Verifiable Multi-server Private Information Retrieval

Liang Feng Zhang and Reihaneh Safavi-Naini

Institute for Security, Privacy and Information Assurance
Department of Computer Science
University of Calgary

Abstract. Private information retrieval (PIR) allows a client to retrieve any block x_i from a database $x = x_1 \cdots x_n$ (stored on a server) such that i remains hidden from the server. PIR schemes with unconditional privacy and sublinear (in n) communication complexity can be constructed assuming multiple honest-but-curious servers. This assumption however cannot be guaranteed in many real life scenarios such as using cloud servers. There are also extra properties such as efficient update of the database. In this paper, we consider a verifiable multi-server PIR (VPIR) model where the servers may be malicious and provide fraudulent answers. We construct an unconditionally t-private and computationally secure k-server VPIR scheme with communication complexity comparable to the best t-private k-server PIR scheme in the honest-but-curious server model. Our scheme supports efficient update of the database, identification of the cheating servers, tolerance of slightly corrupted answers, and multiple database outsourcing.

1 Introduction

Private information retrieval (PIR) allows a client to retrieve any block x_i from a database $x = x_1 \cdots x_n$ (stored on a server) such that i remains hidden from the server. The main efficiency measure of a PIR scheme is its communication complexity and defined to be the total number of bits communicated for retrieving a single bit of x. In a trivial PIR scheme, the client simply downloads x. Although perfectly private, this solution has a prohibitive communication complexity $\geq n$. Chor et al. [5] showed that the trivial solution is optimal in terms of communication complexity if there is only one server and perfect privacy is required. Non-trivial PIR schemes with communication complexity $< n$ have been constructed in information-theoretic (multi-server) setting [5,2,17] and computational (single-server) setting [13]. The former setting still provides privacy even if the server spends unrestricted computational resources to recover i once x_i has been collected. Let $1 \leq t < k$. A k-server PIR scheme is said to be t-private if no collusion of up to t servers can learn any information about i. The most efficient t-private PIR scheme [17] with $t > 1$ has communication complexity $O(n^{1/\lfloor(2k-1)/t\rfloor})$ in the honest-but-curious server model.

The PIR servers' computation complexity can be measured by the total number of database blocks read by the servers and is lower bounded by n in any

I. Boureanu, P. Owesarski, and S. Vaudenay (Eds.): ACNS 2014, LNCS 8479, pp. 62–79, 2014.

PIR schemes [3,1]. Advances in cloud computing makes it possible [14,6,12] to implement multi-server PIR using cloud servers such that the high computation complexity can be offloaded to the powerful clouds. However, the outsourcing requires a stronger adversary model as the clouds may provide incorrect answers due to malicious behaviors or accidental failures. In this paper, we strengthen the honest-but-curious server model of the existing multi-server PIR schemes [5,2,17] to provide security against malicious servers. We require that the client should be able to identify the malicious servers. This is very important as an unidentified malicious server can result in system failure without concern about its reputation. In practice, very few databases stay unchanged over time. Thus, we also would like our PIR scheme to have extra properties such as efficiently updating the outsourced database and catering for multiple databases. In a trivial malicious server PIR scheme the database owner may sign each block x_i of the database using any signature scheme and then send the "extended database" of (block, signature) to the clouds; the client can use any PIR scheme to retrieve x_i along with its signature from the "extended database" and then verify. However, this solution becomes insecure after the first database update as the server can always use old (block, signature) pairs without being detected. To improve this trivial solution, the database owner may consider $x_1, \ldots, x_n$ as leaves of a Merkle tree and publish the root of this tree for verification. To access one block, in this case one leaf of the tree, the client runs a multi-server PIR scheme once for each layer of the tree, obtains the required block and the siblings of all nodes on the path from the leaf to the root and verifies the result against the root of the tree. This solution provides basic update but with higher cost; and more importantly, it does not allow identification of cheating clouds; it treats each database individually with no saving when multiple databases are outsourced.

1.1 Our Contributions

In this paper, we consider verifiable multi-server PIR schemes that support efficient update, cheater identification and multiple database delegations.

VPIR Model. We introduce a verifiable multi-sever PIR (VPIR) model (see Figure 1) that consists of a database owner $\mathcal{D}$, a client $\mathcal{C}$ and k clouds $\mathcal{S}_1, \ldots, \mathcal{S}_k$. Let λ be a security parameter. $\mathcal{D}$ has a database $x = x_1 \cdots x_n \in \mathbb{F}_p^n$, where n is a polynomial of λ and p is a λ-bit prime. The client $\mathcal{C}$ has an index $i \in [n]$ and wants to learn x_i from the clouds, without revealing i. In a VPIR scheme $\Gamma = (\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Challenge}, \mathsf{Respond}, \mathsf{Verify}, \mathsf{Update})$, $\mathcal{D}$ is responsible to set up the system and update x. To set up the system, $\mathcal{D}$ runs a key generation algorithm $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, n)$ and a setup algorithm $vk_x \leftarrow \mathsf{Setup}(pk, sk, x)$. It publishes (pk, vk_x) and gives x to every cloud. To update the database from x to x', $\mathcal{D}$ runs an update algorithm $vk_{x'} \leftarrow \mathsf{Update}(pk, sk, vk_x, x')$, publishes $vk_{x'}$ and then instructs each cloud to change x to x'. To retrieve x_i, the client runs a query algorithm $(Q_1, \ldots, Q_k, \mathsf{aux}) \leftarrow \mathsf{Query}(pk, i)$ and sends a query Q_j to $\mathcal{S}_j$ for every $j \in [k]$. The cloud $\mathcal{S}_j$ runs an answer algorithm $A_j \leftarrow \mathsf{Answer}(pk, x, Q_j)$ and replies with A_j. To verify A_j, the client runs a challenge algorithm $(I_1, \ldots, I_k) \leftarrow \mathsf{Challenge}(pk)$ to produce

a challenge I_j for every cloud $\mathcal{S}_j$ and the cloud $\mathcal{S}_j$ must respond with a proof $\sigma_j \leftarrow \mathsf{Respond}(pk, x, Q_j, I_j)$ vouching for the correctness of A_j. At last, if the k answers $A_1, \ldots, A_k$ are all correct, then the client can run an extract algorithm $\mathsf{Extract}(pk, vk_x, \{(A_j, I_j, \sigma_j) : j \in [k]\}, \mathsf{aux})$ to compute x_i. The running time of all algorithms must be polynomial in λ.

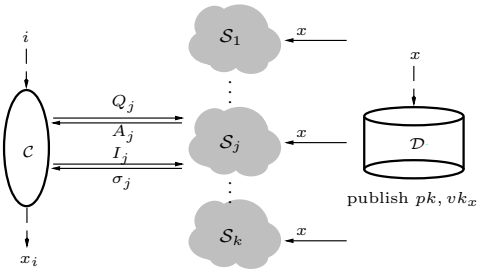

Fig. 1. Verifiable multi-server PIR

We define privacy (information theoretic) and verifiability (computational) of the system in line with the privacy in PIR and the security in VC. The definition is justified because the privacy attacker has unlimited time to break the system, while verifiability of the clouds' answers should be provided without delay. The clouds' answers are verified individually and so the cheating clouds can be identified. The public key pk can be used for outsourcing multiple databases. In fact it can be computed and published once by a third party and used by many database owners. The algorithm Update only requires the difference between x' and x and the algorithm $\mathsf{Extract}$ only requires a small portion of pk. Throughout the paper, the *size* $|w|$ of any string w is defined to be the number of λ-bit blocks it contains.

VPIR Constructions. We propose a basic construction Γ_0 and a main construction Γ_1 of VPIR schemes. The basic construction is obtained by adding verifiability [15] to the best t-private PIR scheme [17]. As a main drawback, the communication complexity of Γ_0 squares that of the underlying PIR scheme [17]. The main construction reduces this cost to be comparable to [17]. Let $t < k$ and $d = \lfloor (2k-1)/t \rfloor$. Let $m > 0$ be such that $\binom{m}{d} \geq n$. The main construction Γ_1 has the following properties.

Privacy. The scheme Γ_1 achieves unconditional t-privacy in the sense that any collusion of up to t clouds learns no information about the client's index i, even if they have unlimited computing power.

Security. No cloud $\mathcal{S}_j$ can deceive the client into accepting an incorrect answer $\bar{A}_j$ with a forged proof $\bar{\sigma}_j$, except with negligible probability. In particular, the client can identify any cheating cloud by verifying that cloud's answer. The security is based on the $(m+d-1)$-SBDH assumption (see Definition 2).

Communication. Each block of our database x has λ bits and so the communication complexity of our VPIR schemes is the number of λ-bit blocks communicated

by the client. To retrieve a block x_i, the client sends a query Q_j of size $O(m)$ to each cloud $\mathcal{S}_j$ and receives an answer A_j of size $O(m)$. It also sends $\mathcal{S}_j$ a challenge I_j of size $O(\log m)$ and receives a proof σ_j of size $O(\lambda m)$. Thus, the communication complexity of Γ_1 is $O(\lambda m) = O(\lambda n^{1/\lfloor (2k-1)/t \rfloor})$.

Computation. (1) Client: The computation of $Q_1, \ldots, Q_k$ consists of evaluating m univariate polynomials of degree $\leq t$ over $\mathbb{F}_p$. The verification of each answer A_j consists of checking λ equations, where each equation requires $\leq 2m$ bilinear paring computations. The extraction of x_i from $A_1, \ldots, A_k$ consists of interpolating and evaluating a univariate polynomial of degree $\leq 2k-1$. The client's computation is dominated by $O(m)$ pairing computations. (2) Cloud: The computation of A_j consists of evaluating a polynomial $P_x(\boldsymbol{z}, y)$ (see equation (2)) at $O(m)$ points. The computation of σ_j consists of decomposing P_x (see Lemma 1) and computing $O(m)$ bilinear group elements and $O(m)$ field elements. We show that the response time of each cloud in Γ_1 can be significantly reduced by distributing the PIR server computation to its numerous computing units.

Storage. Cloud: In Γ_1, each cloud stores a copy of x. Client: The client uses (vk_x, aux) and $O(m)$ elements of pk for verification and reconstruction. For each retrieval, the client also temporarily stores k triples $\{(A_j, I_j, \sigma_j) : j \in [k]\}$ of total size $O(\lambda m)$.

