a_1 \oplus \Delta, a_2) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the probability is computed over all the random coins used by the algorithms of the scheme. The statement holds also when the roles of honest and malicious server are inverted.

Proof. We parse Δ as $(\Delta_{\mathsf{root}}, \Delta_{\mathbf{x}}, \Delta_\pi)$ where $\Delta_{\mathsf{root}} \in \{0,1\}^{\ell_{\mathsf{dig}}}$, $\Delta_{\mathbf{x}} \in \{0,1\}^\ell$, and $\Delta_\pi \in \{0,1\}^{\ell_\pi}$. If $\Delta_{\mathsf{root}} \neq 0^{\ell_{\mathsf{dig}}}$ then parsing $a_1 + \Delta$ and a_2 yields two different roots and the client immediately rejects (line 3 of Reconstruct in Construction 4). Hence, assume the client gets identical roots from the servers, i.e., $\Delta_{\mathsf{root}} = 0^{\ell_{\mathsf{dig}}}$. The client therefore receives two honest digests of the database $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_N \in \{0,1\}^\ell$.

Assume by contradiction that there is an index $i \in [N]$ and $\Delta = (\Delta_{\mathsf{root}}, \Delta_{\mathbf{x}}, \Delta_\pi)$ where $\Delta_{\mathsf{root}} = 0^{\ell_{\mathsf{dig}}}$, $\Delta_{\mathbf{x}} \in \{0,1\}^\ell$, $\Delta_\pi \in \{0,1\}^{\ell_\pi}$, and a database $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_N \in \{0,1\}^\ell$ such that

$$\Pr\left[\begin{array}{rc} & (\mathsf{st}, q_1, q_2) \leftarrow \mathsf{Query}(1^\lambda, i) \\ y \neq \perp : & a_1 \leftarrow \mathsf{Answer}(\mathbf{X}, q_1) \\ & a_2 \leftarrow \mathsf{Answer}(\mathbf{X}, q_2) \\ & y \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a_1 \oplus \Delta, a_2) \end{array}\right] \geq \nu,$$

where ν is *non-negligible* in the security parameter λ. By the assumption that the client gets identical roots from the servers, we can rewrite the above probability as

$$\Pr\left[\begin{array}{rc} & (\mathsf{st}, q_1, q_2) \leftarrow \mathsf{Query}(1^\lambda, i) \\ & (i, \mathsf{st}_{\mathsf{PIR}}) \leftarrow \mathsf{st} \\ & a_1 = (\mathsf{root}_1, a_1') \leftarrow \mathsf{Answer}(\mathbf{X}, q_1) \\ y \neq \perp : & a_2 = (\mathsf{root}_2, a_2') \leftarrow \mathsf{Answer}(\mathbf{X}, q_2) \\ & r_i \leftarrow \mathsf{PIR.Reconstruct}(\mathsf{st}_{\mathsf{PIR}}, a_1' \oplus (\Delta_{\mathbf{x}} \| \Delta_\pi), a_2') \\ & (\mathbf{x}_i, \pi_i) \leftarrow r_i \\ & y \leftarrow \begin{cases} \perp & \mathsf{M.VerifyIncludes}(\mathsf{root}_1, i, \mathbf{x}_i, \pi_i) = \perp \\ \mathbf{x}_i & \text{otherwise} \end{cases} \end{array}\right] \geq \nu.$$

By the linearity of the classic PIR scheme that Construction 4 uses, we can rewrite the above probability as

$$\Pr\left[\begin{array}{rc} & (\mathsf{st}, q_1, q_2) \leftarrow \mathsf{Query}(1^\lambda, i) \\ & a_1 = (\mathsf{root}_1, a_1') \leftarrow \mathsf{Answer}(\mathbf{X}, q_1) \\ & a_2 = (\mathsf{root}_2, a_2') \leftarrow \mathsf{Answer}(\mathbf{X}, q_2) \\ y \neq \perp : & r_i \leftarrow a_1' \oplus \Delta_{\mathbf{x}} \| \Delta_\pi \oplus a_2' \\ & (\mathbf{x}_i, \pi_i) \leftarrow r_i \\ & y \leftarrow \begin{cases} \perp & \mathsf{M.VerifyIncludes}(\mathsf{root}_1, i, \mathbf{x}_i, \pi_i) = \perp \\ \mathbf{x}_i & \text{otherwise} \end{cases} \end{array}\right] \geq \nu.$$

We now show that if $\Delta_{\mathbf{x}} \neq 0^\ell$, then the malicious servers breaks soundness of the Merkle-tree scheme (Definition 19). Alternatively, if $\Delta_{\mathbf{x}} = 0^\ell$, but $\Delta_\pi \neq 0^{\ell_\pi}$, then the malicious server breaks uniqueness of the Merkle-tree scheme (Definition 18).

We analyze the first case, that is, we assume that $\Delta_{\mathbf{x}} \neq 0^\ell$. Let $\mathcal{A}$ be an adversary in the definition of soundness for a Merkle-tree scheme (Definition 19). We show how $\mathcal{A}$ can use $\Delta = (\Delta_{\mathsf{root}}, \Delta_{\mathbf{x}}, \Delta_\pi)$ with $\Delta_{\mathsf{root}} = 0^{\ell_{\mathsf{dig}}}$, $\Delta_{\mathbf{x}} \neq 0^\ell$, $\Delta_\pi \in \{0,1\}^{\ell_\pi}$ to break the soundness property of the Merkle-tree scheme with a non-negligible probability. Given i, Δ, $\mathbf{X}$, the adversary $\mathcal{A}$ proceeds as follows:

1. Construct a query $(\mathsf{st}, q_1, q_2) \leftarrow \mathsf{Query}(1^\lambda, i)$.
2. For $k \in [2]$, compute $a_k = (\mathsf{root}_k, a_k') \leftarrow \mathsf{Answer}(\mathbf{X}, q_k)$.
3. Compute $r_i \leftarrow a_1' \oplus a_2'$ and
4. Parses the reconstructed value as $r_i = (\mathbf{x}_i, \pi_i)$.

Algorithm $\mathcal{A}$ outputs $(\mathbf{X}, i, \mathbf{x}_i \oplus \Delta_{\mathbf{x}}, \pi_i \oplus \Delta_\pi)$ in the soundness game of Definition 19. By assumption, the digest is correct and computed over $\mathbf{X}$. Since $\Delta_{\mathbf{x}} \neq 0^\ell$, we know that

$$(\mathbf{x}_i \oplus \Delta_{\mathbf{x}}) \| (\pi_i \oplus \Delta_\pi) \neq \mathbf{x}_i \| \pi_i.$$

Moreover, the probability stated in Definition 19 is equal to the probability stated in this lemma (i.e., to ν). Since by assumption ν is non-negligible in the security parameter λ, algorithm $\mathcal{A}$ successfully breaks the soundness property of the Merkle-tree scheme.

We analyze now the second case, that is, we assume that $\Delta_{\mathbf{x}} = 0^\ell$ and $\Delta_\pi \neq 0^{\ell_\pi}$. Let $\mathcal{A}'$ be an adversary in the definition of uniqueness for a Merkle-tree scheme (Definition 18). We show how $\mathcal{A}'$ can use $\Delta = (\Delta_{\mathsf{root}}, \Delta_{\mathbf{x}}, \Delta_\pi)$ with $\Delta_{\mathsf{root}} = 0^{\ell_{\mathsf{dig}}}$, $\Delta_{\mathbf{x}} = 0^\ell$, $\Delta_\pi \neq 0^{\ell_\pi}$ to break the uniqueness property of the Merkle-tree scheme with a non-negligible probability. Given i, Δ, $\mathbf{X}$, the adversary $\mathcal{A}'$ proceeds exactly as algorithm $\mathcal{A}$:

1. Construct a query $(\mathsf{st}, q_1, q_2) \leftarrow \mathsf{Query}(1^\lambda, i)$.
2. For $k \in [2]$, compute $a_k = (\mathsf{root}_k, a_k') \leftarrow \mathsf{Answer}(\mathbf{X}, q_k)$.
3. Compute $r_i \leftarrow a_1' \oplus a_2'$ and
4. Parses the reconstructed value as $r_i = (\mathbf{x}_i, \pi_i)$.

Algorithm $\mathcal{A}'$ outputs $(\mathbf{X}, i, \mathbf{x}_i, \pi_i, \pi_i \oplus \Delta_\pi)$ in the uniqueness game of Definition 18. Since $\Delta_\pi \neq 0^{\ell_\pi}$, we know that $\pi_i \neq \pi_i \oplus \Delta_\pi$. Moreover, the probability stated in Definition 18 is equal to the probability stated in this lemma (i.e., to ν). Since

by assumption ν is non-negligible in the security parameter λ, $\mathcal{A}'$ successfully breaks the uniqueness property of the Merkle-tree scheme. □

We now use Lemma 26 to show that the scheme presented in Construction 4 ensures integrity and security, and is hence secure.

Theorem 27 (Integrity of Construction 4)**.** *The authenticated PIR scheme of Construction 4 provides integrity.*

Proof. This follows directly from Lemma 26. □

Theorem 28 (Privacy of Construction 4)**.** *The authenticated PIR scheme of Construction 4 provides privacy.*

Proof. Recall that Construction 4 is an authenticated PIR scheme for point queries: the Query algorithm inputs an index $i \in [N]$ and Reconstruct outputs either a vector $\mathbf{x}_i \in \{0,1\}^\ell$ or the rejection symbol $\perp$. Example 2 shows how to recover this functionality from Definition 1.

Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be the adversary of Definition 24. We syntactically change the distribution modeling the real world by introducing an additional variable $\Delta \leftarrow a_1 \oplus a_{\mathcal{A}}$:

$$\text{REAL'}_{\mathcal{A},i,\lambda,\mathbf{X}} = \left\{ \begin{array}{l} (\mathsf{st}, q_1, q_2) \leftarrow \mathsf{Query}(1^\lambda, i) \\ a_j \leftarrow \mathsf{Answer}(\mathbf{X}, q_j) \quad \forall j \in [2] \\ \mathsf{st}_{\mathcal{A}}, a_{\mathcal{A}} \leftarrow \mathcal{A}_0(\mathbf{X}, q_1) \\ \hat{\beta}: \ \Delta \leftarrow a_1 \oplus a_{\mathcal{A}} \\ y \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a_1 \oplus \Delta, a_2) \\ b \leftarrow \mathbb{1}\{y \neq \perp\} \\ \hat{\beta} \leftarrow \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, b) \end{array} \right\},$$

where $\Delta \in \{0,1\}^{\ell_{\mathsf{dig}}+\ell+\ell_\pi}$, and ℓ_{dig} and ℓ_π are parameters of the Merkle-tree scheme (Definition 16) that Construction 4 uses. Without loss of generality we assume that server 1 is adversarial, i.e., we assign q_1 to the adversary. The proof holds also if we swap the queries.

We additionally adapt the distribution modeling the ideal world to the notation that we use in this proof (by renaming Q to q_1, A to $a_{\mathcal{A}}$ and ignoring weights $\mathbf{w} \in \mathbb{F}^N$ as Construction 4 does):

$$\text{IDEAL'}_{\mathcal{A},\mathcal{S},\mathcal{F},\lambda,\mathbf{X}} = \left\{ \begin{array}{l} \beta: \ (\mathsf{st}_{\mathcal{S}}, q_1) \leftarrow \mathcal{S}_0(1^\lambda, \mathcal{F}, \mathbf{X}) \\ (\mathsf{st}_{\mathcal{A}}, a_{\mathcal{A}}) \leftarrow \mathcal{A}_0(\mathbf{X}, q_1) \\ b \leftarrow \mathcal{S}_1(\mathsf{st}_{\mathcal{S}}, a_{\mathcal{A}}) \\ \beta \leftarrow \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, b) \end{array} \right\}.$$

For any adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ let a simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ such that for every $\lambda \in \mathbb{N}$, $\mathbf{X} = (\mathbf{x}_i, \ldots, \mathbf{x}_N) \in (\{0,1\}^\ell)^N$ the simulator proceeds as follows, where $\mathcal{S}_{\mathsf{PIR}}$ is the simulator induced by the classic PIR scheme that Construction 4 uses (see Definition 14):

Simulator $\mathcal{S}_0\left(1^\lambda, \mathcal{F}, \mathbf{X}\right)$

1: $q_1 \leftarrow \mathcal{S}_{\mathsf{PIR}}$
2: $\mathsf{st}_{\mathcal{S}} \leftarrow (\mathbf{X}, q_1)$
3: **return** $(\mathsf{st}_{\mathcal{S}}, q_1)$

Simulator $\mathcal{S}_1(\mathsf{st}_{\mathcal{S}}, a_{\mathcal{A}})$

1: $(\mathbf{X}, q_1) \leftarrow \mathsf{st}_{\mathcal{S}}$
2: $\Delta \leftarrow \mathsf{Answer}(\mathbf{X}, q_1) \oplus a_{\mathcal{A}}$
3: $b \leftarrow \mathbb{1}\{\Delta = 0\}$
4: **return** b

We now prove that the real and ideal distributions are computationally indistinguishable and hence the scheme presented in Construction 4 provides privacy. To this end, we define five hybrid distributions H_0, H_1, H_2, H_3, H_4:

- H_0: This is the distribution $\text{REAL'}_{\mathcal{A},i,\lambda,\mathbf{X}}$, where we define $\Delta \leftarrow a_1 \oplus a_{\mathcal{A}} = \mathsf{Answer}(\mathbf{X}, q_1) \oplus a_{\mathcal{A}}$ and the bit $b \leftarrow \mathbb{1}\{y \neq \perp\}$ given as input to the adversary $\mathcal{A}_0$ is determined using the output from the Reconstruct algorithm.
- H_1: Same as H_0 except the hybrid uses $\mathsf{Reconstruct}'$ instead of the Reconstruct procedure of Construction 4. $\mathsf{Reconstruct}'$ is the same as Reconstruct, except that it computes $r_i \leftarrow a'_1 \oplus a'_2$ instead of $r_i \leftarrow \mathsf{PIR.Reconstruct}(a'_1, a'_2)$ in line 4. The difference between H_0 and H_1 is boxed in the definition below:

$$\mathrm{H}_1 = \left\{ \begin{array}{l} (\mathsf{st}, q_1, q_2) \leftarrow \mathsf{Query}(1^\lambda, i) \\ a_j \leftarrow \mathsf{Answer}(\mathbf{X}, q_j) \quad \forall j \in [2] \\ \mathsf{st}_{\mathcal{A}}, a_{\mathcal{A}} \leftarrow \mathcal{A}_0(\mathbf{X}, q_1) \\ \hat{\beta}: \ \Delta \leftarrow a_1 \oplus a_{\mathcal{A}} \\ \boxed{y \leftarrow \mathsf{Reconstruct}'(\mathsf{st}, a_1 \oplus \Delta, a_2)} \\ b \leftarrow \mathbb{1}\{y \neq \perp\} \\ \hat{\beta} \leftarrow \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, b) \end{array} \right\},$$

