Private Anonymous Data Access

Ariel Hamlin[*] Rafail Ostrovsky[†] Mor Weiss[‡] Daniel Wichs[§]

April 17, 2018

Abstract

We consider a scenario where a server holds a huge database that it wants to make accessible to a large group of clients. After an initial setup phase, clients should be able to read arbitrary locations in the database while maintaining *privacy* (the server does not learn which locations are being read) and *anonymity* (the server does not learn which client is performing each read). This should hold even if the server colludes with a subset of the clients. Moreover, the run-time of both the server and the client during each read operation should be low, ideally only poly-logarithmic in the size of the database and the number of clients. We call this notion *Private Anonymous Data Access* (PANDA).

PANDA simultaneously combines aspects of *Private Information Retrieval* (PIR) and *Oblivious RAM* (ORAM). PIR has no initial setup, and allows anybody to privately and anonymously access a public database, but the server's run-time is linear in the data size. On the other hand, ORAM achieves poly-logarithmic server run-time, but requires an initial setup after which only a single client with a secret key can access the database. The goal of PANDA is to get the best of both worlds: allow many clients to privately and anonymously access the database as in PIR, while having an efficient server as in ORAM.

In this work, we construct *bounded-collusion* PANDA schemes, where the efficiency scales linearly with a bound on the number of corrupted clients that can collude with the server, but is otherwise poly-logarithmic in the data size and the total number of clients. Our solution relies on standard assumptions, namely the existence of fully homomorphic encryption, and combines techniques from both PIR and ORAM. We also extend PANDA to settings where clients can *write* to the database.

[*]Department of Computer Science, Northeastern University, Boston, Massachusetts, USA, `hamlin.a@husky.neu.edu`.

[†]Department of Computer Science, UCLA, LA, California, USA, `rafail@cs.ucla.edu`.

[‡]Department of Computer Science, Northeastern University, Boston, Massachusetts, USA, `m.weiss@northeastern.onmicrosoft.com`.

[§]Department of Computer Science, Northeastern University, Boston, Massachusetts, USA, `wichs@ccs.neu.edu`.

Contents

1 Introduction

As individuals and organizations increasingly rely on third party data stored remotely, there is often a need to access such data both privately and anonymously. For example, we can envision a service that has a large database of medical conditions, and allows clients to look up their symptoms; naturally clients do not want to reveal which symptoms they are searching for, or even the frequency with which they are performing such searches.

To address this, we consider a setting where a server holds a huge database that it wants to make accessible to a large group of clients. The clients should be able to read various arbitrary locations in the database while hiding from the server which locations are being accessed (*privacy*), and which client is performing each access (*anonymity*). We call this *Private Anonymous Data Access* (PANDA).

In more detail, PANDA allows some initial setup phase, after which the server holds an encoded database, and each client holds a short key. The setup can be performed by a trusted third party, or via a multi-party computation protocol. After the setup phase, any client can execute a read protocol with the server, to retrieve an arbitrary location within the database. We want this protocol to be highly efficient, where both the server's and client's run-time during the protocol should be sub-linear (ideally, poly-logarithmic) in the database size and the total number of clients. For security, we consider an adversarial server that colludes with some subset of clients. We want to ensure that whenever an honest client performs a read access, the server learns nothing about the location being accessed, or the identity of the client performing the access beyond the fact that she belongs to the group of all honest clients. For example, the server should not learn whether two accesses correspond to *two different* clients reading *the same* location of the database, or *one client* reading *two different* database locations.[1]

We call the above a *read-only* PANDA, and also consider extensions that allow clients to write to the database, which we discuss below in more detail.

Connections to PIR and ORAM. PANDA combines aspects of both *Private Information Retrieval* (PIR) [CGKS95, KO97] and *Oblivious RAM* (ORAM) [GO96]. Therefore, we now give a high-level overview of these primitives, their goals, and main properties.

In a (single-database) PIR scheme [KO97], the server holds a public database in the clear. The scheme has no initial setup, and anybody can run a protocol with the server to retrieve an arbitrary location within the database. Notice that since there are no secret keys that distinguish one client from another, a PIR scheme also provides perfect anonymity. However, although the communication complexity of the PIR protocol is sub-linear in the data size, the server's run-time is *inherently* linear in the size of the data. (Indeed, if the server didn't read the entire database during the protocol, it would learn something about the location being queried, since it must be among the ones read.) Therefore, PIR does not provide a satisfactory answer to the PANDA problem, where we want sub-linear efficiency for the server.