Update. To change one block of x, say x_i to x_i', $\mathcal{D}$ needs to compute $vk_{x'}$ for $x' = x_1 \cdots x_{i-1} x_i' x_{i+1} \cdots x_n$ using $d+1$ multiplications in a bilinear group of order p, and then instructs each cloud to change x_i to x_i'. Note that d is a constant. Hence, the update complexity is $O(1)$. The update is verifiable in the sense that any cloud that does not change x accordingly will be detected as cheating in the future executions of Γ_1.

Additional Properties. The construction Γ_1 also provides two additional properties. Error Tolerance: Each cloud's answer in Γ_1 is a codeword under a Reed-Solomon code. We show how to modify Γ_1 such that slight corruptions of each cloud's answer can be tolerated. Multiple Database Delegation: The system public key pk has size $(2d+2+o(1))n$ and can be used by any database owner to delegate their database. For each database x, a short (one group element) public verification key vk_x will be published. Updating the database x is done by updating this short key.

1.2 Related Work

PIR with Malicious Servers. Constructing PIR schemes that are secure in a malicious server model is well-motivated and has been put forth by Beimel [1]. The GMW compiler [10] allows one to compile any PIR scheme in the honest-but-curious server model into a PIR scheme in the malicious server model. However, its communication complexity is higher than the trivial PIR scheme and thus much higher than Γ_1. The robust multi-server PIR schemes that tolerate a limited number of malicious servers have been studied in [4,9,7]. The communication complexities of all these schemes are much higher than Γ_1. Furthermore, if all the answering servers in these schemes collude with each other, then the

client may be deceived into computing an incorrect value of x_i. In contrast, the client in Γ_1 will reject and not be deceived. More importantly, the PIR schemes of [4,9,7] do not support update, while the database owner in Γ_1 can efficiently and verifiably update its outsourced database x.

Outsourced PIR. The practicality of outsourcing PIR has been demonstrated by [14,6,12]. Mayberry et al. [14] presented a MapReduce-based outsourced single-server PIR scheme called "PIRMAP" which is more than one order of magnitude faster than the trivial PIR scheme. Devet [6] developed parallelization techniques that significantly reduce each cloud's response time by distributing the delegated PIR server computation among its numerous computing units. Huang et al. [12] combined certain multi-server PIR with oblivious RAM to obtain outsourced PIR schemes where the access patterns of the database owner and all clients are hidden from the clouds. None of the schemes [14,6,12] consider malicious clouds.

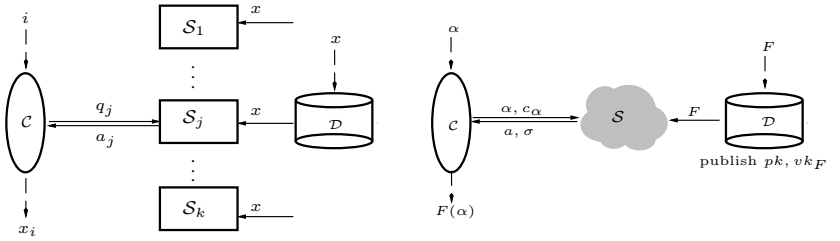

Fig. 2. Private information retrieval and public verifiable computation

1.3 Building Blocks and Our Techniques

Woodruff-Yekhanin PIR Scheme. Let $t, k > 0$ be integers such that $1 \leq t < k$. A t-private k-server PIR scheme (see Figure 2) involves a database owner $\mathcal{D}$, a client $\mathcal{C}$ and k servers $\mathcal{S}_1, \ldots, \mathcal{S}_k$, where $\mathcal{D}$ has a database $x = x_1 \cdots x_n \in \mathbb{F}_p^n$ and $\mathcal{C}$ wants to learn x_i. The $\mathcal{D}$ does not directly communicate with the client but gives x to each server. To privately retrieve x_i, the client computes a query q_j to each server $\mathcal{S}_j$ and receives an answer a_j in return. The queries are computed such that any t of them give no information about i; the k answers allow the client to recover x_i. Let $d = \lfloor (2k-1)/t \rfloor$ and $m = O(n^{1/d})$ be such that $\binom{m}{d} \geq n$. Let $\mathsf{IndEnc} : [n] \to \mathbb{F}_p^m$ be a 1-to-1 index encoding that maps any index $i \in [n]$ to a 0-1 vector of Hamming weight d. Let

$$F_x(\boldsymbol{z}) = \mathsf{PolyEnc}_0(x) \triangleq \sum_{j=1}^{n} x_j \prod_{\ell:\mathsf{IndEnc}(j)_\ell=1} z_\ell \tag{1}$$

be a polynomial encoding of x of total degree $\leq d$ in $\boldsymbol{z} = (z_1, \ldots, z_m)$ such that $F_x(\mathsf{IndEnc}(i)) = x_i$ for every $i \in [n]$. Woodruff and Yekhanin [17] constructed a PIR scheme Π_{wy} where the client simply computes $F_x(\boldsymbol{z})$ at its private input $\mathsf{IndEnc}(i)$ with the servers. The client computes k queries $q_1, \ldots, q_k$ as k shares of

$\mathsf{IndEnc}(i)$ under Shamir's t-private threshold secret sharing scheme; each server $\mathcal{S}_j$ answers with $a_j = (F_x(q_j), \frac{\partial F_x}{\partial z_1}|_{q_j}, \ldots, \frac{\partial F_x}{\partial z_m}|_{q_j})$; from $a_1, \ldots, a_k$ the client can interpolate a univariate polynomial of degree $< 2k$ whose evaluation at 0 gives x_i. In Π_{wy} the client sends m field elements (i.e., $q_j \in \mathbb{F}_p^m$) to each server and receives $m+1$ field elements (i.e., a_j). Its communication complexity is $k(2m+1)$.

Papamanthou et al. PVC Schemes. Papamanthou et al. [15] introduced a publicly verifiable computation (PVC) model (see Figure 2) that involves a function owner $\mathcal{D}$, a cloud $\mathcal{S}$ and a client $\mathcal{C}$, where $\mathcal{D}$ has a function $F \in \mathcal{F}$ and $\mathcal{C}$ wants to learn $F(\alpha)$. The function owner computes (pk, vk_F), makes them public and then gives F to the cloud; the client gives α and a challenge c_α to the cloud; the cloud computes $a = F(\alpha)$ and a proof σ using (pk, F); at last, the client can verify a using (pk, vk_F, σ). Let $\mathcal{F} \subseteq \mathbb{F}_p[\boldsymbol{z}]$ be the set of polynomials of total degree $\leq d$ in $\boldsymbol{z} = (z_1, \ldots, z_m)$. Papamanthou et al. [15] constructed a scheme Π_0 (Section B.1, eprint version of [15]) which allows the client to verify the result $F(\alpha)$ from $\mathcal{S}$ for any $(F, \alpha) \in \mathcal{F} \times \mathbb{F}_p^m$ and a scheme Π_1 (Corollary 1, eprint version of [15]) that allows the client to verify the result $\frac{\partial F}{\partial z_\ell}\big|_\alpha$ from $\mathcal{S}$ for any $((F, \ell), \alpha) \in (\mathcal{F} \times [m]) \times \mathbb{F}_p^m$. In both schemes, the client sends a challenge c_α of size $m-1$ to the cloud and receives a proof of size $O(m)$.

Our Constructions. Our basic VPIR construction Γ_0 is a composition of Π_{wy}, Π_0 and Π_1. In Γ_0, each cloud $\mathcal{S}_j$ performs the computation of the jth server in Π_{wy}. The computation of a_j by $\mathcal{S}_j$ involves one evaluation and m differentiations of the polynomial $F_x(\boldsymbol{z})$ at q_j. We simply enforce the integrity of these computations using Π_0 and Π_1, respectively.

In Γ_0, the client sends a challenge of size $m-1$ to $\mathcal{S}_j$ and receives a proof of size $O(m)$ from $\mathcal{S}_j$ for every component of a_j. Thus, the communication complexity of Γ_0 is $k(m+1) \cdot O(m) = O(m^2)$ and squares that of Π_{wy}. Our main construction Γ_1 reduces it to $O(\lambda m)$ by limiting the proof size of each cloud. To do so, a natural idea is using *probabilistic verification:* for every $j \in [k]$, the client verifies only λ random components of a_j. If the cloud $\mathcal{S}_j$ has tampered with a constant fraction (say δ) of a_j, then $\mathcal{S}_j$ will be detected with overwhelming probability $1 - (1-\delta)^\lambda$. However, a clever $\mathcal{S}_j$ may tamper with only one component of a_j and deceive the client with non-negligible probability $(1 - \frac{1}{m+1})^\lambda$. To thwart this attack, we encode a_j using an error-correcting code $C : \mathbb{F}_p^{m+1} \to \mathbb{F}_p^M$ such that $\mathcal{S}_j$ must change a constant fraction of $A_j = C(a_j)$ in order to change a_j, where $M = O(m+1)$. Otherwise, the client can decode a_j from A_j. We take C to be the Reed-Solomon code that encodes any message $w = w_0 \cdots w_m \in \mathbb{F}_p^{m+1}$ as $C(w) = (f_w(\gamma_1), \ldots, f_w(\gamma_M))$, where $f_w(y) = w_0 + w_1 y + \cdots + w_m y^m$ and $\gamma_1, \ldots, \gamma_M \in \mathbb{F}_p$ are distinct. Let

$$P_x(\boldsymbol{z}, y) = \mathsf{PolyEnc}_1(x) \triangleq F_x(\boldsymbol{z}) + \sum_{\ell=1}^{m} \frac{\partial F_x(\boldsymbol{z})}{\partial z_\ell} y^\ell, \tag{2}$$

where $F_x(\boldsymbol{z}) = \mathsf{PolyEnc}_0(x)$. Then $A_j = C(a_j) = (P_x(q_j, \gamma_1), \ldots, P_x(q_j, \gamma_M))$. The client in Γ_1 learns A_j from $\mathcal{S}_j$ and then randomly verifies λ components of A_j. If $\mathcal{S}_j$ tampers with a constant fraction of A_j, then it will be detected; otherwise, the client can recover a_j. The client can use Π_0 to verifiably compute

the $(m+1)$-variate polynomial P_x of total degree $\leq m+d-1$. But that requires a public key of size $\binom{2m+d}{m+d-1} = \exp(O(n))$. We develop a new PVC scheme Π_2 for P_x which requires a public key of size $(2d+2+o(1))n$. In Π_2, the client sends a challenge of size m to the cloud and receives a proof of size $O(m)$. Thus, the λ proofs required by our client have total size $O(\lambda m)$. Our client must send the M challenges for computing A_j, and then ask for the λ proofs only after A_j has been received; otherwise the cloud will know which λ components of A_j will be verified and then break the security by changing other components. However, we observe that these challenges can be chosen such that they are equal to each other without compromising the security. That is, the client can send one *common challenge* of size $O(m)$ for the M computations. Enforcing the integrity of computing P_x using Π_2 with probabilistic verification and common challenge gives us a VPIR scheme of communication complexity $O(\lambda m)$.