- H_2: Same as H_1 except the hybrid computes the acceptance bit b by checking whether $\Delta = 0$. The difference between H_1 and H_2 is boxed in the definition below:

$$\mathrm{H}_2 = \left\{ \begin{array}{l} (\mathsf{st}, q_1, q_2) \leftarrow \mathsf{Query}(1^\lambda, i) \\ a_j \leftarrow \mathsf{Answer}(\mathbf{X}, q_j) \quad \forall j \in [2] \\ \mathsf{st}_{\mathcal{A}}, a_{\mathcal{A}} \leftarrow \mathcal{A}_0(\mathbf{X}, q_1) \\ \hat{\beta}: \ \Delta \leftarrow a_1 \oplus a_{\mathcal{A}} \\ y \leftarrow \mathsf{Reconstruct}'(\mathsf{st}, a_1 \oplus \Delta, a_2) \\ \boxed{b \leftarrow \mathbb{1}\{\Delta = 0\}} \\ \hat{\beta} \leftarrow \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, b) \end{array} \right\},$$

- H_3: Same as H_3 except the adversary gets a query produced by the simulator $\mathcal{S}_{\mathsf{PIR}}$ induced by the unauthenticated PIR scheme. The difference between H_2 and H_3 is

boxed in the definition below:

$$\mathrm{H}_3 = \left\{ \hat{\beta}: \begin{array}{l} (\mathsf{st}, \_, q_2) \leftarrow \mathsf{Query}(1^\lambda, i) \\ \boxed{q_1 \leftarrow \mathcal{S}_{\mathsf{PIR}}} \\ a_j \leftarrow \mathsf{Answer}(\mathbf{X}, q_j) \quad \forall j \in [2] \\ \mathsf{st}_{\mathcal{A}}, a_{\mathcal{A}} \leftarrow \mathcal{A}_0(\mathbf{X}, q_1) \\ \Delta \leftarrow a_1 \oplus a_{\mathcal{A}} \\ y \leftarrow \mathsf{Reconstruct}'(\mathsf{st}, a_1 \oplus \Delta, a_2) \\ b \leftarrow \mathbb{1}\{\Delta = 0\} \\ \hat{\beta} \leftarrow \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, b) \end{array} \right\},$$

- H_4: This is the distribution $\mathrm{IDEAL}'_{\mathcal{A},\mathcal{S},\mathcal{F},\lambda,\mathbf{X}}$.

We now argue that each pair of adjacent hybrids is indistinguishable. For $j \in \{0,1,2,3,4\}$, let W_b the event that the output of the hybrid experiment H_b is "1."

- Hybrid H_1 is the same as hybrid H_0 except H_1 uses $\mathsf{Reconstruct}'$ instead of $\mathsf{Reconstruct}$. As Construction 4 uses a linear classic PIR scheme, these hybrids are equal, i.e.,

$$|\Pr[W_0] - \Pr[W_1]| = 0.$$

- Hybrid H_2 is the same as hybrid H_1 except H_2 computes the acceptance but b by checking whether $\Delta = 0$. If $\Delta = 0^{\ell_{\mathsf{dig}}+\ell+\ell_\pi}$ (i.e, a binary string of $\ell_{\mathsf{dig}}+\ell+\ell_\pi$ zeros) the simulator $\mathcal{S}_1$ sets $b = 1$; if $\Delta \neq 0^{\ell_{\mathsf{dig}}+\ell+\ell_\pi}$, then $\mathcal{S}_1$ sets $b = 0$. By Lemma 26 we know that

$$|\Pr[b \leftarrow \mathbb{1}\{y \neq \perp\}] - \Pr[b \leftarrow \mathbb{1}\{\Delta = 0\}]| \leq \mathsf{negl}(\lambda),$$

  where the first probability refers to the assignment of bit b in H_1, while the second refers to the assignment of bit b in H_2. As the only difference between the two hybrids is how they set bit b, we can rewrite the above probability as

$$|\Pr[W_1] - \Pr[W_2]| \leq \mathsf{negl}(\lambda).$$

- The only difference between hybrids H_2 and H_3 is how the query q_1 is sampled, i.e., how the query that the adversary gets is sampled. By security of the classic unauthenticated PIR scheme (Definition 14), we have

$$|\Pr[W_2] - \Pr[W_3]| \leq \mathsf{negl}(\lambda).$$

- H_4 is a rewriting of H_3. In H_3, $\mathcal{A}_0$ inputs $q_1 \leftarrow \mathcal{S}_{\mathsf{PIR}}$, where $\mathcal{S}_{\mathsf{PIR}}$ if the simulator induced by the classic PIR scheme; the same happens in H_4. In H_3, $\mathcal{A}_1$ inputs $b \leftarrow \mathbb{1}\{\Delta = 0\}$ and the same happens in H_4. We show the bridging from H_3 to H_4 below, where we rewrite $\Delta \leftarrow a_1 \oplus a_{\mathcal{A}}$ as $\Delta \leftarrow \mathsf{Answer}(\mathbf{X}, q_1)$ and grey lines can be removed from H'_3 to get H_4:

$$\mathrm{H}'_3 = \left\{ \hat{\beta}: \begin{array}{l} (\mathsf{st}, \_, q_2) \leftarrow \mathsf{Query}(1^\lambda, i) \\ q_1 \leftarrow \mathcal{S}_{\mathsf{PIR}} \\ a_j \leftarrow \mathsf{Answer}(\mathbf{X}, q_j) \quad \forall j \in [2] \\ \mathsf{st}_{\mathcal{A}}, a_{\mathcal{A}} \leftarrow \mathcal{A}_0(\mathbf{X}, q_1) \\ \Delta \leftarrow \mathsf{Answer}(\mathbf{X}, q_1) \oplus a_{\mathcal{A}} \\ y \leftarrow \mathsf{Reconstruct}'(\mathsf{st}, a_1 \oplus \Delta, a_2) \\ b \leftarrow \mathbb{1}\{\Delta = 0\} \\ \hat{\beta} \leftarrow \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, b) \end{array} \right\}.$$

Since the hybrids are equal we have

$$|\Pr[W_3] - \Pr[W_4]| = 0.$$

By a standard hybrid argument we conclude that $\mathrm{REAL}'_{\mathcal{A},i,\lambda,\mathbf{X}} \approx_c \mathrm{IDEAL}'_{\mathcal{A},\mathcal{S},\mathcal{F},\lambda,\mathbf{X}}$ and therefore

$$\mathrm{REAL}_{\mathcal{A},i,\lambda,\mathbf{X}} \approx_c \mathrm{IDEAL}_{\mathcal{A},\mathcal{S},\mathcal{F},\lambda,\mathbf{X}}. \qquad \square$$

C.3 Preprocessing costs

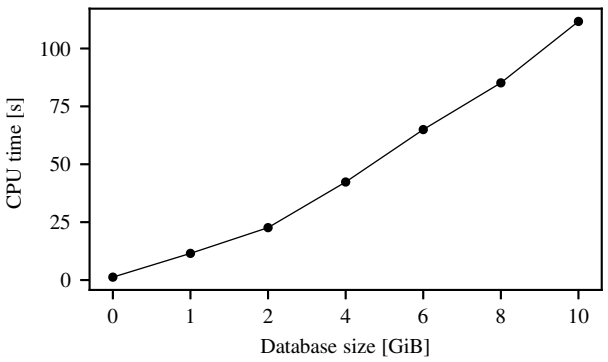

Figure 8: The CPU time that a single server takes to process the database for the authenticated PIR scheme for point queries (§4.1). The Merkle-tree computation is not parallelized.

In our multi-server authenticated-PIR scheme for point queries, the servers must compute a Merkle tree over the N database entries along with their indexes. The computational complexity of the preprocessing phase is dominated by the number N of database records. Fig. 8 shows the CPU time that a single server takes to compute a Merkle tree for different database sizes. The current implementation is not parallelized, but in practice, the Merkle-tree computation can be efficiently divided into multiple cores.

D Multi-server authenticated PIR for predicate queries

In this section we analyze our multi-server authenticated-PIR scheme for predicate queries.

D.1 Security proofs for multi-server authenticated PIR for predicate queries

We prove security only for the case of $k = 2$ servers. All the arguments generalize naturally to the k-server setting with $k > 2$. Correctness of the multi-server authenticated PIR scheme for predicate queries introduced in Construction 1 can be verified by inspection. To prove integrity and security, we find it useful to first prove Lemma 29, which states that if an adversary deviates from the prescribed protocol, the Reconstruct algorithm rejects with high probability.

Lemma 29. *Let the authenticated PIR scheme introduced in Construction 1, where $k = 2$ for this lemma. Then, for every database size $N \in \mathbb{N}$, for every non-zero offset $\Delta = (\Delta_m, \Delta_\tau) \in \mathbb{F}^2$, every database $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_N \in \{0,1\}^\ell$, every vector of weights $\mathbf{w} \in \mathbb{F}^N$, and function $f \in \mathcal{F}$, we have*

$$\Pr\left[y \neq \perp : \begin{array}{r} (\mathsf{st}, q_1, q_2) \leftarrow \mathsf{Query}(1^\lambda, f) \\ \mathbf{a}_1 \leftarrow \mathsf{Answer}(\mathbf{X}, \mathbf{w}, q_1) \\ \mathbf{a}_2 \leftarrow \mathsf{Answer}(\mathbf{X}, \mathbf{w}, q_2) \\ y \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a_1 + \Delta, a_2) \end{array} \right] \leq \frac{1}{|\mathbb{F}| - 1},$$

where the probability is computed over all the random coins used by the algorithms of the experiment. The statement holds also when the Reconstruct *algorithm instead takes as input* $(\mathsf{st}, \mathbf{a}_1, \mathbf{a}_2 + \Delta)$.

Proof. Let $\alpha \xleftarrow{R} \mathbb{F} \setminus \{0\}$. By construction, we can rewrite the probability stated in the lemma as

$$\begin{aligned} \nu &= \Pr\left[\alpha \cdot \left(\sum_{i \in [N]} w_i \cdot f(i, \mathbf{x}_i) + \Delta_m \right) = \alpha \cdot \sum_{i \in [N]} w_i \cdot f(i, \mathbf{x}_i) + \Delta_\tau \right] \\ &= \Pr\left[-\Delta_\tau + \alpha \cdot \Delta_m = 0 \right] \end{aligned}$$

The last quantity is the evaluation of a non-zero degree-1 polynomial with coefficients Δ_τ and Δ_m at a random point $\alpha \xleftarrow{R} \mathbb{F} \setminus \{0\}$. Since a non-zero linear polynomial has at most one root over $\mathbb{F} \setminus \{0\}$, we conclude that $\nu \leq \frac{1}{|\mathbb{F}|-1}$. By interchanging the roles of $\mathbf{a}_1$ and $\mathbf{a}_2$, the statement holds also when the Reconstruct algorithm instead takes as input $(\mathsf{st}, \mathbf{a}_1, \mathbf{a}_2 + \Delta)$. □

We now use Lemma 29 to show that the scheme presented in Construction 1 ensures integrity and security, and hence it is secure.

Theorem 30 (Integrity of Construction 1)**.** *The authenticated PIR scheme of Construction 1 provides integrity.*

Proof. This theorem follows directly from Lemma 29. □

Theorem 31 (Privacy of Construction 1)**.** *The authenticated PIR scheme of Construction 1 provides privacy.*

Proof. The proofs proceeds exactly as the proof for Theorem 28, with the difference that we use the simulator induced by the secure function-secret-sharing scheme instead of the simulator induced by the classic PIR scheme, and we appeal to Lemma 29 instead of Lemma 26 to conclude the proof. □

D.2 Handling functions with larger output

In this section we discuss how to handle functions with larger output in authenticated PIR for predicate queries.

D.2.1 Scheme

The scheme is described in Construction 5.

Construction 5 (k-server authenticated PIR for predicate queries for functions whose output is larger than a single field element tolerating $k-1$ malicious servers)**.** The construction is parametrized by a number of servers $k \in \mathbb{N}$, a number of database rows $N \in \mathbb{N}$, a row length $\ell \in \mathbb{N}$, a finite field $\mathbb{F}$, a security parameter λ, a output length $b \in \mathbb{N}$, a function class $\mathcal{F} \subseteq \mathsf{Funs}[[N] \times \{0,1\}^\ell, \mathbb{F}^b]$ that is closed under scalar multiplication, and a function-secret-sharing scheme $(\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ for the function class $\mathcal{F}$, parametrized bx the security parameter λ. We represent the database as N binary strings of length ℓ each: $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_N \in \{0,1\}^\ell$.

$\mathsf{Query}\left(1^\lambda, f\right) \to (\mathsf{st}, \mathbf{q}_1, \ldots, \mathbf{q}_k)$

1. Sample a random field element $\alpha \xleftarrow{R} \mathbb{F} \setminus \{0\}$.
2. Set the state $\mathsf{st} \leftarrow \alpha$.
3. For $j \in [b]$, let $g_j \leftarrow \alpha^j \cdot f$. These functions g_j must exist since the function class $\mathcal{F}$ is closed under scalar multiplication, as in Definition 7.
4. Compute $q_1, \ldots, q_k \leftarrow \mathsf{FSS.Gen}(1^\lambda, f)$ together with $q_1^{(i)}, \ldots, q_k^{(i)} \leftarrow \mathsf{FSS.Gen}(1^\lambda, g_i)$, for $i \in [b]$.
5. Output $\left(\mathsf{st}, (q_1, q_1^{(1)}, \ldots, q_1^{(b)}), \ldots, (q_k, q_k^{(1)}, \ldots, q_k^{(b)})\right)$.