In an ORAM scheme, there is an initial setup after which the server holds an encoded database, and a client holds a secret key. The client can execute a protocol with the server to privately read or write to arbitrary locations within the database, and the run-time of both the client and the server during each such protocol is sub-linear in the data size. However, only *a single client* in possession of a secret key associated with the ORAM can access the database. Therefore, ORAM is also not directly applicable to the PANDA problem, where we want a large group of clients to

[1]We assume clients have an anonymous communication channel with the server (e.g., using anonymous mix networks [Cha03] such as TOR [DMS04] or [BG12, LPDH17]).

access the database.

1.1 Prior Work Extending PIR and ORAM

Although neither PIR nor ORAM alone solve the PANDA problem, several prior approaches have considered extensions of PIR and ORAM, aimed to overcome their aforementioned limitations. We discuss these approached, and explain why they do not provide a satisfactory solution for PANDA.

ORAM with Multiple Clients. As mentioned above, in an ORAM scheme *only a single client* can access the database, whereas in PANDA we want multiple clients to access it. There are several natural ways that we can hope to extend ORAM to the setting of multiple clients.

The first idea is to store the data in a single ORAM scheme, and give all the clients the secret key for this ORAM. Although this solution provides anonymity (all clients are identical) it does not achieve privacy; if the server colludes with even a single client, the privacy of all other clients is lost.

A second idea is to store the data in a separate ORAM scheme for each client, and give the client the corresponding secret key. Each client then accesses the data using her own ORAM. This achieves privacy even if the server colludes with a subset of clients, but does not provide anonymity since the server sees which ORAM is being accessed.[2]

The third idea is similar to the previous one, where the data is stored in a separate ORAM scheme for each client, and the client accesses the data using her own ORAM. However, unlike the second idea, the client also performs a "dummy" access on the ORAM schemes of all other clients to hide her identity. This requires a special ORAM scheme where any client *without a secret key* can perform a "dummy" access which looks indistinguishable from a real access to someone that *does not* have the secret key. It turns out that existing ORAM schemes can be upgraded relatively easily to have this property (using re-randomizable encryption). Although this solution achieves privacy and anonymity, the efficiency of both the server and the client during each access is linear in the total number of clients.

Lastly, we can also store the data in a single ORAM scheme on the server, and distribute the ORAM secret key across several additional proxy servers. When a client wants to access a location of the data, she runs a multiparty computation protocol with the proxy servers to generate the ORAM access. Although this solution provides privacy, anonymity and efficiency, it requires having multiple non-colluding servers, whereas our focus is on the *single server* setting.

Variants of the above ideas have appeared in several prior works (e.g., [MBN15, MMRS15, KPK16, BHKP16, ZZQ16]) that explored multi-client ORAM. In particular, the work of Backes et al. [BHKP16] introduced the notion of Anonymous RAM (which is similar to our notion of secret-writes PANDA, discussed below), and proposed two solutions which can be seen as variants of the third and fourth ideas discussed above. Nevertheless, despite much research activity, no prior solution simultaneously provides privacy, anonymity and efficiency in the single-server setting.

Doubly Efficient PIR. As noted above, the server run-time in a PIR protocol is inherently linear in the data size, whereas in PANDA we want the run time of both the client *and the server* to be sub-linear. However, it may be possible to get a *doubly efficient PIR* (DEPIR) variant in which the server run-time is sub-linear, by relaxing the PIR problem to allow a *pre-processing* stage after which the server stores *an encoded version* of the database. This concept was first

[2]Also, the server storage in this solution grows proportionally to the number of clients *times* the data size. Reducing the server storage, even without anonymity, is an interesting relaxation of PANDA which we explore in Appendix 5.3.

proposed by Beimel, Ishai and Malkin [BIM00], who showed how to construct information-theoretic DEPIR schemes in the *multi-server setting*, with several non-colluding servers. Two recent works, of Canetti et al. [CHR17] and Boyle et al. [BIPW17], give the first evidence that this notion may even be achievable in the *single-server* setting. Concretely, they consider DEPIR schemes with a pre-processing stage which generates an encoded database for the server, and a key that allows clients to query the database at arbitrary locations. They distinguish between *symmetric-key* and *public-key* variants of DEPIR, depending on whether the key used to query the database needs to be kept secret or can be made public. Both works show how to construct symmetric-key DEPIR under new, previously unstudied, computational hardness assumptions relating to Reed-Muller codes. The work of [BIPW17] also shows how to extend this to get public-key DEPIR by additionally relying on a heuristic use of obfuscation. Unfortunately, both of the above assumptions are non-standard, poorly understood, and not commonly accepted.