2 Preliminaries

2.1 Our Model

We denote any negligible function in λ by $\mathsf{neg}(\lambda)$ and any polynomial function in λ by $\mathsf{poly}(\lambda)$. Our VPIR model (see Figure 1) involves a database owner $\mathcal{D}$, a client $\mathcal{C}$ and k clouds $\mathcal{S}_1, \ldots, \mathcal{S}_k$, where $\mathcal{D}$ has a database $x = x_1 \cdots x_n \in \mathbb{F}_p^n$ and the client $\mathcal{C}$ wants to privately retrieve x_i.

Definition 1. *A* k-server VPIR scheme *is a tuple* $\Gamma = (\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Challenge}, \mathsf{Respond}, \mathsf{Extract}, \mathsf{Update})$ *of eight algorithms, where*

- $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, n)$ *is a key generation algorithm which takes as input* (λ, n) *and outputs a public key* pk *and a secret key* sk*;*
- $vk_x \leftarrow \mathsf{Setup}(pk, sk, x)$ *is a setup algorithm which takes as input* (pk, sk) *and any database* $x \in \mathbb{F}_p^n$ *and outputs a public verification key* vk_x*;*
- $(Q_1, \ldots, Q_k, \mathsf{aux}) \leftarrow \mathsf{Query}(pk, i)$ *is a query algorithm which takes as input* pk *and any index* $i \in [n]$ *and outputs* k *queries* $Q_1, \ldots, Q_k$ *along with some auxiliary information* aux*;*
- $A_j \leftarrow \mathsf{Answer}(pk, x, Q_j)$ *is an answer algorithm which computes an answer* A_j *from* (pk, x, Q_j)*;*
- $(I_1, \ldots, I_k) \leftarrow \mathsf{Challenge}(pk)$ *is a challenge algorithm that generates* k *challenges* $I_1, \ldots, I_k$ *for the clouds;*
- $\sigma_j \leftarrow \mathsf{Respond}(pk, x, Q_j, I_j)$ *is a respond algorithm which takes as input* (pk, x, Q_j, I_j) *and outputs a proof* σ_j *vouching for the correctness of* A_j*;*
- $\{x_i, \perp\} \leftarrow \mathsf{Extract}(pk, vk_x, \{(A_j, I_j, \sigma_j) : j \in [k]\}, \mathsf{aux})$ *is an extract algorithm that reconstructs* x_i *or outputs* $\perp$ *(to indicate failure);*
- $vk_{x'} \leftarrow \mathsf{Update}(pk, sk, vk_x, x')$ *is an update algorithm which generates a new public verification key* $vk_{x'}$ *from* (pk, sk, vk_x) *and the new database* x'*.*

The database owner $\mathcal{D}$ is responsible to set up the system and update x. To set up the system, $\mathcal{D}$ runs KeyGen and Setup to compute (pk, sk) and vk_x, publishes (pk, vk_x) and gives x to every cloud. To update the database from x to x', $\mathcal{D}$

runs Update to compute $vk_{x'}$, publishes $vk_{x'}$ and then instructs each cloud to change x to x'. To retrieve x_i, $\mathcal{C}$ runs Query to compute $(Q_1, \ldots, Q_k, \mathsf{aux})$ and sends a query Q_j to the cloud $\mathcal{S}_j$ for every $j \in [k]$. The cloud $\mathcal{S}_j$ runs $\mathsf{Compute}$, computes and replies with A_j. To verify these answers, the client runs $\mathsf{Challenge}$ to generate k challenges $(I_1, \ldots, I_k)$. Each cloud $\mathcal{S}_j$ generates a proof σ_j using $\mathsf{Respond}$. At last, the client can run $\mathsf{Extract}$ to verify the k answers $A_1, \ldots, A_k$ and then compute x_i if all answers are correct.

Correctness. The scheme Γ is said to be correct if the extract algorithm always outputs the correct value of x_i when all k clouds are honest. Formally, let $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, n)$. Let $x^{(0)} \in \mathbb{F}_p^n$ and $vk_{x^{(0)}} = \mathsf{Setup}(pk, sk, x^{(0)})$. For $u = 1, \ldots, U(= \mathsf{poly}(\lambda))$, let $vk_{x^{(u)}} \leftarrow \mathsf{Update}(pk, sk, vk_{x^{(u-1)}}, x^{(u)})$. Π is *correct* if for any $u \in \{0, 1, \ldots, U\}$, any $i \in [n]$, any $(Q_1, Q_2, \ldots, Q_k, \mathsf{aux}) \leftarrow \mathsf{Query}(pk, i)$ and any $(I_1, \ldots, I_k) \leftarrow \mathsf{Challenge}(pk)$, it holds that $\mathsf{Extract}(pk, vk_{x^{(u)}}, \{(A_j, I_j, \sigma_j) : j \in [k]\}, \mathsf{aux}) = x_i^{(u)}$, where $A_j = \mathsf{Answer}(pk, x^{(u)}, Q_j)$ and $\sigma_j = \mathsf{Respond}(pk, x^{(u)}, Q_j, I_j)$ for every $j \in [k]$.

t-Privacy. The scheme Γ is said to be (unconditionally) t-private if no collusion of up to t servers can learn any information about i. Formally, Γ is (unconditionally) t-private if for any k, n, any $i_1, i_2 \in [n]$ and any set $T \subseteq [k]$ of size $|T| \leq t$, the distributions of $\mathsf{Query}_T(pk, i_1)$ and $\mathsf{Query}_T(pk, i_2)$ are identical, where Query_T denotes concatenation of j-th outputs of Query for $j \in T$.

Security. The scheme Γ is said to be secure if no probabilistic polynomial time (PPT) adversary can deceive the client into reconstructing an incorrect value of x_i. Formally, Γ is secure if no PPT adversary $\mathcal{A}$ can win with non-negligible probability in the following security game:

1. The challenger picks $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, n)$ and then gives pk to $\mathcal{A}$.
2. $\mathcal{A}$ picks a database $x^{(0)} \in \mathbb{F}_p^n$ and makes a query $\mathsf{Setup}(pk, sk, x^{(0)})$ to the challenger. The challenger returns $vk_{x^{(0)}}$. For $u = 1, \ldots, U(= \mathsf{poly}(\lambda))$, $\mathcal{A}$ makes a query $\mathsf{Update}(pk, sk, vk_{x^{(u-1)}}, x^{(u)})$. The challenger returns $vk_{x^{(u)}}$ every time.
3. $\mathcal{A}$ picks $u \in \{0, 1, \ldots, U\}$, $i \in [n]$ and then gives (u, i) to the challenger. The challenger gives k queries $(Q_1, \ldots, Q_k)$ to $\mathcal{A}$ and stores a string aux. $\mathcal{A}$ returns k answers $(\bar{A}_1, \bar{A}_2, \ldots, \bar{A}_k)$. The challenger then gives k challenges $(I_1, \ldots, I_k)$ to $\mathcal{A}$ and receives k proofs $(\bar{\sigma}_1, \bar{\sigma}_2, \ldots, \bar{\sigma}_k)$ in return.
4. $\mathcal{A}$ wins if $\mathsf{Extract}(pk, vk_{x^{(u)}}, \{(\bar{A}_j, I_j, \bar{\sigma}_j) : j \in [k]\}, \mathsf{aux}) \notin \{x_i^{(u)}, \perp\}$.

In our security game, $\mathcal{A}$ cannot deceive the client into computing an incorrect value of $x_i^{(u)}$ even if it can freely choose and update the database x. Thus, a VPIR scheme secure under our definition above not only allows the client to verifiably retrieve any block from the database, but also allows $\mathcal{D}$ to verifiably update its outsourced database without compromising the security. The query, answer and communication complexities of Γ are defined to be $\mathcal{QC}_\Gamma = \max\{|Q_j| + |I_j|\}$, $\mathcal{AC}_\Gamma = \max\{|A_j| + |\sigma_j|\}$ and $\mathcal{CC}_\Gamma = \sum_{j=1}^k (|Q_j| + |I_j| + |A_j| + |\sigma_j|)$, respectively.

2.2 Woodruff-Yekhanin PIR Scheme

Let $1 \leq t < k$ and $d = \lfloor (2k-1)/t \rfloor$. Let $m = O(n^{1/d})$ be such that $\binom{m}{d} \geq n$. Let $\beta_1, \ldots, \beta_k \in \mathbb{F}_p^*$ be distinct. Let $\mathsf{IndEnc} : [n] \to \mathbb{F}_p^m$ be the 1-to-1 index encoding that maps any index $i \in [n]$ as a 0-1 vector of Hamming weight d. Let $F_x(\boldsymbol{z})$ be the polynomial encoding of any database $x \in \mathbb{F}_p^n$ (see (1)) such that $F_x(\mathsf{IndEnc}(i)) = x_i$ for every $i \in [n]$. Woodruff and Yekhanin's t-private k-server PIR scheme is a triple $\Pi_{\text{wy}} = (\mathsf{Query}, \mathsf{Answer}, \mathsf{Extract})$ of algorithms, where

- $(q_1, \ldots, q_k, \mathsf{aux}) \leftarrow \mathsf{Query}(i)$ is a query algorithm that picks $v_1, \ldots, v_t \leftarrow \mathbb{F}_p^m$; computes a query $q_j = \mathsf{IndEnc}(i) + \beta_j v_1 + \cdots + \beta_j^t v_t$ for every $j \in [k]$ and an auxiliary information $\mathsf{aux} = \{v_1, \ldots, v_t\}$.
- $a_j \leftarrow \mathsf{Answer}(x, q_j)$ is an answer algorithm that computes $a_{j,0} = F_x(q_j)$ and $a_{j,\ell} = \frac{\partial F_x}{\partial z_\ell}\big|_{q_j}$ for every $\ell \in [m]$ and outputs $a_j = (a_{j,0}, a_{j,1}, \ldots, a_{j,m})$.
- $\mathsf{Extract}(a_1, \ldots, a_k, \mathsf{aux})$ is an extract algorithm that interpolates a polynomial $f(y) = F_x(\mathsf{IndEnc}(i) + yv_1 + \cdots + y^t v_t)$ and outputs $f(0) = x_i$.