$\mathsf{Answer}\left(\mathbf{x}_1, \ldots, \mathbf{x}_N \in \{0,1\}^\ell, \mathbf{w} \in \mathbb{F}^N, \mathbf{q}\right) \to \mathbf{a} \in \mathbb{F}^{b+1}$

1. Parse $\mathbf{q}$ as $(q_f, q_g^{(1)}, \ldots, q_g^{(b)})$.
2. Compute answer as $a_f \leftarrow \sum_{i \in [N]} w_i \cdot \mathsf{FSS.Eval}(q_f, x_i)$ and $a_g \leftarrow \sum_{j \in [b]} \left(\sum_{i \in [N]} w_i \cdot \mathsf{FSS.Eval}\left(q_g^{(j)}, x_i\right)\right)$.
3. Return $\mathbf{a} \leftarrow (a_f, a_g)$.

$\mathsf{Reconstruct}\left(\mathsf{st}, a_1, \ldots, a_k \in \mathbb{F}^{b+1}\right) \to \mathbb{F}^b \cup \{\perp\}$

1. Parse the state st as $\alpha \in \mathbb{F}$.
2. Compute $\mathbf{a} \leftarrow a_1 + \cdots + a_k \in \mathbb{F}^{b+1}$.
3. Parse a as $(m_1, \ldots, m_b, \tau) \in \mathbb{F}^{b+1}$.
4. Compute $\tau' \leftarrow m_1\alpha + m_2\alpha^2 + \cdots + m_b\alpha^b \in \mathbb{F}$.
5. If $\tau = \tau'$, output $(m_1, \ldots, m_b) \in \mathbb{F}^b$. Otherwise, output $\perp$.

D.2.2 Security analysis

Lemma 32. *Let the authenticated PIR scheme introduced in Construction 5, where $k = 2$ for this lemma. Then, for every database size $N \in \mathbb{N}$, for every non-zero vector $(\Delta_0, \ldots, \Delta_b) \in \mathbb{F}^{b+1}$, every database $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_N \in \{0,1\}^\ell$, every vector of weights $\mathbf{w} \in \mathbb{F}^N$, and every function $f \in \mathcal{F}$, the following holds:*

$$\Pr\left[y \neq \perp : \begin{array}{c} (\mathsf{st}, q_1, q_2) \leftarrow \mathsf{Query}(\lambda, f) \\ \mathbf{a}_1 \leftarrow \mathsf{Answer}(\mathbf{X}, \mathbf{w}, q_1) \\ \mathbf{a}_2 \leftarrow \mathsf{Answer}(\mathbf{X}, \mathbf{w}, q_2) \\ y \leftarrow \mathsf{Reconstruct}(\mathsf{st}, \mathbf{a}_1 + \Delta, \mathbf{a}_2) \end{array} \right] \leq \frac{b}{|\mathbb{F}| - 1},$$

where the probability is computed over all the random coins used by the algorithms of the scheme. Without loss of generality, the statement holds also when the roles of honest and malicious server are inverted.

Proof. Let $\alpha \xleftarrow{R} \mathbb{F} \setminus \{0\}$. Let

$$y = (m_1, \ldots, m_b) \leftarrow \sum_{i \in [N]} w_i \cdot f(i, \mathbf{x}_i) \in \mathbb{F}^b.$$

Then the probability stated in the lemma is

$$\begin{aligned} \nu &= \Pr\left[\sum_{j \in [b]} (m_j + \Delta_j)\alpha^j = \Delta_0 + \sum_{j \in [b]} m_j \alpha^j \right] \\ &= \Pr\left[-\Delta_0 + \sum_{j \in [b]} \Delta_j \alpha^j = 0 \right]. \end{aligned}$$

This last quantity is the evaluation of a non-zero polynomial (whose coefficients are the Δ values) at a random point $\alpha \xleftarrow{R} \mathbb{F} \setminus \{0\}$. Since such a non-zero polynomial of degree at most b can have at most b roots over $\mathbb{F}$, we have that $\nu \leq \frac{b}{|\mathbb{F}|-1}$. By interchanging the roles of a_0 and a_1, the statement holds also when the Reconstruct algorithm instead takes as input $(\mathsf{st}, \mathbf{a}_1, \mathbf{a}_2 + \Delta)$. □

Theorem 33 (Integrity of Construction 5)**.** *The authenticated PIR scheme of Construction 5 provides integrity.*

Proof. The theorem follows directly from Lemma 32. □

Theorem 34 (Security of Construction 5)**.** *The authenticated PIR scheme of Construction 5 provides privacy.*

Proof. The strategy is as in the proof of Theorem 31, except that we appeal to Lemma 32 to complete the argument. □

E Definition of single-server authenticated PIR

In this section we present the formal definitions of single-server authenticated PIR.

E.1 Definitions

Definition 35 (Single-server authenticated PIR correctness)**.** *A single-server authenticated-PIR scheme* $(\mathsf{Digest}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ *satisfies* correctness *if for every database $x \in \{0,1\}^N$, $i \in [N]$, and $\lambda \in \mathbb{N}$, the following holds:*

$$\Pr\left[x'_i = x_i : \begin{array}{c} d \leftarrow \mathsf{Digest}(1^\lambda, x) \\ (\mathsf{st}, q) \leftarrow \mathsf{Query}(d, i) \\ a \leftarrow \mathsf{Answer}(d, x, q) \\ x'_i \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda),$$

Definition 36 (Single-server authenticated PIR integrity)**.** *A single-server authenticated-PIR scheme* $(\mathsf{Digest}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ *has integrity error* ε *if for every efficient (non-uniform) adversary $\mathcal{A}$, every database $x \in \{0,1\}^N$, and index $i \in [N]$,*

$$\Pr\left[x'_i \notin \{x_i, \perp\} : \begin{array}{c} d \leftarrow \mathsf{Digest}(1^\lambda, x) \\ (\mathsf{st}, q) \leftarrow \mathsf{Query}(d, i) \\ a^* \leftarrow \mathcal{A}(d, x, q) \\ x'_i \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a^*) \end{array} \right] \leq \varepsilon(\lambda) + \mathsf{negl}(\lambda),$$

where the probability is only *taken over the choice of* query randomness[2]*. We say the scheme provides integrity if it has integrity error* 0*.*

Remark 37 (On non-uniform hardness)**.** As written, Definition 36 requires integrity to hold against *non-uniform* adversaries. This version of the assumption explicitly captures the fact that the probability of an integrity failure is only taken over the randomness of query generation (and *not* the adversary). Thus, a malicious server cannot induce correlated integrity failures across multiple independently-generated queries. This property is very useful for our integrity amplification transformation (Appendix E.2). We could also consider a more complex (multi-query) variant of this assumption that applies to both uniform and non-uniform adversaries (and which suffices for the transformation in Appendix E.2). For simplicity of exposition, we opt to give the stronger, but simpler-to-describe non-uniform notion here.

Definition 38 (Single-server authenticated PIR privacy)**.** *Let* $(\mathsf{Digest}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ *be a single-server authenticated-PIR scheme. For a security parameter $\lambda \in \mathbb{N}$, a database $x \in \{0,1\}^N$, an index $i \in [N]$, and an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, define the distribution*

$$\mathrm{REAL}_{\mathcal{A},x,i,\lambda} := \left\{ \hat{\beta} : \begin{array}{l} d \leftarrow \mathsf{Digest}(1^\lambda, x) \\ (\mathsf{st}, q) \leftarrow \mathsf{Query}(d, i) \\ (\mathsf{st}_{\mathcal{A}}, a^*) \leftarrow \mathcal{A}_0(d, x, q) \\ x'_i \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a^*) \\ b \leftarrow \mathbb{1}\{x'_i \neq \perp\} \\ \hat{\beta} \leftarrow \mathcal{A}_1(\mathsf{st}_A, b) \end{array} \right\}.$$

[2]Note that since the adversary is allowed to take *non-uniform* advice, we can assume without loss of generality that the adversary is deterministic (and incur at most a constant loss in advantage).

Similarly, for a simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$, define the distribution

$$\text{IDEAL}_{\mathcal{A},\mathcal{S},x,\lambda} := \left\{ \beta : \begin{array}{l} d \leftarrow \mathsf{Digest}(1^\lambda, x) \\ (\mathsf{st}_{\mathcal{S}}, q) \leftarrow \mathcal{S}_0(d, x) \\ (\mathsf{st}_{\mathcal{A}}, a^*) \leftarrow \mathcal{A}_0(d, x, q) \\ b \leftarrow \mathcal{S}_1(\mathsf{st}_{\mathcal{S}}, a^*) \\ \beta \leftarrow \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, b) \end{array} \right\}.$$

An authenticated PIR scheme $(\mathsf{Digest}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ *has* privacy *if for every adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ there exists a simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ such that for every database length $N = N(\lambda)$, database $x \in \{0,1\}^N$, index $i \in [N]$, the following holds:*

$$|\Pr[\text{REAL}_{\mathcal{A},x,i,\lambda} = 1] - \Pr[\text{IDEAL}_{\mathcal{A},\mathcal{S},x,\lambda} = 1]| \leq \mathsf{negl}(\lambda).$$

Remark 39 (Adaptive notions of privacy)**.** We could also consider stronger versions of privacy (Definition 38) where the adversary chooses the query *adaptively* after seeing the digest. In both of our single-server authenticated-PIR constructions (Constructions 2 and 3), the digest is a *deterministic* function of the database, and hence, choosing the query adaptively does not help the adversary. For this reason, we opt to give the (simpler) privacy definition.

E.2 Amplifying integrity

Later on, we will construct lattice-based authenticated-PIR schemes (Construction 2) that has privacy but that have noticeable integrity error $\varepsilon = 1/\mathsf{poly}(\lambda)$. Here, we show to combine a secure authenticated-PIR scheme with integrity error ε with any error-correcting code to obtain a private scheme with negligible integrity error:

- The server first encodes each database record with an error-correcting code. Suppose each encoded record is n bits. The server constructs n databases where the j^{th} database contains the j^{th} bit of the codeword for each record.
- To retrieve a record i, the client makes n authenticated PIR queries to obtain the n bits of the codeword encoding record i. Let $y_1, \ldots, y_n$ be the responses. If $y_j = \perp$ for any $j \in [n]$, the client rejects with output $\perp$. Otherwise, the client decodes $y = y_1 \cdots y_n$ to obtain the record.

If the error-correcting code supports decoding codewords with up to t errors and the authenticated-PIR scheme has integrity error ε, then the integrity error of this construction is at most ε^{t+1}. Specifically, to compromise integrity, the server must corrupt at least $t+1$ bits y_j. Integrity of the underlying scheme ensures that the probability the adversary succeeds in corrupting y_j is at most ε. Each query is *independent*, so the server's success probability is now ε^{t+1}.

A basic instantiation of this paradigm is to instantiate using a repetition code where the encoding of a bit $b \in \{0,1\}$ simply consists of $2t+1$ copies of b. This basic repetition code supports correcting up to t errors so the integrity error is now ε^{t+1}. Setting $t = \lambda/\varepsilon$ then yields a construction with negligible integrity error. When the database records are longer (e.g., field elements instead of bits), we can use better error-correcting codes with higher rate compared to the basic repetition code. This allows amplifying integrity with fewer repetitions. In the following, we describe the construction more formally that supports multi-bit records over any field:

Definition 40 (Error-correcting code)**.** *A (k,n)-error-correcting code over a finite field $\mathbb{F}$ that can correct up to t errors consists of two efficient and deterministic algorithms:*

- $\mathsf{Encode}(\mathbf{x}) \to \mathbf{y}$*: The encoding algorithm takes a message $\mathbf{x} \in \mathbb{F}^k$ and outputs a codeword $\mathbf{y} \in \mathbb{F}^n$.*
- $\mathsf{Decode}(\mathbf{y}) \to \mathbf{x}$*: The decoding algorithm takes a codeword $\mathbb{F}^n$ and outputs a message $\mathbf{x} \in \mathbb{F}^n$.*

Moreover, for all $\mathbf{x} \in \mathbb{F}^k$, $\mathbf{y} \leftarrow \mathsf{Encode}(\mathbf{x})$, and all $\mathbf{y}' \in \mathbb{F}^n$ such that $\mathbf{y}_i = \mathbf{y}'_i$ for all but at most t indices $i \in [n]$, $\mathsf{Decode}(\mathbf{y}') = \mathbf{x}$.

Construction 6 shows how to use an error-correcting code to amplify the integrity of a single-server PIR scheme. Correctness of Construction 6 follows by construction. Thus, we focus on analyzing integrity and security.

E.2.1 Integrity of Construction 6

Theorem 41 (Integrity of Construction 6)**.** *If PIR_0 is secure and provides ε-integrity and ECC is an error-correcting code that can correct up to t errors, then Construction 6 (instantiated with PIR_0 and ECC) provides ε^{t+1}-integrity.*

Proof. Take any database $\mathbf{x} \in (\mathbb{F}^k)^N$, an index $i \in [N]$, and any efficient adversary $\mathcal{A}$. Write $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ and let $\mathbf{y}_i \leftarrow \mathsf{Encode}(\mathbf{x}_i)$ for each $i \in [n]$. Let $\mathbf{z}_j \leftarrow (y_{1,j}, \ldots, y_{n,j})$ and let $d \leftarrow \mathsf{Digest}(1^\lambda, \mathbf{x})$. Then $d = (d_1, \ldots, d_n)$ where $d_j \leftarrow \mathsf{Digest}_0(1^\lambda, \mathbf{z}_j)$. Let $(\mathsf{st}, q) \leftarrow \mathsf{Query}(d, i)$ where $\mathsf{st} = (\mathsf{st}_1, \ldots, \mathsf{st}_n)$, $q = (q_1, \ldots, q_n)$, and $(\mathsf{st}_j, q_j) \leftarrow \mathsf{Query}_0(d_j, i)$. Let $a^* = (a_1^*, \ldots, a_n^*)$ be the adversary's response in the integrity experiment. Let $y'_j \leftarrow \mathsf{Reconstruct}_0(\mathsf{st}_j, a_j^*)$. Consider now the output of $\mathbf{x}' \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a^*)$:

- Suppose there exists $j \in [t]$ such that $y'_j = \perp$. Then $\mathbf{x}' = \perp$.
- Suppose $y'_j = y_{i,j}$ for all but at most t indices $j \in [n]$. Since ECC can correct up to t errors, $\mathbf{x}' = \mathsf{Decode}((y'_1, \ldots, y'_j)) = \mathbf{x}_i$.
- Suppose there are at least $t+1$ indices $j \in [n]$ where $y'_j \notin \{y_{i,j}, \perp\}$. By integrity of PIR_0, for each $j \in [n]$,

$$\Pr\left[y'_j \notin \{y_{i,j}, \perp\}\right] \leq \varepsilon(\lambda) + \mathsf{negl}(\lambda).$$

  Moreover, this probability is taken only over the choice of the query randomness q_j. Since the queries $q_1, \ldots, q_n$ are sampled independently, the probability that there exists $t+1$ indices j where $y'_j \notin \{y_{i,j}, \perp\}$ is at most $\varepsilon^{t+1} + \mathsf{negl}(\lambda)$.