In relation to PANDA, symmetric-key DEPIR suffers from the same drawbacks as ORAM, specifically, only a single client with a secret key can access the database.[3] If we were to give this key to several clients, then all privacy would be lost even if *only a single client* colludes with the server. On the other hand, public-key DEPIR immediately yields a solution to the PANDA problem, at least for the read-only variant. Moreover, it even has additional perks not required by PANDA, specifically: the set of clients does not need to be chosen ahead of time, anybody can use the system given only a public key, and the server is stateless. Unfortunately, we currently appear to be very far from being able to instantiate public-key DEPIR under any standard hardness assumptions.

1.2 Our Results

Read-Only PANDA. In this work, we construct a *bounded-collusion* PANDA scheme, where we assume some upper bound t on the number of clients that collude with the server. The client and server efficiency scales linearly with t, but is otherwise poly-logarithmic in the data size and the total number of clients. In particular, our PANDA scheme allows for up to a poly-logarithmic collusion size t while maintaining poly-logarithmic efficiency for the server and the client. Our construction relies on the generic use of (leveled) *Fully Homomorphic Encryption (FHE)* [RAD78, Gen09] which is in turn implied by the *Learning With Errors (LWE)* assumption [Reg09]. Our basic construction provides security against a semi-honest adversary, and we also discuss how to extend this to get security in the fully malicious setting. In summary, we get the following theorem.

Theorem 1.1 (Informal statement of Theorem 3.12). *Assuming the existence of FHE, there exists a (read-only) PANDA scheme with n clients, t collusion bound, database size L and security parameter λ such that, for any constant $\varepsilon > 0$, we get:*

- *The client/server run-time per* read *operation is* $t \cdot \mathsf{poly}(\lambda, \log L)$.
- *The server storage is* $t \cdot L^{1+\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$.

PANDA with Writes. We also consider extensions of PANDA to a setting that supports writes to the database. If the database is public and shared by all clients, then the location and content of write operations is inherently public as well. However, we still want to maintain privacy and anonymity for read operations, as well as anonymity for write operations. We call this a *public-writes*

[3]The main difference between symmetric-key DEPIR and ORAM is that in the former the server is stateless and only stores a static encoded database, while in the latter the server is stateful and its internal storage is continuously updated after each operation. In PANDA, we allow the server to be stateful.

PANDA and it may, for example, be used to implement a public message board where clients can anonymously post and read messages, while hiding from the server which messages are being read. We also consider an alternate scenario where each client has her own individual private database which only she can access. In this case we want to maintain privacy and anonymity for *both the reads and writes* of each client, so that the server does not learn the content of the data, which clients are accessing their data, or what parts of their data they are accessing. We call this a *secret-writes* PANDA.[4] We show the following results.

Theorem 1.2 (Informal statement of Theorems 4.4 and 4.9)**.** *Assuming the existence of FHE, there exists a* public-writes *PANDA with n clients, t collusion bound, database size L and security parameter λ such that, for any constant $\varepsilon > 0$, we get:*

- *The client/server run-time per* read *operation is $t \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The client run-time per* write *is $O(\log L)$, and the server run-time is $t \cdot L^{\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The server storage is $t \cdot L^{1+\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$.*

The same results as above hold for secret-writes *PANDA, except that the client run-time per* write *increases to $t \cdot \mathsf{poly}(\lambda, \log L)$, and L now denotes the sum of the initial database size and the total number of writes performed throughout the lifetime of the system.*

Extensions. We also consider the PANDA problem in stronger security models in which the adversary can *adaptively* choose the access pattern, and maliciously corrupt parties. Our constructions are also secure in the adaptive setting. Moreover, the read-only PANDA scheme is secure against maliciously-corrupted clients, and a variant of it (which employs Merkle hash trees and succinct interactive arguments of knowledge) is secure if additionally the server is maliciously corrupted. Finally, we discuss modifications of our PANDA with writes schemes that remain secure in the presence of malicious corruptions. See Section 5 for further details.