The algorithm Query will be run by the client to generate k queries and aux; the algorithm Answer will be run by each server to compute an answer; the algorithm $\mathsf{Extract}$ will be run by the client to recover x_i. In Π_{wy}, the k queries $q_1, \ldots, q_k$ are k shares of $\mathsf{IndEnc}(i)$ under Shamir's t-private threshold secret sharing scheme. Thus, no t or less servers can learn any information about i and thus the t-privacy follows. The communication complexity of Π_{wy} is $k(2m+1) = O(m)$.

2.3 Papamanthou et al. PVC Schemes

Papamanthou et al.'s PVC scheme [15] (see Figure 2) is a tuple $\Pi = (\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Challenge}, \mathsf{Compute}, \mathsf{Verify}, \mathsf{Update})$ of six algorithms, where

- $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{F})$ is a key generation algorithm that takes as input λ and a function family $\mathcal{F}$ and outputs a public key pk and a secret key sk;
- $vk_F \leftarrow \mathsf{Setup}(pk, sk, F)$ is a setup algorithm that computes a public verification key vk_F for any $F \in \mathcal{F}$ with the knowledge of (pk, sk);
- $c_\alpha \leftarrow \mathsf{Challenge}(pk, \alpha)$ is a challenge algorithm that produces a challenge c_α for any α in the domain of F;
- $(a, \sigma) \leftarrow \mathsf{Compute}(pk, F, \alpha, c_\alpha)$ computes $a = F(\alpha)$ along with a proof σ;
- $\{F(\alpha), \perp\} \leftarrow \Pi.\mathsf{Verify}(pk, vk_F, \alpha, c_\alpha, a, \sigma)$ is a verification algorithm that checks if a is indeed equal to $F(\alpha)$;
- $vk_{F'} \leftarrow \mathsf{Update}(pk, sk, vk_F, F')$ is an update algorithm that computes a public verification key $vk_{F'}$ based on (pk, sk, vk_F) and the changes of the new function F' with respect to F.

In a PVC scheme, $\mathcal{D}$ is responsible to set up the system and update F. To set up the system, $\mathcal{D}$ runs the first two algorithms to compute (pk, sk) and vk_F. It publishes (pk, vk_F) and gives F to the cloud. To update the function from F to F', $\mathcal{D}$ runs Update to compute $vk_{F'}$, publishes $vk_{F'}$ and instructs the cloud $\mathcal{S}$ to change F to F'. To compute $F(\alpha)$, the client $\mathcal{C}$ runs $\mathsf{Challenge}$ and picks

a challenge c_α to the cloud. The cloud runs $\mathsf{Compute}$ and replies with (a, σ). At last, the client runs Verify to check if a is indeed equal to $F(\alpha)$. The query, answer and communication complexities of Π are defined to be $|\alpha|+|c_\alpha|$, $|a|+|\sigma|$ and $|\alpha| + |c_\alpha| + |a| + |\sigma|$, respectively. Let $\mathbb{F} = \{z_1^{e_1} \cdots z_m^{e_m} : e_1 + \cdots + e_m \leq d\}$ and $\mathcal{F} = \mathrm{span}(\mathbb{F}) \subseteq \mathbb{F}_p[\boldsymbol{z}]$. Papamanthou et al. [15] constructed a PVC scheme Π_0 and a PVC scheme Π_1 for computing the evaluations and differentiations of any $F \in \mathcal{F}$. In both schemes, the client sends a challenge of size $m - 1$ to the cloud and receives a proof of size $O(m)$ in return. We also observe that the Verify in both schemes uses at most $m + d - 2$ out of the $\binom{m+d}{d}$ components of pk.

2.4 Bilinear Maps and Assumptions

Let $\mathcal{G}$ be a generator which takes λ as input and outputs a bilinear map instance $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, where $\mathbb{G} = \langle g \rangle$ and $\mathbb{G}_T$ are cyclic groups of prime order p; and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a non-degenerate bilinear map such that $e(g^a, g^b) = e(g, g)^{ab}$ for any $a, b \in \mathbb{F}_p$ and $e(g, g)$ is a generator of $\mathbb{G}_T$.

Definition 2. (d-SBDH) *Let* $\Lambda = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}(1^\lambda)$. *Let* $d = \mathsf{poly}(\lambda)$. *We say that the* bilinear d-strong Diffie-Hellman assumption *holds if for any PPT algorithm* $\mathcal{A}$, $\Pr[s \leftarrow \mathbb{F}_p^* : \mathcal{A}(\Lambda, g, g^s, \ldots, g^{s^d}) = (\theta, e(g, g)^{1/(s+\theta)})] < \mathsf{neg}(\lambda)$, *where* $\theta \in \mathbb{F}_p^* \setminus \{-s\}$.

3 Our Constructions

In this section we first present our basic construction Γ_0 as a composition of Π_{wy}, Π_0 and Π_1. We then improve Γ_0 to our main construction Γ_1 whose communication complexity is comparable to the PIR of [17].

3.1 Basic Construction

In Π_{wy} the PIR servers' computations consist of evaluating and differentiating the polynomial $F_x(\boldsymbol{z}) = \mathsf{PolyEnc}_0(x)$ at k points $q_1, \ldots, q_k$ which are k shares of $\mathsf{IndEnc}(i)$ under Shamir's t-private threshold secret sharing scheme. In Γ_0 (see **Fig. 3**), the k clouds $\mathcal{S}_1, \ldots, \mathcal{S}_k$ perform these computations on behalf of the PIR servers. For every $j \in [k]$, the client runs Π_0 with $\mathcal{S}_j$ to compute $a_{j,0}$ and runs Π_1 with $\mathcal{S}_j$ to compute $a_{j,\ell}$ for every $\ell \in [m]$.

Correctness, Privacy and Security. The correctness of Γ_0 follows from that of Π_{wy}, Π_0 and Π_1. The only information about i that the clouds can learn is from the k points $q_1, \ldots, q_k$ which are k shares of $\mathsf{IndEnc}(i)$ under Shamir's t-private threshold secret sharing scheme. Thus, no collusion of up to t clouds can learn any information about $\mathsf{IndEnc}(i)$, i.e., Γ_0 achieves unconditional t-privacy. In Γ_0, the client runs $k(m+1)$ independent Π_0 and Π_1 instances with the clouds. If any cloud can break one of the instances, then we can simulate that cloud to break the d-SBDH assumption as in the security proofs of Π_0 and Π_1 (see [15] for the proofs). Hence, Γ_0 is secure under the d-SBDH assumption.

Theorem 1. *Γ_0 is an unconditionally t-private and computationally secure VPIR scheme. Its security is based on the d-SBDH assumption.*

$\mathsf{KeyGen}(1^\lambda, n)$: Output $(pk, sk) \leftarrow \Pi_0.\mathsf{KeyGen}(1^\lambda, \mathrm{span}(\mathbb{F}))$, where $sk = \tau \in \mathbb{F}_p^m$.

$\mathsf{Setup}(pk, sk, x)$: Output $vk_x \leftarrow \Pi_0.\mathsf{Setup}(pk, sk, F_x)$.

$\mathsf{Query}(pk, i)$: Compute $(q_1, \ldots, q_k, \mathsf{aux}') \leftarrow \Pi_{\mathrm{wy}}.\mathsf{Query}(i)$. For every $j \in [k]$ and $\ell \in \{0, 1, \ldots, m\}$, compute $c_{j,\ell} \leftarrow \Pi.\mathsf{Challenge}(pk, q_j)$, where $\Pi = \Pi_0$ if $\ell = 0$ and $\Pi = \Pi_1$ otherwise. Output $Q_j = (q_j, c_{j,0}, c_{j,1}, \ldots, c_{j,m})$ for all $j \in [k]$ and $\mathsf{aux} = (\mathsf{aux}', \{c_{j,\ell} : j \in [k], 0 \le \ell \le m\})$.

$\mathsf{Answer}(pk, x, Q_j)$: Compute $(a_{j,\ell}, \sigma_{j,\ell}) \leftarrow \Pi.\mathsf{Compute}(pk, F_x, q_j, c_{j,\ell})$, where $\Pi = \Pi_0$ if $\ell = 0$ and $\Pi = \Pi_1$ if $\ell \in [m]$. Output $A_j (= a_j) = (a_{j,0}, a_{j,1}, \ldots, a_{j,m})$.

$\mathsf{Challenge}(pk)$: For every $j \in [k]$, set $I_j = \{0, 1, \ldots, m\}$. Output $(I_1, \ldots, I_k)$.

$\mathsf{Respond}(pk, x, Q_j, I_j)$: Output $\sigma_j = \{\sigma_{j,\ell} : \ell \in I_j\}$.

$\mathsf{Extract}(pk, vk_x, \{(A_j, I_j, \sigma_j) : j \in [k]\}, \mathsf{aux})$: Compute $b_{j,\ell} = \Pi.\mathsf{Verify}(pk, vk_x, q_j, c_{j,\ell}, a_{j,\ell}, \sigma_{j,\ell})$, for every $j \in [k]$ and $\ell \in I_j$. where $\Pi = \Pi_0$ if $\ell = 0$ and $\Pi = \Pi_1$ otherwise. If all $b_{j,\ell}$'s are 1, output $x_i = \Pi_{\mathrm{wy}}.\mathsf{Extract}(A_1, \ldots, A_k, \mathsf{aux}')$. Otherwise, output $\perp$.