Construction 6 (Amplifying integrity of single-server authenticated PIR)**.** Let $\mathsf{ECC} = (\mathsf{Encode}, \mathsf{Decode})$ be a (k,n)-error-correcting code over a finite field $\mathbb{F}$ that can correct up to t errors. Let $\mathsf{PIR}_0 = (\mathsf{Digest}_0, \mathsf{Query}_0, \mathsf{Answer}_0, \mathsf{Reconstruct}_0)$ be a secure single-server authenticated PIR scheme for records in $\mathbb{F}$ and which provides ε-integrity. We construct a new single-server authenticated PIR scheme from PIR_0 with records in $\mathbb{F}^k$.

$\underline{\mathsf{Digest}(1^\lambda, \mathbf{x} \in (\mathbb{F}^k)^N) \to d}$

1. Parse $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ where $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{F}^k$.
2. For each $i \in [N]$, let $\mathbf{y}_i \leftarrow \mathsf{Encode}(\mathbf{x}_i) \in \mathbb{F}^n$. Write $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,n})$.
3. For each $j \in [n]$, let $\mathbf{z}_j = (y_{1,j}, \ldots, y_{n,j}) \in \mathbb{F}^N$.
4. For each $j \in [n]$, compute $d_j \leftarrow \mathsf{Digest}_0(1^\lambda, \mathbf{z}_j)$ and output $d = (d_1, \ldots, d_n)$.

$\underline{\mathsf{Query}(d, i \in [N]) \to (\mathsf{st}, q)}$

1. For each $j \in [n]$, sample $(\mathsf{st}_j, q_j) \leftarrow \mathsf{Query}_0(d_j, i)$.
2. Output $\mathsf{st} = (\mathsf{st}_1, \ldots, \mathsf{st}_n), q = (q_1, \ldots, q_n)$.

$\underline{\mathsf{Answer}\left(d, \mathbf{x} \in (\mathbb{F}^k)^N, q\right) \to a}$

1. Parse $d = (d_1, \ldots, d_n)$ and $q = (q_1, \ldots, q_n)$.
2. For each $j \in [n]$, compute $\mathbf{z}_j \in \mathbb{F}^N$ from $\mathbf{x}$ using the same procedure as Digest.
3. For each $j \in [n]$, compute $a_j \leftarrow \mathsf{Answer}_0(d_j, \mathbf{z}_j, q_j)$ and output $a = (a_1, \ldots, a_n)$.

$\underline{\mathsf{Reconstruct}(\mathsf{st}, a) \to \mathbb{F}^k \cup \{\perp\}}$

1. Parse the state $\mathsf{st} = (\mathsf{st}_1, \ldots, \mathsf{st}_n)$ and the responses $a = (a_1, \ldots, a_n)$.
2. For each $j \in [n]$, compute $y_j \leftarrow \mathsf{Reconstruct}_0(\mathsf{st}_j, a_j)$.
3. If there exists $j \in [n]$ such that $y_j = \perp$, output $\perp$.
4. Otherwise, let $\mathbf{y} = (y_1, \ldots, y_n)$ and output $\mathsf{Decode}(\mathbf{y})$.

By the above analysis, we conclude that

$$\Pr[\mathbf{x}' \notin \{\mathbf{x}_i, \perp\}] \leq \varepsilon^{t+1} + \mathsf{negl}(\lambda),$$

and the claim holds. □

E.2.2 Privacy of Construction 6

Theorem 42 (Privacy of Construction 6)**.** *If* PIR_0 *provides privacy, then Construction 6 (instantiated with* PIR_0*) provides privacy.*

Proof. Take any database $\mathbf{x} \in (\mathbb{F}^k)^N$, an index $i \in [N]$, and any efficient adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$. Let $\mathcal{S}' = (\mathcal{S}'_0, \mathcal{S}'_1)$ be the simulator for PIR_0. We use $(\mathcal{S}'_0, \mathcal{S}'_1)$ to construct a simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ for the transformed scheme:

Simulator $\mathcal{S}_0\left(1^\lambda, d, \mathbf{x}\right)$

parse d as $(d_1, \ldots, d_n)$
parse $\mathbf{x}$ as $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$
$\mathbf{y}_i \leftarrow \mathsf{Encode}(\mathbf{x}_i)\ \forall i \in [N]$
for all $j \in [n]$:
  $\mathbf{z}_j \leftarrow (y_{1,j}, \ldots, y_{n,j})$
  $(\mathsf{st}_j, q_j) \leftarrow \mathcal{S}'_0(1^\lambda, d_j, \mathbf{z}_j)$
$\mathsf{st}_\mathcal{S} \leftarrow (\mathsf{st}_1, \ldots, \mathsf{st}_n)$
$q \leftarrow (q_1, \ldots, q_n)$
return $(\mathsf{st}_\mathcal{S}, q)$

Simulator $\mathcal{S}_1(\mathsf{st}_\mathcal{S}, a^*)$

parse $\mathsf{st}_\mathcal{S}$ as $(q_1, \ldots, q_n)$
parse a^* as $(a_1^*, \ldots, a_n^*)$
$b_j \leftarrow \mathcal{S}'_1(\mathsf{st}_j, a_j^*)\ \forall j \in [n]$
$b \leftarrow \mathbb{1}\{\forall j \in [n] : b_j = 1\}$
return b

We show that the real distribution $\text{REAL}_{\mathcal{A},\mathbf{x},i,\lambda}$ and ideal distribution $\text{IDEAL}_{\mathcal{A},\mathcal{S},\mathbf{x},\lambda}$ are computationally indistinguishable. We define a sequence of hybrid experiments:

- H_0: This is the real distribution $\text{REAL}_{\mathcal{A},\mathbf{x},i,\lambda}$:
  - The challenger starts by parsing $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and computes $\mathbf{y}_i \leftarrow \mathsf{Encode}(\mathbf{x}_i)$ for each $i \in [N]$. Then it forms $\mathbf{z}_j = (y_{1,j}, \ldots, y_{n,j})$ for each $j \in [n]$. It computes $d_j \leftarrow \mathsf{Digest}_0(1^\lambda, \mathbf{z}_j)$ and sets $d_j = (d_1, \ldots, d_n)$.
  - The challenger then samples $(\mathsf{st}_j, q_j) \leftarrow \mathsf{Query}_0(d_j, \mathbf{z}_j)$. It define $q = (q_1, \ldots, q_n)$ and gives $(d, \mathbf{x}, q)$ to $\mathcal{A}$.
  - The adversary responds with $a^* = (a_1^*, \ldots, a_n^*)$. For each $j \in [n]$, the challenger computes $y_j \leftarrow \mathsf{Reconstruct}_0(\mathsf{st}_j, a_j^*)$.
  - Then it computes $b \leftarrow \mathbb{1}\{\forall j \in [n] : y_j \neq \perp\}$ and gives b to $\mathcal{A}$.
  - The output of the experiment is $\mathcal{A}$'s output.
- H_1: Same as H_0 except after the challenger computes $y_1, \ldots, y_n$ from a^*, the challenger computes $b_j \leftarrow \mathbb{1}\{y_j \neq \perp\}$. Then, it sets $b \leftarrow \mathbb{1}\{\forall j \in [n] : b_j = 1\}$.
- H_2: Same as H_1 except the challenger computes $(\mathsf{st}_j, q_j) \leftarrow \mathcal{S}'_0(1^\lambda, d_j, \mathbf{z}_j)$ for each $j \in [N]$. After the adversary responds with $a^* = (a_1^*, \ldots, a_n^*)$, the challenger computes b_j as $b_j \leftarrow \mathcal{S}'_1(\mathsf{st}_j, a_j^*)$. This is the ideal distribution $\text{IDEAL}_{\mathcal{A},\mathcal{S},\mathbf{x},\lambda}$.

The difference between H_0 and H_1 is syntactic and their outputs are identically distributed. Hybrid H_1 and H_2 are computationally indistinguishable by security of PIR_0; formally, this follows by a sequence of n hybrid experiments where in experiment j, we switch to using the PIR_0 simulator $\mathcal{S}'$ to simulate the query q_j and the response bit b_j. □

F Single-server authenticated PIR from LWE

In this section, we analyze Construction 2.

F.1 Lattice preliminaries

For a real value $s > 0$, we write $\rho_s \colon \mathbb{R} \to \mathbb{R}^+$ to denote the Gaussian function $\rho_s(x) := \exp(-\pi x^2/\sigma^2)$. The discrete Gaussian distribution $D_{\mathbb{Z},s}$ with width parameter s is a discrete distribution over the integers with probability mass function

$$\Pr[X = x : X \leftarrow D_{\mathbb{Z},s}] = \frac{\rho_s(x)}{\sum_{y\in\mathbb{Z}} \rho_s(y)}.$$

We say that a distribution D (over $\mathbb{R}$) is subgaussian with parameter s if for every $t \geq 0$,

$$\Pr[|x| > t : x \leftarrow D] \leq 2\exp(-\pi t^2/s^2). \tag{1}$$

The discrete Gaussian distribution $D_{\mathbb{Z},s}$ is *subgaussian* with parameter s. In particular, this means that if we sample $e \leftarrow D_{\mathbb{Z},s}$, then $|e| \leq \sqrt{\lambda}s$ with probability $1 - \mathsf{negl}(\lambda)$. Moreover, if x_1, x_2 are independent subgaussian random variables with parameters s_1, s_2, then $x = \alpha x_1 + \beta x_2$ is subgaussian with parameter $\sqrt{\alpha^2 s_1^2 + \beta^2 s_2^2}$ for any $\alpha, \beta \in \mathbb{R}$.

In the following description, unless otherwise noted, all operations are performed over $\mathbb{Z}_q$. For a value $x \in \mathbb{Z}_q$, we write $|x|$ to denote the absolute value of its canonical representative in the interval $\mathbb{Z} \cap [-q/2, q/2]$.

F.2 The learning-with-errors assumption

We now recall the learning with errors assumption [82]:

Definition 43 (Learning with Errors [82])**.** *Let λ be a security parameter. Let $n = n(\lambda)$ be the lattice dimension, $m = m(\lambda)$ be the number of samples, $q = q(\lambda)$ be a modulus, and $s = s(\lambda)$ be a Gaussian width parameter. Then, the learning with errors (LWE) assumption $\mathsf{LWE}_{n,m,q,s}$ states that the following distributions are computationally indistinguishable:*

$$(\mathbf{A}, \mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T}) \approx_c (\mathbf{A}, \mathbf{u}^\mathsf{T}),$$

where $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m}$, $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow D_{\mathbb{Z},s}^m$, and $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^n$.

The security of our construction will rely on the "extended LWE" assumption [20], which essentially says that LWE holds even if the distinguisher learns a linear combination of the LWE errors. We state the assumption below:

Definition 44 (Extended LWE [20])**.** *Let λ be a security parameter and let $n = n(\lambda)$, $m = m(\lambda)$, $q = q(\lambda)$, and $s = s(\lambda)$ be lattice parameters (as in Definition 43). Then, the extended learning with errors (extLWE) assumption $\mathsf{extLWE}_{n,m,q,s}$ states that for every $\mathbf{x} \in \{0,1\}^m$, the following distributions are computationally indistinguishable:*

$$(\mathbf{A}, \mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T}, \mathbf{e}^\mathsf{T}\mathbf{x}) \approx_c (\mathbf{A}, \mathbf{u}^\mathsf{T}, \mathbf{e}^\mathsf{T}\mathbf{x}),$$

where $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m}$, $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow D_{\mathbb{Z},s}^m$, and $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^n$. More precisely, for an adversary $\mathcal{A}$, we write $\mathsf{Adv}^{(n,m,q,s)}_{\mathsf{extLWE}}[\mathcal{A}]$ to denote the distinguishing advantage of $\mathcal{A}$ for the aforementioned distributions.

Previously, Brakerski et al. [20, Lemmas 4.3, 4.7] showed that hardness of the extended LWE assumption $\mathsf{extLWE}_{n,m,q,s}$ can be based on the hardness of the vanilla LWE assumption $\mathsf{LWE}_{n,m,q,s'}$ for $s' = O(s)$.