1.3 Our Techniques

We now give a high-level overview of our PANDA constructions. We start with the read-only setting, and then discuss how to enable writes.

1.3.1 Read-Only PANDA

Our construction relies on *Locally Decodable Codes (LDCs)* [KT00], which have previously been used to construct multi-server PIR [CGKS95, WY05]. We first give an overview of what these are, and then proceed to use them to build our scheme in several steps.

Locally Decodable Codes (LDCs). An LDC consists of a procedure that *encodes* a message into a codeword, and a procedure that *locally decodes* any individual location in the message by reading only few locations in the codeword. We denote the locality by k. An LDC has s-*smoothness* if any s out of k of the codeword locations accessed by the local decoder are uniformly random and independent of the message location being decoded. Such LDCs (with good parameters)

[4]Note that in the read-only setting, having a scheme for a shared public database is strictly more flexible than a scheme for individual private databases. We can always use the former to handle the latter by having clients encrypt their individual data and store it in a shared public database. However, once we introduce writes, these settings become incomparable.

immediately give information-theoretic *multi-server doubly-efficient PIR* without any keys [BIM00]: each of the k servers holds a copy of the encoded database, and the client runs the local decoding procedure by reading each of the k queried codeword locations from a different server. Even if s out of k servers collude, they don't learn anything about the database location that the client is retrieving.[5] LDCs with sufficiently good parameters for our work can be constructed using Reed-Muller codes [Ree54, Mul54].

Initial Idea: LDCs + ORAM. Although LDCs naturally only give a *multi-server* PIR, our initial idea is to think of these as "virtual servers" which will all be emulated by *a single real server* by placing each virtual server under a separate ORAM instance. Each client is assigned a random committee consisting of a small subset of these virtual servers, for which she gets the corresponding ORAM keys. When the adversary corrupts a subset of the clients, it gets all of their ORAM keys, and can therefore be seen as corrupting all the virtual servers that are on the committees of these clients. Nevertheless, we can ensure that the committee of any honest client has sufficiently few corrupted virtual servers for LDC smoothness to hide the client's queries.

In more detail, we think of having k' different virtual servers, for some k' which is sufficiently larger than the locality k of the LDC. For each virtual server, we choose a fresh ORAM key, and store an LDC encoding of the database under this ORAM. Each client is assigned to a random committee consisting of k out of k' of these virtual servers, for which she gets the corresponding ORAM keys. To read a location of the database, the client runs the LDC local decoding algorithm, which requests to see k codeword locations. The client then reads each of the k codeword locations from a different virtual server on her committee, by using the corresponding ORAM scheme. Notice that an adversary that corrupts some subset of t clients, thus obtaining all of their ORAM keys, can be seen as corrupting all the virtual servers on their committees. We can choose the parameters to ensure that the probability of the adversary corrupting more than s out of k of the virtual servers on the committee *of any honest client* is negligible (specifically, setting $k' = tk^2$ and s to be the security parameter). As long as this is the case, our scheme guarantees *privacy*, since the server only learns at most s out of the k codeword locations being queried (by the security of ORAM), and these locations reveal nothing about the database location being read (by the LDC smoothness).

Indeed, although the above solution already gives a non-trivial multi-client ORAM with privacy and low server storage (see further discussion in Appendix 5.3), it does not provide any *anonymity*. The problem is that each client only accesses the k out of k' ORAM schemes belonging to her committee, and doesn't have the keys needed to access the remaining ORAM schemes. Therefore, the server can distinguish between different clients based on which of the ORAMs they access.

One potential idea to fix this issue would be for the client to make some "dummy" accesses to the $k' - k$ remaining ORAM schemes (which are not on her committee) without knowing the corresponding keys. Most ORAM schemes can be easily modified to enable such "dummy" accesses without a key, that look indistinguishable from real accesses to a distinguisher *that doesn't have the key*. Unfortunately, in our case the adversarial colluding server *does have the keys* for many of these ORAM schemes. Therefore, to make this idea work in our setting, we would need an ORAM where clients can make a "*smart* dummy" access without a key that looks indistinguishable from a real access to a random location even to a "smart" distinguisher *that has the key*. The square-root ORAM scheme [Gol87, GO96] can be modified to have this property, but the overall client/server efficiency in the final solutions would be at least square-root of the data size. Unfortunately, more

[5] In standard PIR schemes, the servers hold the original database, and each query is answered by computing the requested codeword symbol on the fly. However, if the codeword size is polynomial, then the servers can compute the codeword first in a preprocessing phase, and then use the pre-computed codeword to answer each query in sub-linear time.

efficient ORAM schemes with poly-logarithmic overhead (such as hierarchical ORAM [Ost90, GO96] or tree-based ORAM [SvDS+13]) do not have this property, and it does not appear that they could be naturally modified to add it. Instead, we take a different approach and get rid of ORAM altogether.