$\mathsf{Update}(pk, sk, vk_x, x')$: Output $vk_{x'} \leftarrow \Pi_0.\mathsf{Update}(pk, sk, vk_x, \mathsf{PolyEnc}_0(x'))$.

Fig. 3. The scheme Γ_0

Complexity. The database owner $\mathcal{D}$ runs $\Pi_0.\mathsf{KeyGen}$ and $\Pi_0.\mathsf{Setup}$ to set up the system. Both algorithms are executed once and take time $O(n)$. $\mathcal{D}$ may also run $\Pi_0.\mathsf{Update}$ to update x. To change one component of x, $\mathcal{D}$ needs to multiply vk_x with one element from $\{g^{h(\tau)} : h(\boldsymbol{z}) \in \mathbb{F}\}$ (see [15]). Therefore, the update complexity is $O(1)$. The client $\mathcal{C}$ computes and sends Q_j and I_j to the cloud $\mathcal{S}_j$ for every $j \in [k]$. It receives A_j and σ_j. Therefore, $\mathcal{QC}_{\Gamma_0} = \max\{|Q_j| + |I_j|\} = O(m^2)$, $\mathcal{AC}_{\Gamma_0} = \max\{|A_j| + |\sigma_j|\} = O(m^2)$ and $\mathcal{CC}_{\Gamma_0} = O(m^2)$. For verification, the client uses aux ($|\mathsf{aux}| = O(m^2)$) and $m + d - 2 = O(m)$ components of pk. Each cloud stores x, runs $\Pi_0.\mathsf{Compute}$ once and $\Pi_1.\mathsf{Compute}$ m times.

Variant. In Γ_0, the client communicates four messages Q_j, A_j, I_j and σ_j with every cloud $\mathcal{S}_j$ in two rounds. Note that I_j is always equal to $\{0, 1, \ldots, m\}$. We can merge the four messages and obtain a two-message version of Γ_0: the client simply sends Q_j to each cloud $\mathcal{S}_j$ and $\mathcal{S}_j$ replies with (A_j, σ_j). Thus, our basic construction can be made into one round.

3.2 Main Construction

In this section, we reach Γ_1 using a series of modifications to Γ_0. For every $j \in [k]$, the client in Γ_0 sends a query $Q_j = (q_j, c_{j,0}, c_{j,1}, \ldots, c_{j,m})$ and a challenge $I_j = \{0, 1, \ldots, m\}$ to $\mathcal{S}_j$, where $q_j \in \mathbb{F}_p^m$, $c_{j,0}, c_{j,1}, \ldots, c_{j,m} \in (\mathbb{F}_p^*)^{m-1}$. Clearly, $|Q_j| + |I_j|$ contributes $O(m^2)$ to $\mathcal{CC}_{\Gamma_0}$. If we use a common challenge $c_{j,0} = c_{j,1} = \cdots = c_{j,m} = c_j \in (\mathbb{F}_p^*)^{m-1}$ in Γ_0, then $|Q_j| + |I_j|$ would be reduced to $O(m)$ for every $j \in [k]$ and thus $\sum_{j=1}^{k}(|Q_j| + |I_j|)$ will be reduced to $O(m)$. We shall see that this modification will not compromise the security of Γ_0. We denote by Γ_0' this modified version of Γ_0.

In Γ_0', each cloud $\mathcal{S}_j$ answers with $a_j = (a_{j,0}, a_{j,1}, \ldots, a_{j,m})$ and responds with $\sigma_j = (\sigma_{j,0}, \sigma_{j,1}, \ldots, \sigma_{j,m})$, which contributes $|a_j| + |\sigma_j| = O(m) + O(m^2) = O(m^2)$ to $\mathcal{CC}_{\Gamma_0'}$. If we can modify Γ_0' such that only $O(\lambda)$ components of σ_j are needed

to verify a_j, then we would reduce $\mathcal{CC}_{\Gamma_0'}$ to $O(\lambda m)$. A natural idea is performing probabilistic verification: instead of requesting the σ_j from $\mathcal{S}_j$, the client requests λ random components of σ_j, say $\{\sigma_{j,\ell} : \ell \in I_j\}$, where $I_j \subseteq \{0, 1, \ldots, m\}$ is a random λ-subset; the client accepts a_j only if $(a_{j,\ell}, \sigma_{j,\ell})$ verifies for every $\ell \in I_j$. More precisely, $\Gamma_0'.\mathsf{Challenge}$ will simply output k independent random λ-subsets $I_1 \ldots, I_k \subseteq \{0, 1, \ldots, m\}$. We denote by Γ_0'' this modified version of Γ_0'. Note that the modification should not compromise the security of Γ_0''. If a constant fraction (say δ) of the elements of a_j have been tampered with by $\mathcal{S}_j$, then except with negligible probability $(1-\delta)^\lambda$ at least one element of $\{a_{j,\ell} : \ell \in I_j\}$ happens to be tampered with and therefore $\mathcal{S}_j$ will be detected. However, a clever $\mathcal{S}_j$ may tamper with only one element of a_j. This will allow $\mathcal{S}_j$ to deceive the client $\mathcal{C}$ into accepting a_j with non-negligible probability $(1 - \frac{1}{m+1})^\lambda$. Then k such malicious clouds would be able to deceive the client into accepting their answers $a_1, \ldots, a_k$ with non-negligible probability as $\lambda = o(m)$ and $k = O(1)$. Whenever this occurs, the client will compute a wrong value of x_i from those answers, i.e., the security of Γ_0'' is compromised.

In order to thwart this attack, we ask the cloud $\mathcal{S}_j$ to return an encoding A_j of a_j under an error-correcting code $C : \mathbb{F}_p^{m+1} \to \mathbb{F}_p^M$, such that $\mathcal{S}_j$ must change a constant fraction of A_j in order to change even one component of a_j. Furthermore, the client turns to verify A_j. Let C be an $[M, m+1, d']_p$ linear code with constant expansion $\rho = M/(m+1)$ and error rate $\delta = (d'-1)/(2M)$, then Γ_0'' can be further modified as below: the cloud $\mathcal{S}_j$ answers with $A_j = (A_{j,1}, \ldots, A_{j,M}) = C(a_j)$ instead of a_j; the client gives a random λ-subset $I_j \subseteq [M]$ to $\mathcal{S}_j$ and $\mathcal{S}_j$ returns the proofs for $\{A_{j,\ell} : \ell \in I_j\}$; the client then verifies $A_{j,\ell}$ for every $\ell \in I_j$; it accepts and decodes A_j to a_j only if all the verifications are successful. Let Γ_0''' be this modified version of Γ_0''. If such an idea of modifying Γ_0'' can be realized, then each cloud $\mathcal{S}_j$ can deceive the client into accepting $\bar{a}_j \neq a_j$ only if it tampers with $> \delta M$ components of A_j but $\{A_{j,\ell} : \ell \in I_j\}$ are not tampered with. Clearly, this event occurs with probability $\leq (1-\delta)^\lambda$, which is negligible. On the other hand, if $\mathcal{S}_j$ only tampers with $< \delta$ fraction of A_j, then the client can correctly decode a_j from A_j.

A remaining problem is how to choose C such that the idea can be realized, i.e., enable the verifiable computation of A_j. We take C to be an $[M, m+1, M-m]_p$ Reed-Solomon code with constant expansion $\rho = M/(m+1)$ and error rate $\delta = \frac{1}{2} - \frac{1}{2\rho}$. Under this choice each cloud $\mathcal{S}_j$ answers with $A_j = C(a_j) = (f_{a_j}(\gamma_1), f_{a_j}(\gamma_2), \ldots, f_{a_j}(\gamma_M))$, where $f_{a_j}(y) = a_{j,0} + a_{j,1}y + \cdots + a_{j,m}y^m$ and $\gamma_1, \ldots, \gamma_M \in \mathbb{F}_p$ are distinct. Recall that $a_{j,0} = F_x(q_j)$ and $a_{j,\ell} = \frac{\partial F_x}{\partial z_\ell}\big|_{q_j}$ for every $\ell \in [m]$. Let $P_x(\boldsymbol{z}, y) = \mathsf{PolyEnc}_1(x)$. Then $f_{a_j}(\gamma_\ell) = P_x(q_j, \gamma_\ell)$ for every $\ell \in [M]$, where $(q_j, \gamma_\ell) \in \mathbb{F}_p^{m+1}$. Thus, computing A_j is equivalent to evaluating P_x at M points $(q_j, \gamma_1), \ldots, (q_j, \gamma_M)$. The PVC scheme Π_0 can compute m-variate polynomials of total degree $\leq m + d - 1$ which in particular include P_x. However, that requires a public key of size $\binom{2m+d}{m+d-1}$ which is exponential in $n \leq \binom{m}{d}$ as $m = \omega(1)$ and $d = O(1)$. Below we construct a new PVC scheme Π_2 for evaluating P_x. Enforcing the integrity of computing P_x using Π_2 with the common challenge and the probabilistic verification techniques gives us Γ_1.

Let $\mathbb{P}_0$ be the set of all monomials $z_1^{e_1}\cdots z_m^{e_m}$ with $e_1,\ldots,e_m \in \{0,1\}$ and $e_1+\cdots+e_m = d$. Let $\mathbb{P}_1$ be the set of all monomials $y^u \cdot z_1^{e_1}\cdots z_m^{e_m}$ with $u \in [m], e_1,\ldots,e_m \in \{0,1\}$ and $e_1+\cdots+e_m = d-1$. Then

$$P_x \in \mathcal{P} = \mathrm{span}(\mathbb{P}_0 \cup \mathbb{P}_1).$$

For every $i \in [m-1]$, let $\mathbb{B}_i$ be the set of monomials $z_i^{e_i}\cdots z_m^{e_m}$ with $e_{i+2},\ldots,e_m \in \{0,1\}$ and $e_i+\cdots+e_m \le d-1$; let $\mathbb{D}_i$ be the set of monomials $y^u z_i^{e_i}\cdots z_m^{e_m}$ with $u \in [m], e_{i+2},\ldots,e_m \in \{0,1\}$ and $e_i+\cdots+e_m \le d-2$. Let $\mathbb{B}_m = \{y^u z_m^v : 0 \le u+v \le d-1\}$ and $\mathbb{D}_m = \{y^u z_m^v : 0 \le v \le d-2,\ u+v \le m+d-2\}$. We have the following technical lemma.