F.3 Correctness

Theorem 45 (Correctness of Construction 2)**.** *If $B \geq \sqrt{\lambda N}s$, then Construction 2 is correct.*

Proof. Take any database $x \in \{0,1\}^N$ and index $i \in [N]$. Let $\mathbf{d} = \mathbf{A}\mathbf{x}$ be the digest, $\mathbf{q}^\mathsf{T} = \mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T} + t \cdot \eta_i^\mathsf{T}$ be the query, and $a \leftarrow \mathbf{q}^\mathsf{T}\mathbf{x}$ be the response. Then, we have

$$\begin{aligned} a - \mathbf{s}^\mathsf{T}\mathbf{d} - x_i t &= \mathbf{q}^\mathsf{T}\mathbf{x} - \mathbf{s}^\mathsf{T}\mathbf{d} - x_i t \\ &= \mathbf{s}^\mathsf{T}\mathbf{A}\mathbf{x} + \mathbf{e}^\mathsf{T}\mathbf{x} + t \cdot \eta_i^\mathsf{T}\mathbf{x} - \mathbf{s}^\mathsf{T}\mathbf{A}\mathbf{x} - x_i t \\ &= \mathbf{e}^\mathsf{T}\mathbf{x}. \end{aligned} \tag{2}$$

Since the components of $\mathbf{e}$ are independent discrete Gaussian random variables with parameter s, $\mathbf{e}^\mathsf{T}\mathbf{x}$ is subgaussian with parameter $\|\mathbf{x}\| \cdot s \leq \sqrt{N}s$ since $\mathbf{x} \in \{0,1\}^N$. By Eq. (1),

$$\begin{aligned} \Pr[|\mathbf{e}^\mathsf{T}\mathbf{x}| < B : \mathbf{e} \leftarrow D_{\mathbb{Z},s}^N] &\geq \Pr[|\mathbf{e}^\mathsf{T}\mathbf{x}| \leq \sqrt{\lambda N}s : \mathbf{e} \leftarrow D_{\mathbb{Z},s}^N] \\ &= 1 - \mathsf{negl}(\lambda). \end{aligned} \tag{3}$$

To complete the proof, we show that $|a - \mathbf{s}^\mathsf{T}\mathbf{d} - (1 - x_i)t| \geq B$. By Eq. (2),

$$|a - \mathbf{s}^\mathsf{T}\mathbf{d} - (1 - x_i)t| = |\mathbf{e}^\mathsf{T}\mathbf{x} + (1 - 2x_i)t|.$$

By Eq. (1), $|\mathbf{e}^\mathsf{T}\mathbf{x}| < B$ with overwhelming probability. Since $1 - 2x_i \in \{-1, 1\}$ and $t \in [2B, q - 2B]$, with overwhelming probability over the choice of $\mathbf{e}$, we have $\mathbf{e}^\mathsf{T}\mathbf{x} + (1 - 2x_i)t \in [B, q - B]$, or equivalently, $|\mathbf{e}^\mathsf{T}\mathbf{x} + (1 - 2x_i)t| \geq B$. □

F.4 A key lemma

Lemma 46. *Let λ be a security parameter, $\mathbf{x} \in \{0,1\}^N$ be a database, $i \in [N]$ be an index, and $\mathcal{A}$ be an adversary. Consider Construction 2 and define distributions $D^{(0)}_{\mathcal{A},x,i,\lambda}$, $D^{(1)}_{\mathcal{A},x,i,\lambda}$:*

***Distribution** $D^{(0)}_{\mathcal{A},x,i,\lambda}$*

$\mathbf{d} \leftarrow \mathsf{Digest}(1^\lambda, \mathbf{x})$
$(\mathsf{st}, \mathbf{q}) \leftarrow \mathsf{Query}(\mathbf{d}, i)$
$(\mathsf{st}_\mathcal{A}, a^*) \leftarrow \mathcal{A}(\mathbf{d}, \mathbf{x}, \mathbf{q})$
$x'_i \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a^*)$
return x'_i

***Distribution** $D^{(1)}_{\mathcal{A},x,i,\lambda}$*

$\mathbf{d} \leftarrow \mathsf{Digest}(1^\lambda, \mathbf{x})$
$\mathbf{q} \xleftarrow{R} \mathbb{Z}_q^N, \mathbf{e} \leftarrow D_{\mathbb{Z},s}^N$
$(\mathsf{st}_\mathcal{A}, a^*) \leftarrow \mathcal{A}(\mathbf{d}, \mathbf{x}, \mathbf{q})$
$t \xleftarrow{R} [2B, q - 2B]$
$\mathbf{u}^\mathsf{T} \leftarrow \mathbf{q}^\mathsf{T} - t \cdot \eta_i^\mathsf{T}$
$\hat{a}^* \leftarrow a^* - \mathbf{u}^\mathsf{T}\mathbf{x} + \mathbf{e}^\mathsf{T}\mathbf{x}$
if $|\hat{a}^*| < B$ **then**
  $x'_i \leftarrow 0$
elseif $|\hat{a}^* - t| < B$ **then**
  $x'_i \leftarrow 1$
else $x'_i \leftarrow \perp$
return x'_i

Suppose the $\mathsf{extLWE}_{n,N,q,s}$ *assumption holds and H is modeled as a random oracle. Then, for every database length $N = N(\lambda)$, database $x \in \{0,1\}^N$, index $i \in [N]$, and every adversary $\mathcal{A}$ running in time $t = t(\lambda)$, there exists an adversary $\mathcal{B}$ running in time* $\mathsf{poly}(t)$ *such that*

$$|\Pr[D^{(0)}_{\mathcal{A},x,i,\lambda} = 1] - \Pr[D^{(1)}_{\mathcal{A},x,i,\lambda} = 1]| \leq \mathsf{Adv}^{(n,N,q,s)}_{\mathsf{extLWE}}[\mathcal{B}].$$

Proof. Fix a database $\mathbf{x} \in \{0,1\}^N$, an index $i \in [N]$, and any efficient adversary $\mathcal{A}$. In the following analysis, we write $\mathbf{a}_i \in \mathbb{Z}_q^n$ to denote $H(i)$ and we model H as a random oracle (which the reduction algorithm is allowed to program [13]). We now define a sequence of hybrid experiments:

- H_0: This is the distribution $D^{(0)}_{\mathcal{A},\mathbf{x},i,\lambda}$. In this distribution, the output x'_i is computed via $x'_i \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a^*)$.
- H_1: Same as H_0 except the challenger changes how x'_i is computed. Instead of computing $x'_i \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a^*)$, the challenger sets x'_i as follows:
  - If $|a^* - (\mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T})\mathbf{x} + \mathbf{e}^\mathsf{T}\mathbf{x} - kt| < B$ for $k \in \{0,1\}$, then $x'_i \leftarrow k$.
  - Otherwise, the challenger sets $x'_i \leftarrow \perp$.
- H_2: Same as H_1 except the challenger replaces $\mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T}$ with a uniform random vector $\mathbf{u}^\mathsf{T} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^N$. Specifically, the challenger computes $\mathbf{q}^\mathsf{T} \leftarrow \mathbf{u}^\mathsf{T} + t \cdot \eta_i^\mathsf{T}$ and x'_i as follows:
  - If $|a^* - \mathbf{u}^\mathsf{T}\mathbf{x} + \mathbf{e}^\mathsf{T}\mathbf{x} - kt| < B$ for $k \in \{0,1\}$, then $x'_i \leftarrow k$.
  - Otherwise, the challenger sets $x'_i \leftarrow \perp$.
- H_3: Same as H_2 except the challenger samples $\mathbf{q} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^N$. Then, *after* the adversary outputs the response a^*, it samples $t \xleftarrow{\mathrm{R}} [2B, q-2B]$ and sets $\mathbf{u}^\mathsf{T} \leftarrow \mathbf{q}^\mathsf{T} - t \cdot \eta_i^\mathsf{T}$. The response a'_i is computed exactly as in H_2. This is the distribution $D^{(1)}_{\mathcal{A},x,i,\lambda}$.

To complete the proof, we now show that each adjacent pair of distributions is indistinguishable.

- Hybrids H_0 and H_1 are identical distributions. In both experiments, $\mathbf{d} = \mathbf{A}\mathbf{x}$, $\mathbf{q}^\mathsf{T} = \mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T} + t \cdot \eta_i^\mathsf{T}$ and $\mathsf{st} = (\mathbf{d}, \mathbf{s}, t)$. Let a^* be the adversary's response in H_0 and consider the value of $x'_i \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a^*)$. Let $z = a^* - \mathbf{s}^\mathsf{T}\mathbf{d}$. Then,

$$\begin{aligned} z = a^* - \mathbf{s}^\mathsf{T}\mathbf{d} &= a^* - \mathbf{s}^\mathsf{T}\mathbf{A}\mathbf{x} \\ &= a^* - (\mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T})\mathbf{x} + \mathbf{e}^\mathsf{T}\mathbf{x} \end{aligned}$$

  In H_0, the challenger outputs $k \in \{0,1\}$ if $|a^* - \mathbf{s}^\mathsf{T}\mathbf{d} - kt| = |z - kt| < B$ and $\perp$ otherwise. By the above calculation, this precisely coincides with the procedure in H_1.
- Hybrids H_1 and H_2 are computationally indistinguishable under the $\mathsf{extLWE}_{n,N,q,s}$ assumption and modeling H as a random oracle. To see this, suppose there exists an efficient adversary $\mathcal{A}$ that is able to distinguish hybrids H_1 and H_2 with non-negligible advantage. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks the extended LWE assumption:
  1. At the beginning of the game, algorithm $\mathcal{B}$ receives an extended LWE challenge $(\mathbf{A}, \mathbf{z}^\mathsf{T}, y)$ where $\mathbf{A} \in \mathbb{Z}_q^{n \times N}$, $\mathbf{z} \in \mathbb{Z}_q^N$, and $y \in \mathbb{Z}_q$.
  2. Let $\mathbf{a}_1, \ldots, \mathbf{a}_N \in \mathbb{Z}_q^n$ be the columns of $\mathbf{A}$. Algorithm $\mathcal{B}$ programs the random oracle $H(i) \mapsto \mathbf{a}_i$ for each $i \in [N]$. If $\mathcal{A}$ ever queries H on an input $k \notin N$, algorithm $\mathcal{B}$ samples a random $\mathbf{r}_k \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ and defines the mapping $H(k) \mapsto \mathbf{r}_k$.
  3. Algorithm $\mathcal{B}$ now constructs the digest $\mathbf{d} \leftarrow \mathbf{A}\mathbf{x}$ as in H_1 and H_2. To construct the query, algorithm $\mathcal{B}$ samples $t \xleftarrow{\mathrm{R}} [2B, q-2B]$ and sets $\mathbf{q}^\mathsf{T} \leftarrow \mathbf{z}^\mathsf{T} + t \cdot \eta_i^\mathsf{T}$. It gives the digest $\mathbf{d}$, the database $\mathbf{x}$, and the query $\mathbf{q}$ to $\mathcal{A}$.
  4. Algorithm $\mathcal{A}$ outputs a response a^*. Algorithm $\mathcal{B}$ computes x'_i as follows:
     - If $|a^* - \mathbf{z}^\mathsf{T}\mathbf{x} + y - kt| < B$ for $k \in \{0,1\}$, then $x'_i \leftarrow k$.
     - Otherwise, $x'_i \leftarrow \perp$.
  5. Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with x'_i and outputs whatever $\mathcal{A}$ outputs.

  Since $\mathbf{A} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times N}$, the outputs of the random oracle are correctly simulated. Corresponding, algorithm $\mathcal{B}$ perfectly simulates the distribution of the digest $\mathbf{d}$ for $\mathcal{A}$. We now consider the two possible challenge distributions:
  - Suppose $\mathbf{z}^\mathsf{T} = \mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T}$ and $y = \mathbf{e}^\mathsf{T}\mathbf{x}$. Then the query $\mathbf{q}$ and the response x'_i are distributed exactly as in H_1.
  - Suppose $\mathbf{z}^\mathsf{T} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^N$ and $y = \mathbf{e}^\mathsf{T}\mathbf{x}$. Then, the query $\mathbf{q}$ and the response x'_i are distributed exactly as in H_2.

  We conclude that algorithm $\mathcal{B}$ breaks the extended LWE assumption with the same distinguishing advantage as $\mathcal{A}$ and the claim follows. More precisely, we can write $\mathrm{H}_i(\mathcal{A})$ to denote the output of a distinguisher $\mathcal{A}$ on input a sample from H_i. Then our reduction shows that for all adversaries $\mathcal{A}$ running in time t, there exists an adversary $\mathcal{B}$ running in time $\mathsf{poly}(t)$ such that

$$\mathsf{Adv}^{n,N,q,s}_{\mathsf{extLWE}}[\mathcal{B}] \geq |\Pr[\mathrm{H}_1[\mathcal{A}] = 1] - \Pr[\mathrm{H}_2[\mathcal{A}] = 1]|.$$

- Hybrids H_2 and H_3 are identically distributed. In H_2, $\mathbf{q} = \mathbf{u} + t \cdot \eta_i$ where $\mathbf{u} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^N$ and $\mathbf{u}$ is sampled independently of all other quantities. Thus, the distribution of $\mathbf{q}$ in H_2 is uniform over $\mathbb{Z}_q^N$, which matches the distribution in H_3. In both experiments, $\mathbf{u} = \mathbf{q} - t \cdot \eta_i$, where $t \xleftarrow{\mathrm{R}} [2B, q - 2B]$. □

F.5 Integrity

Theorem 47 (Integrity of Construction 2)**.** *Suppose the* $\mathsf{extLWE}_{n,N,q,s}$ *assumption holds and H is modeled as a random oracle. Then, Construction 2 (instantiated with parameters n, N, q, s, B and hash function H) has integrity error at most $\varepsilon = (2B-1)/(q-4B+1)$.*

Proof. Fix a database $\mathbf{x} \in \{0,1\}^N$, an index $i \in [N]$, and any efficient adversary $\mathcal{A}$. We now define a sequence of hybrid experiments:

- H_0: This is the real integrity game.
- H_1: Same as H_0 except the challenger samples $\mathbf{q} \xleftarrow{R} \mathbb{Z}_q^N$ and $\mathbf{e} \leftarrow D_{\mathbb{Z},s}^N$. Then, after the adversary outputs the response a^*, the challenger samples $t \xleftarrow{R} [2B, q-2B]$ and sets $\mathbf{u}^\mathsf{T} \leftarrow \mathbf{q}^\mathsf{T} - t \cdot \boldsymbol{\eta}_i^\mathsf{T}$. If $|a^* - \mathbf{u}^\mathsf{T}\mathbf{x} + \mathbf{e}^\mathsf{T}\mathbf{x} - kt| < B$ for some $k \in \{0,1\}$, then the challenger sets $x_i' \leftarrow k$. Otherwise, the challenger sets $x_i' \leftarrow \perp$.
- H_2: Same as H_1 except the challenger changes how it computes x_i':
  - If $|a^* - \mathbf{u}^\mathsf{T}\mathbf{x} + \mathbf{e}^\mathsf{T}\mathbf{x} - x_i t| < B$, then $x_i' \leftarrow x_i$.
  - Otherwise, the challenger sets $x_i' \leftarrow \perp$.