Lemma 1. *Let $P(\boldsymbol{z},y) \in \mathcal{P}$, $\alpha=(\alpha_1,\ldots,\alpha_{m+1}) \in \mathbb{F}_p^{m+1}$ and $r=(r_1,\ldots,r_m) \in (\mathbb{F}_p^*)^m$. Then there exist $\Phi_1(\boldsymbol{z},y) \in \mathrm{span}(\mathbb{B}_1 \cup \mathbb{D}_1),\ldots,\Phi_m(\boldsymbol{z},y) \in \mathrm{span}(\mathbb{B}_m \cup \mathbb{D}_m)$ and $\phi_0,\ldots,\phi_{m+d-2} \in \mathbb{F}_p$ such that $P(\boldsymbol{z},y)-P(\alpha) = \sum_{i=1}^m (r_i(z_i-\alpha_i)+z_{i+1}-\alpha_{i+1})\Phi_i + (y-\alpha_{m+1})\sum_{j=0}^{m+d-2}\phi_j y^j$. Furthermore, $\sum_{i=1}^m |\mathbb{B}_i| \le (1+O(m^{-1}))\binom{m}{d}$ and $\sum_{i=1}^m |\mathbb{D}_i| \le (d+O(m^{-2}))\binom{m}{d}$.*

We defer the proof of Lemma 1 to the full version. Let $\mathbb{P} = \mathbb{P}_0 \cup \mathbb{P}_1 \cup \cup_{i=1}^m (\mathbb{B}_i \cup \mathbb{D}_i)$. Below is the scheme Π_2 for verifiably evaluating the polynomials in $\mathcal{P}$.

$\mathsf{KeyGen}(1^\lambda,\mathcal{P})$: Pick $\Lambda = (p,\mathbb{G},\mathbb{G}_T,e,g) \leftarrow \mathcal{G}(1^\lambda)$ and $\tau = (\tau_1,\tau_2,\ldots,\tau_{m+1}) \leftarrow \mathbb{F}_p^{m+1}$. Output $sk=\tau$ and $pk=(\Lambda,\mathbb{M})$, where $\mathbb{M} = \{g^{h(\tau)} : h(\boldsymbol{z},y) \in \mathbb{P}\}$.

$\mathsf{Setup}(pk,sk,P)$: Compute and output $vk_P = g^{P(\tau)}$.

$\mathsf{Challenge}(pk)$: Pick $r=(r_1,\ldots,r_m) \leftarrow (\mathbb{F}_p^*)^m$ and output $c=r$.

$\mathsf{Compute}(pk,P,\alpha,c)$: Compute $\Phi_1(\boldsymbol{z},y),\ldots,\Phi_m(\boldsymbol{z},y)$ and $\phi_0,\ldots,\phi_{m+d-2}$ such that the decomposition of $P(\boldsymbol{z},y)-P(\alpha)$ in Lemma 1 holds. Compute $w_i = g^{\Phi_i(\tau)}$ for every $i \in [m]$. Output $a=P(\alpha)$ and $\sigma=(w_1,\ldots,w_m,\phi_0,\ldots,\phi_{m+d-2})$.

$\mathsf{Verify}(pk,vk_P,\alpha,c,a,\sigma)$: If $e(vk_P \cdot g^{-a},g) = \prod_{i=1}^m e\left(g^{r_i(\tau_i-\alpha_i)+\tau_{i+1}-\alpha_{i+1}},w_i\right) \cdot \prod_{j=0}^{m+d-2} e\left(g^{\tau_{m+1}-\alpha_{m+1}},g^{\phi_j \tau_{m+1}^j}\right)$, output 1; otherwise, output 0.

$\mathsf{Update}(pk,sk,vk_P,P')$: Compute $P'(\boldsymbol{z},y)-P(\boldsymbol{z},y)$, say it is equal to $\eta h(\boldsymbol{z},y)$ for $\eta \in \mathbb{F}_p$ and $h(\boldsymbol{z},y) \in \mathbb{P}$. Output $vk_{P'} = vk_P \cdot (g^{h(\tau)})^\eta$.

Lemma 1 shows that $\Phi_1,\ldots,\Phi_m,\ z_1,\ldots,z_m,y,y^2,\ldots,y^{m+d-2}$ and $P(\boldsymbol{z},y)$ all belong to $\mathrm{span}(\mathbb{P})$. Thus the cloud can use pk to compute σ. On the other hand, the $2m+d-2$ components $pk' = (g^{\tau_1},\ldots,g^{\tau_m},g^{\tau_{m+1}},\ldots,g^{\tau_{m+1}^{m+d-2}})$ of pk suffice for executing $\Pi_2.\mathsf{Verify}$. It is a trivial generalization of the security proof in [15] to show Π_2 is secure under the $(m+d-1)$-SBDH assumption.

Theorem 2. *Π_2 is a secure PVC scheme for evaluating the polynomials in $\mathcal{P}$ with $\mathcal{CC}_{\Pi_2} = O(m)$. Its security is based on the $(m+d-1)$-SBDH assumption.*

Due to Lemma 1, we have $|\mathbb{P}| \le (2d+2+o(1))n$. Thus, Π_2 only requires a public key of size $(2d+2+o(1))n$.

The Main Construction (Γ_1). Recall that the polynomial encoding P_x belongs to $\mathcal{P} = \mathrm{span}(\mathbb{P}_0 \cup \mathbb{P}_1)$. Our main construction Γ_1 is described as below.

$\mathsf{KeyGen}(1^\lambda, n)$: Output $(pk, sk) \leftarrow \Pi_2.\mathsf{KeyGen}(1^\lambda, \mathcal{P})$, where $sk = \tau \in (\mathbb{F}_p^*)^m$.

$\mathsf{Setup}(pk, sk, x)$: Output $vk_x \leftarrow \Pi_2.\mathsf{Setup}(pk, sk, P_x)$.

$\mathsf{Query}(pk, i)$: Compute $(q_1, \ldots, q_k, \mathsf{aux}') \leftarrow \Pi_{\mathrm{wy}}.\mathsf{Query}(i)$. For every $j \in [k]$, compute $c_j \leftarrow \Pi_2.\mathsf{Challenge}(pk)$. Output $Q_j = (q_j, c_j)$ for every $j \in [k]$ and $\mathsf{aux} = (\mathsf{aux}', c_1, \ldots, c_k)$.

$\mathsf{Answer}(pk, x, Q_j)$: For every $\ell \in [M]$, compute $(A_{j,\ell}, \sigma_{j,\ell}) \leftarrow \Pi_2.\mathsf{Compute}(pk, P_x, Q_{j,\ell}, c_j)$, where $Q_{j,\ell} = (q_j, \gamma_\ell)$. Output $A_j = (A_{j,1}, A_{j,2}, \ldots, A_{j,M})$.

$\mathsf{Challenge}(pk)$: Output k independent random λ-subsets $I_1, \ldots, I_k \subseteq [M]$.

$\mathsf{Respond}(pk, x, Q_j, I_j)$: Output $\sigma_j = \{\sigma_{j,\ell} : \ell \in I_j\}$.

$\mathsf{Extract}(pk, vk_x, \{(A_j, I_j, \sigma_j) : j \in [k]\}, \mathsf{aux})$: Compute $b_{j,\ell} = \Pi_2.\mathsf{Verify}(pk, vk_x, Q_{j,\ell}, c_j, A_{j,\ell}, \sigma_{j,\ell})$ for every $j \in [k]$ and $\ell \in I_j$. If all $b_{j,\ell}$'s are 1, then decode A_j to $a_j = (a_{j,0}, a_{j,1}, \ldots, a_{j,m})$ for every $j \in [k]$, where $a_{j,0} = F_x(q_j)$ and $a_{j,\ell} = \frac{\partial F_x}{\partial z_\ell}\Big|_{q_j}$ for every $\ell \in [m]$; and output $x_i \leftarrow \Pi_{\mathrm{wy}}.\mathsf{Extract}(a_1, \ldots, a_k, \mathsf{aux}')$. Otherwise, output $\perp$.

$\mathsf{Update}(pk, sk, vk_x, x')$: Suppose that x' is not different from x except that $x'_i \neq x_i$. Suppose that the components of $\mathsf{IndEnc}(i)$ are all 0 except those labeled by $i_1, \ldots, i_d \in [m]$. Then $F_{x'}(\boldsymbol{z}) = F_x(\boldsymbol{z}) + (x'_i - x_i)\pi$, where $\pi = \prod_{j=1}^d z_{i_j}$. It follows that $P_{x'}(\boldsymbol{z}, y) = P_x(\boldsymbol{z}, y) + (x'_i - x_i)\pi(1 + \sum_{j=1}^d z_{i_j}^{-1} y^{i_j})$. The algorithm outputs $vk_{x'} = g^{P_{x'}(\tau)} = vk_x \cdot g^{\Delta}$, where $\Delta = (x'_i - x_i) \cdot \tau_{i_1} \cdots \tau_{i_d}(1 + \sum_{j=1}^d \tau_{i_j}^{-1} \tau_{m+1}^{i_j})$.

Fig. 4. The scheme Γ_1

Correctness, Privacy and Security. The correctness of Γ_1 follows from that of Π_{wy} and Π_2. The unconditional t-privacy of Γ_1 follows from a similar argument as in Γ_0. In Γ_1, any malicious cloud $\mathcal{S}_j$ that tries to deceive the client into computing an incorrect result must change at least $\delta = 1/2 - 1/(2\rho)$ fraction of its answer A_j. The client in Γ_1 verifies λ random components of A_j. Therefore, $\mathcal{S}_j$ will be detected with overwhelming probability $\geq 1 - (1 - \delta)^\lambda$.