  Specifically, in H_2, it is guaranteed that $x_i' \in \{x_i, \perp\}$.

We now show that the outputs of each adjacent pair of hybrid distributions are computationally indistinguishable:

- Hybrids H_0 and H_1 are computationally indistinguishable by Lemma 46.
- The statistical distance between H_1 and H_2 is at most $(2B+1)/(q-4B+1)$. By construction, the two experiments are identical unless

$$|a^* - \mathbf{u}^\mathsf{T}\mathbf{x} + \mathbf{e}^\mathsf{T}\mathbf{x} - (1-x_i)t| < B. \tag{4}$$

  Now, $\mathbf{u} = \mathbf{q} - t \cdot \boldsymbol{\eta}_i$, so

$$a^* - \mathbf{u}^\mathsf{T}\mathbf{x} + \mathbf{e}^\mathsf{T}\mathbf{x} - (1-x_i)t = a^* - \mathbf{q}^\mathsf{T}\mathbf{x} + \mathbf{e}^\mathsf{T}\mathbf{x} - (1-2x_i)t.$$

  Since $1 - 2x_i \in \{-1, 1\}$, there are at most $2B-1$ values of $t \in \mathbb{Z}_q$ for which Eq. (4) holds. Since t is sampled uniformly at random from a set of size $q-4B+1$ and independently of a^*, $\mathbf{u}$, $\mathbf{x}$, and $\mathbf{e}$, the probability that t lands in the interval of size $2B-1$ is at most $(2B-1)/(q-4B+1)$. Correspondingly, the statistical distance between H_1 and H_2 is $(2B-1)/(q-4B+1)$.

By construction, the output x_i' in H_2 is guaranteed to be either x_i or $\perp$. By a hybrid argument, in the real integrity game H_0, it must be the case that

$$\Pr[x_i' \in \{x_i, \perp\}] \leq \frac{2B-1}{q-4B+1} + \mathsf{negl}(\lambda),$$

which proves the claim. □

F.6 Privacy

Theorem 48 (Privacy of Construction 2)**.** *Suppose the* $\mathsf{extLWE}_{n,N,q,s}$ *assumption holds and H is modeled as a random oracle. Then, Construction 2 (instantiated with parameters n, N, q, s, B and hash function H) provides privacy. More precisely, for every adversary running in time $t = t(\lambda)$, there exists an adversary $\mathcal{B}$ running in time* $\mathsf{poly}(t)$ *such that*

$$|\Pr[\mathrm{REAL}_{\mathcal{A},x,i,\lambda} = 1] - \Pr[\mathrm{IDEAL}_{\mathcal{A},\mathcal{S},x,\lambda} = 1]| \leq \mathsf{Adv}^{n,N,q,s}_{\mathsf{extLWE}}[\mathcal{B}],$$

where $\mathrm{REAL}_{\mathcal{A},x,i,\lambda}$ *and* $\mathrm{IDEAL}_{\mathcal{A},\mathcal{S},x,\lambda}$ *are the distributions defined in Definition 38.*

Proof. Fix a database $\mathbf{x} \in \{0,1\}^N$, an index $i \in [N]$, and any efficient adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$. We construct an efficient simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ as follows:

Simulator $\mathcal{S}_0\left(1^\lambda, \mathbf{d}, \mathbf{x}\right)$	**Simulator** $\mathcal{S}_1(\mathsf{st}_\mathcal{S}, a^*)$
$\mathbf{q} \xleftarrow{R} \mathbb{Z}_q^N, \mathbf{e} \leftarrow D_{\mathbb{Z},s}^N$	if $\lvert a^* - \mathsf{st}_\mathcal{S}\rvert < B$, $b \leftarrow 1$
$\mathsf{st}_\mathcal{S} \leftarrow \mathbf{q}^\mathsf{T}\mathbf{x} - \mathbf{e}^\mathsf{T}\mathbf{x}$	else, sample $t \xleftarrow{R} [2B, q-2B]$
return $(\mathsf{st}_\mathcal{S}, \mathbf{q})$	and $b \leftarrow \mathbb{1}\{\lvert a^* - \mathsf{st}_\mathcal{S} - t\rvert < B\}$
	return b

We show that the real distribution $\mathrm{REAL}_{\mathcal{A},\mathbf{x},i,\lambda}$ and ideal distribution $\mathrm{IDEAL}_{\mathcal{A},\mathcal{S},\mathbf{x},\lambda}$ are computationally indistinguishable. We define a sequence of hybrid experiments:

- H_0: This is the real distribution $\mathrm{REAL}_{\mathcal{A},\mathbf{x},i,\lambda}$. In this distribution, the response x_i' is computed via $x_i' \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a^*)$.
- H_1: Same as H_0 except the challenger samples $\mathbf{q} \xleftarrow{R} \mathbb{Z}_q^N$ and $\mathbf{e} \leftarrow D_{\mathbb{Z},s}^N$. Then, after the adversary outputs the response a^*, the challenger samples $t \xleftarrow{R} [2B, q-2B]$ and sets $\mathbf{u}^\mathsf{T} \leftarrow \mathbf{q}^\mathsf{T} - t \cdot \boldsymbol{\eta}_i^\mathsf{T}$. If $|a^* - \mathbf{u}^\mathsf{T}\mathbf{x} + \mathbf{e}^\mathsf{T}\mathbf{x} - kt| < B$ for some $k \in \{0,1\}$, then the challenger sets $x_i' \leftarrow k$. Otherwise, the challenger sets $x_i' \leftarrow \perp$.
- H_2: Same as H_1, except instead of computing x_i', the challenger sets $b = 1$ if $|a^* - \mathbf{q}^\mathsf{T}\mathbf{x} + \mathbf{e}^\mathsf{T}\mathbf{x}| < B$. Otherwise, it samples $t \xleftarrow{R} [2B, q-2B]$ and sets $b \leftarrow \mathbb{1}\{|a^* - \mathbf{q}^\mathsf{T}\mathbf{x} + \mathbf{e}^\mathsf{T}\mathbf{x} - t| < B\}$ This is the ideal distribution $\mathrm{IDEAL}_{\mathcal{A},\mathcal{S},\mathbf{x},\lambda}$.

To complete the proof, we now show that each adjacent pair of distributions is indistinguishable. First, hybrids H_0 and H_1 are computationally indistinguishable by Lemma 46. To complete the proof, we show that H_1 and H_2 are identically distributed:

- Hybrids H_1 and H_2 are identically distributed. Let $a^* \in \mathbb{Z}_q$ be the adversary's response in the two experiments. Define the quantity $z = a^* - \mathbf{q}^\mathsf{T}\mathbf{x} + \mathbf{e}^\mathsf{T}\mathbf{x}$. We consider two possibilities:

– Suppose $|z| < B$. In H_2, the challenger always sets $b = 1$. We claim this is also the case in H_1. By construction, we can first write

$$\mathbf{u}^\top \mathbf{x} = \mathbf{q}^\top \mathbf{x} - t \cdot \eta_i^\top \mathbf{x} = \mathbf{q}^\top \mathbf{x} - x_i t. \tag{5}$$

This means

$$|z| = \left|a^* - \mathbf{q}^\top \mathbf{x} + \mathbf{e}^\top \mathbf{x}\right| = \left|a^* - \mathbf{u}^\top \mathbf{x} + \mathbf{e}^\top \mathbf{x} - x_i t\right|. \tag{6}$$

Since $x_i \in \{0,1\}$, we have $x_i' = x_i$ and $b = 1$ in H_1.

– Suppose $|z| \geq B$. In this case, the challenger in H_2 samples $t \xleftarrow{R} [2B, q-2B]$ and sets $b = 1$ if $|z-t| < B$ and $b = 0$ otherwise. Consider the challenger's behavior in H_1. By Eq. (6), we have that $b = 1$ only if

$$\left|a^* - \mathbf{u}^\top \mathbf{x} + \mathbf{e}^\top \mathbf{x} - (1-x_i)t\right| < B.$$

By Eq. (5), this is equivalent to $|z-(1-2x_i)t| < B$. Like in H_2, the challenger in H_1 samples $t \xleftarrow{R} [2B, q-2B]$ *after* the adversary outputs a^*. We consider two possibilities:

* If $x_i = 0$, then $1-2x_i = 1$, and the challenger in H_1 sets $b \leftarrow \mathbb{1}\{|z-t| < B\}$. This is identical to the behavior in H_2.
* If $x_i = 1$, then $1-2x_i = -1$, and the challenger in H_1 sets $b \leftarrow \mathbb{1}\{|z+t| < B\}$. Since $t \xleftarrow{R} [2B, q-2B]$ the distributions of t mod q and $-t$ mod q are identical (the interval is symmetric about 0 over $\mathbb{Z}_q$). Since t and z are independent, the distribution of $\mathbb{1}\{|z+t| < B\}$ is identically distributed as that of $\mathbb{1}\{|z-t| < B\}$. Once again, the distribution of b in H_1 is distributed identically to that in H_2.

We conclude that the distribution of b is identical in H_1 and H_2 in this case. □

G Single-server authenticated PIR from DDH

We now analyze our DDH-based single-server scheme (Construction 3). Correctness follows by construction. Therefore, this section focuses on security.

G.1 Decisional Diffie-Hellman assumption

We first recall the decisional Diffie-Hellman assumption:

Definition 49 (Decisional Diffie-Hellman)**.** *Let λ be a security parameter and let $\mathbb{G}$ be a group of prime order p where $1/p = \mathsf{negl}(\lambda)$. Let g be a generator of $\mathbb{G}$. We say that the decisional Diffie-Hellman assumption (DDH) holds in $\mathbb{G}$ if the following distributions are computationally indistinguishable:*

$$(g, h, g^x, h^x) \approx_c (g, h, g^x, z)$$

where $h, z \xleftarrow{R} \mathbb{G}$ and $x \xleftarrow{R} \mathbb{Z}_p$.

By a random self-reduction [72, 85], it is straightforward to show that if the DDH assumption holds in $\mathbb{G}$, then for all polynomials $N = N(\lambda)$, the following distributions are also computationally indistinguishable:

$$(g, h_1, h_1^r, \ldots, h_N, h_N^r) \approx_c (g, h_1, z_1, \ldots, h_N, z_N), \tag{7}$$

where $h_1, \ldots, h_N, z_1, \ldots, z_N \xleftarrow{R} \mathbb{G}$ and $r \xleftarrow{R} \mathbb{Z}_p$.

G.2 A key lemma

To analyze the security (and integrity) of Construction 3, we start by proving the following lemma, which will feature in both the security and the integrity analysis.

Lemma 50. *Let λ be a security parameter, $x \in \{0,1\}^N$ be a database, $i \in [N]$ be an index, and $\mathcal{A}$ be an adversary. Consider Construction 3 and define distributions $D^{(0)}_{\mathcal{A},x,i,\lambda}$ and $D^{(1)}_{\mathcal{A},x,i,\lambda}$:*

***Distribution** $D^{(0)}_{\mathcal{A},x,i,\lambda}$*

1: $d \leftarrow \mathsf{Digest}(1^\lambda, x)$
2: $(\mathsf{st}, q) \leftarrow \mathsf{Query}(d, i)$
3: $(\mathsf{st}_{\mathcal{A}}, a^*) \leftarrow \mathcal{A}(d, x, q)$
4: $x_i' \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a^*)$
5: **return** x_i'

***Distribution** $D^{(1)}_{\mathcal{A},x,i,\lambda}$*

1: $d \leftarrow \mathsf{Digest}(1^\lambda, x)$
2: $q \xleftarrow{R} \mathbb{G}^N$
3: $(\mathsf{st}_{\mathcal{A}}, a^*) \leftarrow \mathcal{A}(d, x, q)$
4: **if** $a^* = \prod_{j \in [N]} q_j^{x_j}$ **then** $x_i' \leftarrow x_i$
5: **else** $x_i' \leftarrow \perp$
6: **return** x_i'

Suppose the DDH assumption holds in $\mathbb{G}$ and H is modeled as a random oracle. Then, for every database length $N = N(\lambda)$, database $x \in \{0,1\}^N$, index $i \in [N]$, and efficient adversary $\mathcal{A}$,

$$\left|\Pr[D^{(0)}_{\mathcal{A},x,i,\lambda} = 1] - \Pr[D^{(1)}_{\mathcal{A},x,i,\lambda} = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Take any database length $N = N(\lambda)$, database $x \in \{0,1\}^N$ and an index $i \in [N]$. We show that the distributions $D^{(0)}_{\mathcal{A},x,i,\lambda}$ and $D^{(1)}_{\mathcal{A},x,i,\lambda}$ are computationally indistinguishable. In the following analysis, we write $h_i \in \mathbb{G}$ to denotes $H(i)$, and we model H as a random oracle (which the reduction algorithm is allowed to program) [13]. We now define a sequence of hybrid experiments:

- H_0: This is the distribution $D^{(0)}_{\mathcal{A},x,i,\lambda}$. In this distribution, the response $x_i' \in \{0,1,\perp\}$ is computed via $x_i' \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a^*)$.
- H_1: Same as H_0, except the challenger changes how x_i' is computed. Instead of computing $x_i' \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a^*)$, the challenger sets x_i' as follows:
  – If $a^* = h_i^{yt} (h_i^r)^{x_i} \prod_{j \neq i} (h_j^r)^{x_j}$ for $y \in \{0,1\}$, then $x_i' \leftarrow y$.

– Otherwise, the challenger sets $x'_i \leftarrow \perp$.

- H_2: Same as H_1, except the challenger replaces the tuple of group elements $(g, h_1, h_1^r, \ldots, h_N, h_N^r)$ with $(g, h_1, z_1, \ldots, h_N, z_N)$ where $z_1, \ldots, z_N \xleftarrow{R} \mathbb{G}$ and $r \xleftarrow{R} \mathbb{Z}_p$. Specifically, the challenger constructs the query $q = (q_1, \ldots, q_N)$ by setting $q_j \leftarrow z_j$ for $j \neq i$ and $q_i \leftarrow z_i h_i^t$. When computing x'_i, the challenger proceeds as follows:
  – If $a^* = h_i^{yt} z_i^{x_i} \prod_{j \neq i} z_j^{x_j}$ for $y \in \{0,1\}$, then $x'_i \leftarrow y$.
  – Otherwise, the challenger sets $x'_i \leftarrow \perp$.
- H_3: Same as H_2 except the challenger samples $q \xleftarrow{R} \mathbb{G}^N$. Then, *after* the adversary outputs the response a^*, it sets $z_j = q_j$ for all $j \neq i$ and $z_i \leftarrow q_i / h_i^t$ where $t \xleftarrow{R} \mathbb{Z}_p$. The response a'_i is computed exactly as in H_2.
- H_4: Same as H_2 except the challenger again changes how it computes x'_i:
  – If $a^* = h_i^{x_i t} z_i^{x_i} \prod_{j \neq i} z_j^{x_j}$, then $x'_i \leftarrow x_i$.
  – Otherwise, the challenger sets $x'_i \leftarrow \perp$.
- H_5: Same as H_4, except the the challenger sets $x'_i \leftarrow x_i$ if $a^* = \prod_{j \in [N]} q_j^{x_j}$ and $x'_i \leftarrow \perp$ otherwise. This is the distribution $D^{(1)}_{\mathcal{A},x,i,\lambda}$.

To complete the proof, we now show that each adjacent pair of distributions are indistinguishable:

- Hybrids H_0 and H_1 are identical distributions. In both experiments, $d = \prod_{j \in [N]} h_j^{x_j}$, $q = (q_1, \ldots, q_N)$, and $\mathsf{st} = (i, d, r, t)$, where $q_j = h_j^r$ for $j \neq i$ and $q_i = h_i^{r+t}$ for some $r, t \in \mathbb{Z}_p$. Let a^* be the adversary's response in H_0, and consider the value of $x'_i \leftarrow \mathsf{Reconstruct}(\mathsf{st}, a^*)$ in H_0:
  – If $a^* = d^r$, then $x'_i = 0$. If $a^* = d^r h_i^t$, then $x'_i = 1$. This is equivalent to setting $x'_i = y \in \{0,1\}$ if $a^* = d^r h_i^{yt}$. Substituting in the above relations, this means that in H_0, $x'_i = y \in \{0,1\}$ if
$$a^* = d^r h_i^{yt} = \left(\prod_{j \in [N]} h_j^{x_j} \right)^r h_i^{yt} = h_i^{yt} (h_i^r)^{x_i} \prod_{j \neq i} (h_j^r)^{x_i}.$$
  – Otherwise $x'_i = \perp$.

  This is precisely the distribution of x'_i in H_1.
- Hybrids H_1 and H_2 are computationally indistinguishable under the DDH assumption and modeling H as a random oracle. To see this, suppose there exists an efficient adversary $\mathcal{A}$ that is able to distinguish hybrids H_1 and H_2 with non-negligible probability. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that distinguishes the distributions in Eq. (7):
  1. At the beginning of the game, algorithm $\mathcal{B}$ receives a challenge vector $(g, h_1, T_1, \ldots, h_N, T_N)$.
  2. Algorithm $\mathcal{B}$ programs the random oracle $H(i) \mapsto h_i$ for each $i \in [N]$. If $\mathcal{A}$ ever queries H on an input $k \notin [N]$, algorithm $\mathcal{B}$ samples a random $r_k \xleftarrow{R} \mathbb{G}$ and defines the mapping $H(k) \mapsto r_k$.
  3. Algorithm $\mathcal{B}$ now constructs the digest $d \leftarrow \prod_{j \in [N]} h_j^{x_j}$ as in H_1 and H_2. To construct the query, algorithm $\mathcal{B}$ sets $q_j \leftarrow T_j$ for $j \neq i$ and $q_i \leftarrow T_i h_i^t$ where $t \xleftarrow{R} \mathbb{Z}_p$. It gives the digest d, the database x, and the query q to $\mathcal{A}$.
  4. Algorithm $\mathcal{A}$ outputs a response a^*. Algorithm $\mathcal{B}$ computes x'_i as follows:
     – If $a^* = h_i^{yt} T_i^{x_i} \prod_{j \neq i} T_j^{x_j}$, then $x'_i \leftarrow x_i$.
     – Otherwise, $x'_i \leftarrow \perp$.
  5. Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with x'_i and outputs whatever $\mathcal{A}$ outputs.

  Since $h_1, \ldots, h_N \xleftarrow{R} \mathbb{G}$, the outputs of the random oracle are correctly simulated. Correspondingly, algorithm $\mathcal{B}$ perfectly simulates the distribution of the digest d for $\mathcal{A}$. We now consider the two possible challenge distributions:
  – Suppose $T_i = h_i^r$ for all $i \in [N]$ and where $r \xleftarrow{R} \mathbb{Z}_p$. In this case, $q_j = h_j^r$ for all $j \neq i$ and $q_j = h_i^{r+t}$ where $t \xleftarrow{R} \mathbb{Z}_p$. Similarly, $x'_i = y \in \{0,1\}$ if $a^* = h_i^{yt} (h_i^r)^{x_i} \prod_{j \neq i} (h_j^r)^{x_j}$, which exactly matches the distribution in H_1.
  – Suppose $T_i = z_i \xleftarrow{R} \mathbb{G}$ for all $i \in [N]$. In this case, $q_j = z_j$ for all $j \neq i$ and $q_j = z_i h_i^t$ where $t \xleftarrow{R} \mathbb{Z}_p$. This is the query distribution in H_2. Similarly, to compute the response x'_i, algorithm $\mathcal{B}$ sets $x'_i = y \in \{0,1\}$ if $a^* = h_i^{yt} z_i^{x_i} \prod_{j \neq i} z_j^{x_j}$, which matches the distribution in H_2.

  We conclude that algorithm $\mathcal{B}$ distinguishes between the distributions in Eq. (7) with the same distinguishing advantage as $\mathcal{A}$, and the claim follows.
- Hybrids H_2 and H_3 are identically distributed. In H_2, the z_j's are sampled uniformly and independently from $\mathbb{G}$ (and also independent of $h_1, \ldots, h_N, t$). Thus, the distribution of $q = (q_1, \ldots, q_N)$ in H_2 is identical to that in H_3. Finally, in H_2, $q_i = z_i h_i^t$, where $t \xleftarrow{R} \mathbb{Z}_p$. This is the distribution in H_3.
- The statistical distance between H_3 and H_4 is $1/p = \mathsf{negl}(\lambda)$. By construction, the two experiments are identical unless the adversary outputs a^* where $a^* = h_i^{(1-x_i)t} z_i^{x_i} \prod_{j \neq i} z_j^{x_j}$. Using the relation $z_i = q_i / h_i^t$, this becomes
$$a^* = h_i^{(1-x_i)t} \frac{q_i^{x_i}}{h_i^{x_i t}} \prod_{j \neq i} z_j^{x_j} = (h_i^t)^{1-2x_i} q_i^{x_i} \prod_{j \neq i} z_j^{x_j},$$
  or equivalently, if
$$(h_i^t)^{1-2x_i} = \frac{a^*}{q_i^{x_i} \prod_{j \neq i} z_j^{x_j}}. \tag{8}$$
  Now, in H_3 and H_4, the challenger samples $t \xleftarrow{R} \mathbb{Z}_p$ *after*

the adversary outputs a^*. Moreover, since $x_i \in \{0,1\}$, it follows that $1-2x_i \in \{-1,1\}$. Since t is sampled independently of a^*, q_i and z_j for all $j \in [N]$, and h_i is a generator of $\mathbb{G}$ (with overwhelming probability), Eq. (8) holds with probability at most $1/p = \mathsf{negl}(\lambda)$ over the randomness of t.

- Hybrids H_4 and H_5 are identical experiments. In H_4, the challenger sets $x'_i = x_i$ if and only if

$$a^* = h_i^{x_i t} z_i^{x_i} \prod_{j \neq i} z_j^{x_j} = (z_i h_i^t)^{x_i} \prod_{j \neq i} z_j^{x_j} = \prod_{j \in [N]} q_j^{x_j},$$

since $q_j = z_j$ for all $j \neq i$ and $q_i = z_i h_i^t$. This is the distribution in H_5. □

G.3 Integrity

Theorem 51 (Integrity of Construction 3)**.** *Suppose the DDH assumption holds in $\mathbb{G}$ and H is modeled as a random oracle. Then, Construction 3 (instantiated with group $\mathbb{G}$ and hash function H) provides integrity.*

Proof. Fix a database $x \in \{0,1\}^N$ and an index $i \in [N]$, and take any efficient adversary $\mathcal{A}$ for the integrity game. We define the following hybrid experiments:

- H_0: This is the real integrity game.
- H_1: Same as H_0, except the challenger samples $q \xleftarrow{R} \mathbb{G}^N$ and sets $x'_i \leftarrow x_i$ if $a^* = \prod_{j \in [N]} q_j^{x_j}$ and $x'_i \leftarrow \perp$ otherwise.

The outputs of H_0 and H_1 are computationally indistinguishable by Lemma 50. Next, in H_1, $\Pr[x'_i \notin \{x_i, \perp\}] = 0$ by construction. The claim now follows by a hybrid argument. □

G.4 Privacy

Theorem 52 (Privacy of Construction 3)**.** *Suppose the DDH assumption holds in $\mathbb{G}$ and H is modeled as a random oracle. Then, Construction 3 (instantiated with group $\mathbb{G}$ and hash function H) provides privacy.*

Proof. Fix a database $x \in \{0,1\}^N$ and an index $i \in [N]$. Take any efficient adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$. We construct an efficient simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ as follows:

Simulator $\mathcal{S}_0(1^\lambda, d, x)$	**Simulator** $\mathcal{S}_1(\mathsf{st}_\mathcal{S}, a^*)$
$q = (q_1, \ldots, q_N) \xleftarrow{R} \mathbb{G}^N$	$b \leftarrow \mathbb{1}\{a^* = \mathsf{st}_\mathcal{S}\}$
$\mathsf{st}_\mathcal{S} \leftarrow \prod_{j \in [N]} q_j^{x_j}$	**return** b
return $(\mathsf{st}_\mathcal{S}, q)$	

We show that the real distribution $\text{REAL}_{\mathcal{A},x,i,\lambda}$ and ideal distribution $\text{IDEAL}_{\mathcal{A},\mathcal{S},x,\lambda}$ are computationally indistinguishable. We define a sequence of hybrid experiments:

- H_0: This is the real distribution $\text{REAL}_{\mathcal{A},x,i,\lambda}$.
- H_1: Same as H_0, except the challenger samples $q \xleftarrow{R} \mathbb{G}^N$ and sets $x'_i \leftarrow x_i$ if $a^* = \prod_{j \in [N]} q_j^{x_j}$ and $x'_i \leftarrow \perp$ otherwise.
- H_2: This is the ideal distribution $\text{IDEAL}_{\mathcal{A},\mathcal{S},x,\lambda}$.

We now argue that adjacent pair of hybrid experiments are indistinguishable:

- H_0 and H_1 are computationally indistinguishable by Lemma 50.
- H_1 and H_2 are identical experiments. Namely, in H_2, the challenger sets $b = 1$ if and only if $a^* = \prod_{j \in [N]} q_j^{x_j}$, which coincides with the behavior in H_1. □

G.5 Handling larger database rows

Our DDH-based construction (Construction 3) directly supports (small) multi-bit database records with no communication overhead. The cost is the client's computational cost increases by a factor of $2^{\ell/2}$, where ℓ is the bit-length of the record.

The idea is simple. Suppose the database consists of N ℓ-bit records $x_1, \ldots, x_\ell \in \{0,1\}^\ell$. The digest, query, and answer algorithms are unchanged (the only difference is that instead of each record $x_i \in \{0,1\}$ being a single bit, we now treat each record $x_i \in \{0,1\}^\ell$ as an integer between 0 and $2^\ell - 1$). The only difference is during reconstruction, the client now learns the value $h_i^{x_i t}$. Since the client knows the blinding factor t, it can exponentiate with $t^{-1} \bmod p$ to obtain $h_i^{x_i}$. Namely, the client is able to obtain an encoding of the database record in the exponent. Recovering the value of x_i now requires computing a discrete logarithm (base h_i). This can be computed in time $O(\sqrt{2^\ell})$ using Pollard's kangaroo method [80], or alternatively, if ℓ is very small, then the client can precompute a lookup table of possible values for $h_i^{x_i}$. Thus, this approach is suitable for small values of ℓ (e.g., $\ell \leq 32$).

While there are applications for a small-row single-server authenticated PIR scheme, we still hope that it is possible to construct a more bandwidth- and computation-efficient scheme in the future. We unsuccessfully attempted to combine an unauthenticated classic single-server PIR scheme with some sort of algebraic integrity-protection mechanism, but it seems non-trivial to provide our integrity properties while making only black-box use of the underlying single-server PIR scheme. Further investigation along these lines would be an interesting task for future work.

Supporting multi-bit records in the lattice-based setting. We note that a similar approach as above can be applied to the lattice-based construction (Construction 2) to support multi-bit records. While correctness holds, the security analysis is more challenging. Namely, both integrity and privacy of Construction 2 (Theorems 47 and 48) rely on the extended LWE assumption where we require that LWE holds even if the distinguisher is given a linear combination $\mathbf{e}^\mathsf{T}\mathbf{x}$ of the LWE error. When the database entries are binary-valued (i.e.,

Database size N [bits]:	2^{13}	2^{23}	2^{33}
Integrity error $\varepsilon_\Pi = 2^{-64}$	3	4	7
Integrity error $\varepsilon_\Pi = 2^{-128}$	6	9	15

Table 9: Selection of the error correcting code parameter t for different database sizes and integrity errors.

$\mathbf{x} \in \{0,1\}^N$), we can appeal to [20, Lemma 4.3, Claim 4.6, Lemma 4.7] to base hardness on standard LWE. It seems plausible that a similar (possibly less tight) reduction applies when the database $\mathbf{x} \in (\{0,1\}^\ell)^N$ consists of ℓ-bit integers, and this is an interesting question for further exploration.