Theorem 3. *Γ_1 is an unconditionally t-private and computationally secure VPIR scheme. Its security is based on the the $(m + d - 1)$-SBDH assumption.*

Proof. Suppose there is an adversary $\mathcal{A}$ that breaks the security of Γ_1 with non-negligible probability ϵ. We construct a simulator that simulates $\mathcal{A}$ and breaks the $(m+d-1)$-SBDH assumption. The SBDH challenger picks a random bilinear map instance $\Lambda = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}(1^\lambda)$ and a random field element $s \leftarrow \mathbb{F}_p^*$. Given Λ and $(g, g^s, \ldots, g^{s^{m+d-1}})$, the simulator proceeds as below:

1. The simulator needs to mimic $\Gamma_1.\mathsf{KeyGen}$ and gives a public key pk to $\mathcal{A}$. To do so, the simulator implicitly sets $\tau_\ell = \mu_\ell s + \nu_\ell$ for every $\ell \in [m + 1]$, where μ_ℓ and ν_ℓ are uniformly chosen from $\mathbb{F}_p$; Clearly, the simulator does not know the secret key $\tau = (\tau_1, \ldots, \tau_{m+1}) \in \mathbb{F}_p^{m+1}$ but can compute the public key $\mathbb{M} = \{g^{h(\tau)} : h(\boldsymbol{z}, y) \in \mathbb{P}\}$. In fact, each monomial $h(\boldsymbol{z}, y) \in \mathbb{P}$ has total degree $\leq m + d - 1$. Thus, $h(\tau)$ is a polynomial of degree $\leq m + d - 1$ in s whose coefficients are known to the simulator. Thus, the simulator can compute $g^{h(\tau)}$ as it knows $g, g^s, \ldots, g^{s^{m+d-1}}$; At the end, the simulator picks k random field elements $\theta_1, \ldots, \theta_k \leftarrow \mathbb{F}_p$ and gives $pk = (\Lambda, \mathbb{M})$ to $\mathcal{A}$.

2. Given pk, $\mathcal{A}$ picks $x^{(0)} \in \mathbb{F}_p^n$ and makes a query $\Gamma_1.\mathsf{Setup}(pk, sk, x^{(0)})$ to the simulator. The simulator replies with $vk_{x^{(0)}} = g^{P_0(\tau)}$, where $P_0(\boldsymbol{z}, y) = \mathsf{PolyEnc}_1(x^{(0)})$. For every $u = 1, 2, \ldots, U(= \mathsf{poly}(\lambda))$, $\mathcal{A}$ makes a query $\Gamma_1.\mathsf{Update}(pk, sk, vk_{x^{(u-1)}}, x^{(u)})$ to the simulator. The simulator replies with $vk_{x^{(u)}} = g^{P_u(\tau)}$, where $P_u(\boldsymbol{z}, y) = \mathsf{PolyEnc}_1(x^{(u)})$ and $g^{P_u(\tau)}$ can be computed by the simulator although it does not know $sk = \tau$.
3. The adversary $\mathcal{A}$ picks $u \in \{1, \ldots, U\}$, $i \in [n]$ and gives (u, i) to the simulator. The simulator runs $\Pi_{\mathrm{wy}}.\mathsf{Query}(i)$ and picks k points $q_1, \ldots, q_k \in \mathbb{F}_p^m$ along with a string aux'. Furthermore, the simulator needs to choose a challenge $c_j \in (\mathbb{F}_p^*)^m$, define $Q_j = (q_j, c_j)$ for every $j \in [k]$ and $\mathsf{aux} = (\mathsf{aux}', c_1, \ldots, c_k)$. Note that the adversary may control some of the clouds. For every $j \in [k]$, the challenge c_j must be chosen in a way such that the cloud $\mathcal{S}_j$ can be successfully simulated in order to break the SBDH instance if it is corrupted by $\mathcal{A}$ and provides incorrect answers. Therefore, the simulator will not pick the challenge c_j using the $\Pi_2.\mathsf{Challenge}$ as the client in Γ_1 has done. Instead, for every $j \in [k]$, it will guess an index $\ell_j \in [M]$ such that the cloud $\mathcal{S}_j$ will provide an incorrect answer A_{j,ℓ_j}. For notational convenience, for every $j \in [k]$, we denote $Q_{j,\ell} = (q_j, \gamma_{\ell_j}) = (\alpha_{j,1}, \ldots, \alpha_{j,m+1})$. The simulator will carefully compute $c_j = (r_{j,1}, \ldots, r_{j,m}) \in (\mathbb{F}_p^*)^m$ such that

$$r_{j,\ell}(\tau_\ell - \alpha_{j,\ell}) + \tau_{\ell+1} - \alpha_{j,\ell+1} = s_{j,\ell}(s + \theta_j) \tag{3}$$

for every $\ell \in [m]$. Note that the simulator had set $\tau_\ell = \mu_\ell s + \nu_\ell$ for every $\ell \in [m+1]$. It is easy to verify that (3) will hold for any $s \in \mathbb{F}_p$ when

$$r_{j,\ell} = -\frac{\theta_j \mu_{\ell+1} + \alpha_{j,\ell+1} - \nu_{\ell+1}}{\theta_j \mu_\ell + \alpha_{j,\ell} - \nu_\ell} \text{ and } s_{j,\ell} = \frac{\alpha_{j,\ell}\mu_{\ell+1} - \mu_{\ell+1}\alpha_{j,\ell+1} + \mu_\ell \nu_{\ell+1} - \mu_{\ell+1}\nu_\ell}{\theta_j \mu_\ell + \alpha_{j,\ell} - \nu_\ell}$$

for every $\ell \in [m]$. The simulator defines $Q_j = (q_j, c_j)$ for every $j \in [k]$, $\mathsf{aux} = (\mathsf{aux}', c_1, \ldots, c_k)$ and then gives $(Q_1, \ldots, Q_k)$ to $\mathcal{A}$. Note that c_j is uniform over $(\mathbb{F}_p^*)^m$ due to the choices of $\{\mu_\ell, \nu_\ell : \ell \in [m+1]\}$.
4. $\mathcal{A}$ answers with $(\bar{A}_1, \ldots, \bar{A}_k)$, where $\bar{A}_j = (\bar{A}_{j,1}, \ldots, \bar{A}_{j,M})$ for every $j \in [k]$.
5. For every $j \in [k]$, the simulator picks a random $(\lambda - 1)$-subset $I'_j \subseteq [M]$ such that $\ell_j \notin I'_j$ and then set $I_j = I'_j \cup \{\ell_j\}$. It then gives $(I_1, \ldots, I_k)$ to $\mathcal{A}$. Clearly, the distribution of $(I_1, \ldots, I_k)$ is identical to the distribution of those sets generated by $\Gamma_1.\mathsf{Challenge}$. Therefore, $\mathcal{A}$ cannot distinguish between the simulation and the real execution of Γ_1.
6. $\mathcal{A}$ responds with $(\bar{\sigma}_1, \ldots, \bar{\sigma}_k)$, where $\bar{\sigma}_j = \{\bar{\sigma}_{j,\ell} : \ell \in I_j\}$ for every $j \in [k]$.

To deceive the client into computing an incorrect value of x_i, at least one of the k answers $\bar{A}_1, \ldots, \bar{A}_k$, say $\bar{A}_j$, should have Hamming distance $> M\delta$ with the correct answer $A_j = (A_{j,1}, \ldots, A_{j,M})$, where $A_{j,\ell} = \Pi_2.\mathsf{Compute}(pk, P_u, Q_{j,\ell}, c_j)$ for every $\ell \in [M]$. Otherwise, the client would be able to correct the errors and then compute the correct value of x_i using $\Pi_{\mathrm{wy}}.\mathsf{Extract}$. Furthermore, the forged answers $\bar{A}_1, \ldots, \bar{A}_k$ should not be rejected. Equivalently, all of the forged pairs $\{(\bar{A}_{j,\ell}, \bar{\sigma}_{j,\ell}) : j \in [k], \ell \in I_j\}$ should pass the client's verification, i.e.,

$$\Pi_2.\mathsf{Verify}(pk, vk_{x^{(u)}}, Q_{j,\ell}, c_j, \bar{A}_{j,\ell}, \bar{\sigma}_{j,\ell}) = 1 \tag{4}$$

for every $j \in [k]$ and $\ell \in I_j$ because otherwise the client will reject. Let $\mathbf{E}$ be the event that (1) at least one of the forged answers $\bar{A}_1, \ldots, \bar{A}_k$ has Hamming distance $> M\delta$; and (2) the forged answers $\bar{A}_1, \ldots, \bar{A}_k$ are not rejected, i.e., (4) holds for every $j \in [k]$ and $\ell \in I_j$. Due to our assumption, $\Pr[\mathbf{E}] \geq \epsilon$. For notational convenience, we suppose that the Hamming distance between $\bar{A}_{j^*}$ and A_{j^*} is $> M\delta$, where $j^* \in [k]$. Let $L = \{\ell \in [M] : \bar{A}_{j^*,\ell} \neq A_{j^*,\ell}\} \subseteq [M]$ be the set of indices where $\bar{A}_{j^*}$ and A_{j^*} do not agree with each other. Then $|L| > M\delta$. Let $\mathbf{G}$ be the event that $I_{j^*} \cap L = \emptyset$ and let $\neg\mathbf{G}$ be the event that $I_{j^*} \cap L \neq \emptyset$. Since the indices in I_{j^*} are chosen uniformly and independently, $\Pr[\mathbf{G}] = (1 - |L|/M)^\lambda \leq (1-\delta)^\lambda$, which is negligible. It follows that $\Pr[\mathbf{E}] = \Pr[\mathbf{E}|\mathbf{G}]\Pr[\mathbf{G}] + \Pr[\mathbf{E}|\neg\mathbf{G}]\Pr[\neg\mathbf{G}] \leq (1-\delta)^\lambda + \Pr[\mathbf{E}|\neg\mathbf{G}]\Pr[\neg\mathbf{G}]$. Recall that $\Pr[\mathbf{E}] \geq \epsilon$. We have $\Pr[(\neg\mathbf{G}) \wedge \mathbf{E}] = \Pr[\mathbf{E}|\neg\mathbf{G}]\Pr[\neg\mathbf{G}] \geq \epsilon - (1-\delta)^\lambda \triangleq \epsilon_1$, which is non-negligible. In other words, the event $(\neg\mathbf{G}) \wedge \mathbf{E}$ that $I_{j^*} \cap L \neq \emptyset$ and the forged answers $\bar{A}_1, \ldots, \bar{A}_k$ are accepted must occur with probability $\geq \epsilon_1$. Recall that the indices $\ell_1, \ldots, \ell_k \in [M]$ have been uniformly chosen by the simulator at step 3 of the simulation. Let $\mathbf{F}$ be the event that $\ell_{j^*} \in L$. Note that ℓ_{j^*} is not different from the other indices in I_{j^*} since c_{j^*} is uniform over $(\mathbb{F}_p^*)^m$ and thus gives no information on ℓ_{j^*}. Thus, we have that $\Pr[\mathbf{F}|\neg\mathbf{G} \wedge \mathbf{E}] \geq 1/|I_{j^*}| \geq 1/\lambda$. When $\mathbf{F}$ and $(\neg\mathbf{G}) \wedge \mathbf{E}$ occur simultaneously, $\mathcal{A}$ can be simulated by a simulator in the security game of the Π_2 instance for computing $\bar{A}_{j^*,\ell_{j^*}}$. Using that simulator, the simulator in the security game of Γ_1 can eventually break the $(m+d-1)$-SBDH instance. Note that $\Pr[\mathbf{F} \wedge (\neg\mathbf{G}) \wedge \mathbf{E}] = \Pr[\mathbf{F}|\neg\mathbf{G} \wedge \mathbf{E}]\Pr[(\neg\mathbf{G}) \wedge \mathbf{E}] \geq \epsilon_1/\lambda$, which is non-negligible. Recall the security proof for Π_2. By simulating an adversary that breaks the security of Π_2 with probability ϵ_1/λ, one can break a random $(m+d-1)$-SBDH instance with probability $\geq \lambda^{-1}\epsilon_1(1-(m+d-1)/p)$, which is non-negligible. Therefore, the simulator in the security game of Γ_1 can eventually break the random $(m+d-1)$-SBDH instance with non-negligible probability. Therefore, Γ_1 must be secure under the $(m+d-1)$-SBDH assumption. □