H Parameter selection

In this section we discuss parameter selection for the scheme that use integrity amplification (Construction 6). As the base authenticated PIR scheme (denoted PIR_0 in Construction 6) we use the LWE-based scheme (Construction 2) with modulus $q = 2^{32}$ and lattice dimension $n = 1100$, which has integrity error $\varepsilon = (2B-1)/(q-4B+1)$ (cf. Theorem 47). The correctness of Construction 2 (Theorem 45) states that $B \leq \sqrt{\lambda N} s$. By Theorem 41 we know that if we use a simple repetition code (which corrects up to t errors by expanding each database bit into $2t+1$ codeword bits) and PIR_0 has integrity error ε, then Construction 6 has integrity error ε^{t+1}. Table 9 shows the choice of t to achieve integrity error ε_Π in Construction 6 for different database sizes N, where N indicates the number of single bit records in the database.

I Artifact Appendix

I.1 Abstract

The source code for our single- and multi-server authenticated-PIR schemes and the Keyd public-key server is available at `https://github.com/dedis/apir-code` under open-source license. The same repository contains unauthenticated-PIR schemes that we implemented as baselines for comparison; as single-server PIR baseline we use the original implementation of SimplePIR [52]. Our implementation and the implementation of SimplePIR use C for the performance-critical functions. We perform all the experiments on machines equipped with two Intel Xeon E5-2680 v3 (Haswell) CPUs, each with 12 cores, 24 threads, and operating at 2.5 GHz, and 256 GB of RAM.

I.2 Description & Requirements

I.2.1 Security, privacy, and ethical concerns

None.

I.2.2 How to access

The source code for all the authenticated-PIR schemes, the classic-PIR schemes and Keyd under which this artifact evaluation was tested is available at `https://github.com/dedis/apir-code/tree/af3202e3776d4cb880256372dd51613ee34532ba`.

I.2.3 Hardware dependencies

We perform all the experiments on machines equipped with two Intel Xeon E5-2680 v3 (Haswell) CPUs, each with 12 cores, 24 threads, and operating at 2.5 GHz. Each machine has 256 GB of RAM, and runs Ubuntu 20.04 and Go 1.17.5. Machines are connected with 10 Gigabit Ethernet. In the experiments for the multi-server schemes and Keyd (Sections 7.1, 7.2 and 7.4), the client and the servers run on separate machines—the experiments use at most six machines. For single-server schemes we use a single machine that runs both client and server. However, it is possible to run the code, together with the accompanying tests, benchmarks and experiments, on any machine equipped the software dependencies listed in the next section.

I.2.4 Software dependencies

Running run the code requires Go (tested with Go 1.17.5 and 1.19.5) and a C compiler (tested with GCC 9.4.0).

Reproducing the evaluation results requires GNU Make, Screen, Python 3[3], Fabric, Tomli, Numpy and Matplotlib.

I.2.5 Benchmarks

None.

I.3 Set-up

I.3.1 Installation

Installation instructions are given in the `Setup` sections of https://github.com/dedis/apir-code/blob/main/README.md and we report them here. To run the code in the repository install Go (tested with Go 1.17.5) and a C compiler (tested with GCC 9.4.0). To reproduce the evaluation results, install GNU Make, Screen, Python 3, Fabric, Numpy and Matplotlib.

I.3.2 Basic Test

To run all basic tests, users should clone the repository, and download the dump of the SKS PGP key directory using the command

```
bash scripts/download_sks_parsed.sh
```

[3]The package `python-is-python3` might be needed.

in the repository's root directory.

To run the basic test, use the following command:

```
go test
```

in the repository's root directory. This command takes about six minutes to run and outputs the time taken by each test. If all the tests pass, the output ends as follows:

```
PASS
ok    github.com/si-co/apir-code
```

I.4 Evaluation workflow

I.4.1 Major Claims

Our paper claims what follows.

Multi-server point queries (Section 7.1).

(C1): The maximum overhead for our multi-server authenticated-PIR scheme for point queries, in comparison with classic unathenticated PIR is 2.9× for user time and 1.8× for bandwidth. This is the outcome of experiment (E1), whose results are presented in Fig. 3. **Update October 19, 2024:** We fix the experiment and update the results as 3× for user time and 1.6× for bandwidth. The evaluation workflow does not change. See Appendix J.2 for more information.

(C2): The impact of the number of servers on our multi-server authenticated-PIR scheme for point queries is almost negligible for user time and imposes a linear increase for bandwidth. This is the result of experiment (E2), whose results are reported in Fig. 4 in the body of the paper.

(C3): The preprocessing cost for our multi-server authenticated PIR scheme for point queries is linear in the database size. This is the result of experiment (E3), whose results are reported in Fig. 8 in Appendix C.

Multi-server complex queries (Section 7.2).

(C4): The user time and bandwidth overheads of the authenticated-PIR schemes for complex queries against classic unauthenticated-PIR schemes are less than 1.1×. This is the outcome of experiment (E4), whose results are presented in Fig. 5.

Single-server point queries (Section 7.3).

(C5): The authenticated-PIR schemes from the decisional Diffie-Hellman assumption (DDH) and from the learning-with-errors assumption (LWE) have integrity error 2^{-128}. The DDH construction has a smaller digest, i.e., lower offline bandwidth, but has twice the online bandwidth of the LWE construction. The LWE construction is also faster (3-79×). The scheme with integrity amplification (LWE$^+$) has integrity error 2^{-64} but the same classic-PIR privacy as SimplePIR [52]. LWE$^+$ is faster than LWE for the 1 KiB and 1 MiB databases, but slower (1.4×) for the 1 GiB database. SimplePIR is 30-100× faster than LWE$^+$. These results are the outcome of experiment (E5), whose results are presented in Fig. 6 in the body of the paper.

Application evaluation (Section 7.4).

(C6): For classic key look-ups we measure the wall-clock time needed to retrieve a PGP public-key with authenticated PIR, classic PIR without authentication, and by direct download. We measure 1.11 seconds for authenticated PIR, 1.10 seconds for unauthenticated PIR and 0.22 seconds for non-private direct look-up. This is the result of experiment (E6), whose results are discussed in Section 7.4 in the body of the paper.

(C7): To analyze the performance of Keyd in computing private statistics over keys, we measure user-perceived time and bandwidth of different predicate queries. For all the predicates, the user-perceived time and bandiwdth overheads of authenticated PIR are upper bounded by a factor of 1.05×. This is the outcome of experiment (E7), whose results are presented in Table 7 in the body of the paper.

I.4.2 Experiments

The experiments use at most six server machines (to run the client and servers) and an additional machine (that we call *local*) to manage the experiments. The local machine can be a commodity computer, since it is used only to run light scripts and gather results. Clone the repository on all the server machines and on the local machine.

(E1): [*15 human-minutes + 2 compute-hour*]: This experiment measures the user-time and bandwidth overheads of *two*-server authenticated-PIR schemes for point queries in comparison with unauthenticated PIR. This experiments uses three server machines: one client and two servers.

Preparation: Edit `simulations/multi/config.toml` on the *local* machine to indicate the IP address of the client machine and the IP addresses and ports of the two server machines. The default port numbers are safe to use.

Execution: Run the following commands from the repository's root on the *local* machine:

```
cd simulations/multi
APIR_USER=<username>
APIR_PASSWORD=<password>
APIR_PATH=<path>
python simul.py -e point
```

where `<username>` and `<password>` are the username and password for the servers, respectively, and `<path>` is the path of the repository's root on the servers.

Results: Run the following commands from the repository's root:

```
cd simulations/multi
python plot.py -e point
```

The command stores the figure in `simulations/multi/figures/point.eps`.

(E2): *[15 human-minutes + 18 compute-minutes]*: This experiment measures the impact of the number of servers on our multi-server authenticated-PIR schemes for point queries. This experiments uses six server machines: one client and five servers.
Preparation: Edit `simulations/multi/config.toml` on the *local* machine to indicate the IP address of the client machine and the IP addresses and ports of the five server machines. The default port numbers are safe.
Execution: Run the following commands from the repository's root on the *local* machine:

```
cd simulations/multi
APIR_USER=<username>
APIR_PASSWORD=<password>
APIR_PATH=<path>
python simul.py -e point_multi
```

where `<username>`, `<password>` and `<path>` are as in experiment E1.
Results: Run the following commands from the repository's root:

```
cd simulations/multi
python plot.py -e point_multi
```

The command stores the figure in `simulations/multi/figures/multi.eps`.

(E3): *[5 human-minutes + 9 compute-minutes]*: This experiment measures the cost of preprocessing for our multi-server authenticated-PIR scheme for point queries. This experiment uses one server machine.
Preparation: Nothing.
Execution: Run the following commands from the repository's root on the *server* machine:

```
cd simulations
make preprocessing
```

Results: Run the following commands from the repository's root:

```
cd simulations
python plot.py -e preprocessing
```

The command stores the figure in `simulations/figures/preprocessing.eps`.

(E4): *[15 human-minutes + 36 compute-minutes]*: This experiment measures the user-time and bandwidth overheads of two-server authenticated-PIR schemes for complex queries in comparison with unauthenticated PIR. This experiments uses three server machines: one client and two servers.
Preparation: As in experiment E1. The file `simulations/multi/config.toml` on the *local* machine must list *only two* servers.
Execution: Run the following commands from the repository's root on the *local* machine:

```
cd simulations/multi
APIR_USER=<username>
APIR_PASSWORD=<password>
APIR_PATH=<path>
python simul.py -e predicate
```

where `<username>`, `<password>` and `<path>` are as in experiment E1.
Results: Run the following commands from the repository's root:

```
cd simulations/multi
python plot.py -e predicate
```

The command stores the figure in `simulations/multi/figures/complex_lines.eps`.

(E5): *[15 human-minutes + 21 compute-hour]*: This experiment measures the user-time and bandwidth overheads of single-server authenticated-PIR schemes for point queries in comparison with SimplePIR [52]. This experiment uses one server machine.
Preparation: Nothing.
Execution: Run the following commands from the repository's root on the *server* machine:

```
cd simulations
make single
```

To evaluate SimplePIR, clone the following repository: `https://github.com/si-co/simplepir`. The code is the same as the original repository, but it runs the evaluation on the same database sizes as authenticated PIR and produces a compatible JSON file for the results. Run the following command (45 compute-minutes) from the repository's root on the *server* machine:

```
cd pir
go test -timeout 0 -run=PirSingle
```

Copy the file `simplePIR.json` (in the `pir` directory) in `simulation/results`.
Results: Run the following commands from the repository's root:

```
cd simulations
python plot.py -e single
```

The command stores the figure in `simulations/figures/single_bar.eps`.

(E6): *[20 human-minutes + 10 compute-minutes]*: This experiment measures the user-time needed download a PGP public-key with authenticated PIR for point queries, classic unauthenticated PIR for point queries and by direct download. This experiment uses three machines: one client and two servers.

Preparation: Download the dump of the SKS PGP key directory using the command

```
bash scripts/download_sks_parsed.sh
```

in the repository's root directory on the *two* servers. Set the IP addresses of the two servers in `simulations/real/real_client_pir.sh` and in `config.toml` (in the repository's root) on the client machine.
Execution: Run the following commands from the repository's root on the *first server*:

```
cd simulations/real
bash real_server_pir.sh 0
```

Similarly, on the *second server* run:

```
cd simulations/real
bash real_server_pir.sh 1
```

A server is running properly when it logs:

```
gRPC server started at <ip>:<port>
```

Once both servers started, run the following command on the *client machine*:

```
cd simulations/real
bash real_client_pir.sh
```

This command executes 30 look-ups with unauthenticated PIR and 30 with authenticated PIR. At the end, the client automatically shuts both servers down.
Results: Copy `simulations/results/stats_*` from the three machines (two servers and the client) on the *local* machine in the folder `/simulations/results`. Run the following commands:

```
cd simulations
python plot.py -e real
```

The command prints the results directly on the terminal.

(E7): *[20 human-minutes + 5 compute-hours]*: This experiment measures the user-time needed to compute statistics on the PGP public-keys with authenticated PIR for predicate queries and unauthenticated PIR. This experiment uses three machines: one client and two servers.
Preparation: As in Experiment E6, but set the IP addresses of the two servers in `simulations/real/real_client_fss.sh`.
Execution: Run the following commands from the repository's root on the *first server*:

```
cd simulations/real
bash real_server_fss.sh 0
```

Similarly, on the *second server* run:

```
cd simulations/real
bash real_server_fss.sh 1
```

For this experiment, it is not needed to wait for the servers to properly start. Run the following command on the *client machine*:

```
cd simulations/real
bash real_client_fss.sh
```

Results: Copy `simulations/results/stats_*` from the three machines (two servers and the client) on the *local* machine in the folder `/simulations/results`. Run the following commands:

```
cd simulations
python plot.py -e realcomplex
```

The command prints the results directly on the terminal. Table 7 has a different formatting, but values are the same as the one that the command prints on screen.

I.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at `https://secartifacts.github.io/usenixsec2023/`.

J Changelog

J.1 February 23, 2024

This revision includes the following changes.

- We update Appendix C by adding an overview of the proof strategy (Appendix C.1) and by fixing the proofs of Lemma 26 and Theorem 28 after Brett Falk, Pratyush Mishra, and Matan Shtepel pointed out a flaw in the original proof of Theorem 28 [84].
- We add the USENIX Security '23 Artifact Appendix in Appendix I and the corresponding badges at the beginning of the paper.
- We fix some typos in the document.

J.2 October 19, 2024

This revision included the following changes.

- We fix an error in Fig. 3, which shows the cost of retrieving a 1 KiB record using classic and authenticated PIR for point queries from two servers. The previous plot was the results of an experiment that used plain secret-sharing-based PIR as underlying classic PIR sheme, whereas the updated figure reflects results using a state-of-the-art PIR scheme based on distributed point point functions (DPF) [17, 18, 48]. Additionally, we change the structure of the database: the previous plot used a database structured as a matrix, where multiple blocks of data are stored in a single row. The current

approach represents the database as a vector, which better fits DPF-based PIR. We update Section 1 and Section 7.1 accordingly, and mention this error in Appendix I. The commit of reference for the new version of Fig. 3 is `0089520d113178aceca4534ebfd0e59d941d8bc7` and the new results file are included in `simulations/multi/final_results_fix`.

- We fix the caption of Fig. 4, which now mention the underlying classic PIR scheme that the experiment uses, i.e., classic secret-sharing over the binary field. We use this approach instead of DPF-based PIR as DPF constructions have tractable shares only with two servers. We update the second paragraph of Section 7.1 accordingly.
- We fix some additional typos in the document.