Complexity. The database owner $\mathcal{D}$ runs $\Pi_2.\mathsf{KeyGen}$ and $\Pi_2.\mathsf{Setup}$ once. The running time of $\Pi_2.\mathsf{KeyGen}$ is dominated by the computation of $\mathbb{M} = \{g^{h(\tau)} : h(\boldsymbol{z}, y) \in \mathbb{P}\}$ in Π_2 that requires $(2d + 2 + o(1))n = O(n)$ exponentiations. The algorithm $\Pi_2.\mathsf{Setup}$ computes one exponentiation $g^{P_x(\tau)}$. $\mathcal{D}$ also runs $\Pi_0.\mathsf{Update}$ to update x. To change one component of x, $\mathcal{D}$ needs to multiply vk_x with $d+1$ elements from $\mathbb{M}$, i.e., the update complexity is $d+1 = O(1)$. The client $\mathcal{C}$ computes and sends Q_j, I_j to $\mathcal{S}_j$ and receives A_j, σ_j from $\mathcal{S}_j$ for every $j \in [k]$. We have that $\mathcal{QC}_{\Gamma_0} = \max\{|Q_j| + |I_j|\} = O(m)$, $\mathcal{AC}_{\Gamma_0} = \max\{|A_j| + |\sigma_j|\} = O(\lambda m)$ and $\mathcal{CC}_{\Gamma_0} = \sum_{j=1}^{k}(|Q_j| + |I_j| + |A_j| + |\sigma_j|) = O(\lambda m)$. For each verification, the client uses aux and $2m + d - 2$ elements pk' of pk, where $|\mathsf{aux}| = O(m)$. Each cloud stores x and runs $\Pi_2.\mathsf{Compute}$ M times.

Response Time. A computationally powerful cloud may employ hundreds of thousands of computing units. For example, the Amazon Elastic Compute Cloud (EC2) was employing more than 158000 computing units until May 2013. Mayberry et al. [14] and Devet [6] suggested to outsource the PIR servers' computation to the clouds. Each cloud can distribute the PIR server computation to its computing units and then combine the results of these units. This parallelization

technique can effectively reduce the response time of each cloud. It is trivial to see the computation of each cloud in Γ_1 only involves polynomial decompositions, polynomial valuations and exponentiations. All of them can be distributed to multiple computing units. Thus, each cloud can significantly reduce the response time using the parallelization techniques.

Error Tolerance. In Γ_1, the client verifies A_j by checking a random λ-subset of the components of A_j for every $j \in [k]$. If any one of the λk verifications is unsuccessful, then the client rejects; otherwise, it decodes every A_j to a_j and then reconstructs x_i from $a_1, \ldots, a_k$. Note that if a very small constant fraction of a_j has been corrupted, the client will reject with overwhelming probability. For example, the answer A_j will be rejected with overwhelming probability even if $2\delta/3$ fraction of A_j have been corrupted, where $\delta = \frac{1}{2} - \frac{1}{2\rho}$ and $\rho = M/(m+1)$. Recall that $A_j = C(a_j)$ is a Reed-Solomon encoding that can tolerate δ fraction of corruptions. We say that A_j is slightly corrupted if a constant $\delta' \leq 2\delta/3$ fraction of A_j have been corrupted. The client's verification is so severe that a slightly corrupted A_j will be rejected as well while the corruptions can be efficiently corrected. As a result, the client must execute the scheme Γ_1 again to retrieve x_i, which incurs efficiency loss. In particular, if the corruptions were introduced by the infrastructure failures, then rerunning Γ_1 is unlikely to be useful. In this case we can extend (modify) Γ_1 such that the client will not reject A_j except the event $\mathbf{E} : |\{\ell \in I_j : b_{j,\ell} = 0\}| \geq 3\delta|I_j|/4$ occurs. Suppose δ' fraction of $\{A_{j,\ell} : \ell \in I_j\}$ have been corrupted. If $\delta' \leq 2\delta/3$, then the client will not reject and efficiently reconstruct a_j except with probability $\exp(-O(\lambda))$ as $\mathbf{E}$ occurs with probability $\exp(-O(\lambda))$; if $\delta' > \delta$, then $\mathbf{E}$ will occur with probability $1-\exp(-O(\lambda))$ and thus the client will reject A_j with overwhelming probability.

Multiple Database Delegation. It is an interesting observation that none of the algorithms in Γ_1 actually takes sk as input. As a result, a trusted third party rather than each database owner can run the algorithm KeyGen and then publish pk such that any database owners, clouds and clients can freely use Γ_1. Therefore, Γ_1 actually allows multiple database delegation.

Variant. In Γ_1 the client must communicate with each cloud $\mathcal{S}_j$ in two rounds. In the first round, it sends a query $Q_j = (q_j, c_j)$ and receives an answer A_j; in the second round, it sends a challenge I_j and receives a response σ_j. This two-round communication is essential as sending I_j along with Q_j in the first round would reveal which components of A_j will be verified, and thus enable a cheating cloud to break the security of Γ_1. To get a one-round scheme, the client can consider the M proofs for the M components of A_j as a database and use a single-server PIR [11] with constant communication rate in the honest-but-curious server model to privately retrieve the λ proofs it requires.

4 Conclusions

In this paper, we formally defined verifiable multi-server PIR schemes and constructed an unconditionally t-private and computationally secure VPIR scheme based on the best t-private PIR scheme of Woodruff et al. [17] in the honest-but-curious server model and a new PVC scheme for multivariate polynomial

evaluation. Our scheme has communication complexity $O(\lambda n^{1/\lfloor(2k-1)/t\rfloor})$ which is comparable to [17]. Constructing VPIR schemes in the single server setting seems difficult because the known VC with input privacy rely on FHE.

Acknowledgement. This research is in part supported by Alberta Innovates Technology Futures.

References

1. Beimel, A.: Private Information Retrieval: A Primer (2008) (manuscript)
2. Beimel, A., Ishai, Y., Kushilevitz, E.: General Constructions for Information-Theoretic Private Information Retrieval. J. Comput. Syst. Sci. 71(2), 213–247 (2005)
3. Beimel, A., Ishai, Y., Malkin, T.: Reducing the Servers' Computation in Private Information Retrieval: PIR with Preprocessing. J. Cryptol. 17(2), 125–151 (2004)
4. Beimel, A., Stahl, Y.: Robust Information-Theoretic Private Information Retrieval. J. Cryptol. 20(3), 295–321 (2007)
5. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private Information Retrieval. In: FOCS, pp. 41–50 (1995)
6. Devet, C.: Evaluating Private Information Retrieval on the Cloud. Technical Report (2013), http://cacr.uwaterloo.ca/techreports/2013/cacr2013-05.pdf
7. Devet, C., Goldberg, I., Heninger, N.: Optimally Robust Private Information Retrieval. In: USENIX Security Symposium (2012)
8. Gennaro, R., Gentry, C., Parno, B.: Non-Interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (2010)
9. Goldberg, I.: Improving the Robustness of Private Information Retrieval. In: IEEE Symposium on Security and Privacy, pp. 131–148 (2007)
10. Goldreich, O., Micali, S., Widgerson, A.: How to Play any Mental Game-A Completeness Theorem for Protocols with Honest Majority. In: STOC, pp. 218–229. ACM (1987)
11. Groth, J., Kiayias, A., Lipmaa, H.: Multi-query Computationally-Private Information Retrieval with Constant Communication Rate. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 107–123. Springer, Heidelberg (2010)
12. Huang, Y., Goldberg, I.: Outsourced Private Information Retrieval. In: Workshop on Privacy in the Electronic Society, pp. 119–130. ACM (2013)
13. Kushilevitz, E., Ostrovsky, R.: Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval. In: FOCS, pp. 364–373 (1997)
14. Mayberry, T., Blass, E., Chan, A.: PIRMAP: Efficient Private Information Retrieval for MapReduce. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 371–385. Springer, Heidelberg (2013)
15. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of Correct Computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 222–242. Springer, Heidelberg (2013), http://eprint.iacr.org/2011/587.pdf
16. Sion, R., Carbunar, B.: On the Computational Practicality of Private Information Retrieval. In: NDSS 2007 (2007)
17. Woodruff, D.P., Yekhanin, S.: A Geometric Approach to Information-Theoretic Private Information Retrieval. SIAM J. Comp. 37(4), 1046–1056 (2007)