Bounded-Access PANDA: LDCs + Permute. Our second idea is inspired by the recent works of Canetti et al. [CHR17] and Boyle et al. [BIPW17] on DEPIR, as well as earlier works of Hemenway et al. [HO08, HOSW11]. Instead of implementing the virtual servers by storing the LDC codeword under an ORAM scheme, we do something much simpler and use a *Pseudo-Random Permutation* (PRP) to permute the codeword locations. In particular, for each of the k' virtual servers we choose a different PRP key, and use it to derive a different permuted codeword. Each client still gets assigned a random committee consisting of k out of k' of the virtual servers, for which she gets the corresponding PRP keys. To retrieve a value from the database, the client runs the LDC local decoding algorithm, which requests to see k codeword locations, and reads these locations using the virtual servers on her committee by applying the corresponding PRPs. She also reads uniformly random locations from the $k' - k$ virtual servers that are not on her committee.

In relation to the first idea, we can think of the PRP as providing much weaker security than ORAM. Namely, it reveals when *the same* location is read multiple times, but hides *everything else* about the locations being read (whereas an ORAM scheme even hides the former). On the other hand, it is now extremely easy to perform a "smart dummy" access (as informally defined above) by reading a truly random location in the permuted codeword, which is something we don't know how to do with poly-logarithmic ORAM schemes.

It turns out that this scheme is already secure if we fix some a-priori bound B on the total number of read operations that honest clients will perform. We call this notion a *bounded-access PANDA*. Intuitively, even though permuting the codewords provides much weaker security than putting them in an ORAM, and leaks partial information about the access pattern to the codeword, the fact that this access pattern is sampled via a smooth LDC ensures that this leakage is harmless when the number of accesses is sufficiently small. More specifically, our proof follows the high-level approach of Canetti et al. [CHR17], who constructed a *bounded-access (symmetric-key)* DEPIR which is essentially equivalent to the above scheme in the setting with *a single honest* client and *exactly* k virtual servers, where the adversary *doesn't get any of the PRP keys*. In our case, we need to extend this proof to deal with the fact that the adversary colludes with some of the clients, and therefore learns some subset of the PRP keys.

Upgrading to Unbounded-Access PANDA. Our bounded-access PANDA scheme is only secure when the number of accesses is a-priori bounded by some bound B, and can actually be shown to be insecure for sufficiently many accesses beyond that bound (following the analysis of [CHR17]). In the work of Canetti et al. [CHR17] and Boyle et al. [BIPW17], going from bounded-access DEPIR to unbounded-access DEPIR required new non-standard computational hardness assumptions. In our case, we will convert bounded-access PANDA to unbounded-access PANDA using standard assumptions, namely *leveled* FHE which can be instantiated under the LWE assumption. The main reason that we can use our approach for PANDA, but not for DEPIR, is that it makes the server *stateful*. This is something we allow in PANDA, whereas the main goal of DEPIR was to avoid it.

Our idea is essentially to "refresh" the bounded-access PANDA after every B accesses. More specifically, we think of the execution as proceeding in *epochs*, each consisting of B accesses. We associate a different *Pseudo-Random Function* (PRF) key with each virtual server and, for epoch i, we derive an epoch-specific PRP key for each server by applying the corresponding PRF on i. We then use this PRP key to freshly permute the codeword in each epoch. The clients get the PRF keys for the virtual servers in their committee. This lets clients derive the corresponding epoch-specific PRP keys for any epoch, and they can then proceed as they would using the bounded-access PANDA. The only difficulty is making sure that the server can correctly permute the codeword belonging to each virtual server in each epoch without knowing the associated PRF/PRP keys. We do this by storing FHE encryptions of each of the PRF keys on the server and, at the beginning of each epoch, the server performs a homomorphic computation to derive an encryption of the correctly permuted codeword for each virtual server. The clients also get the FHE decryption keys for the virtual servers in their committee, and thus can decrypt the codeword symbols that they read from the virtual servers. Note that the server has to do a large amount of work, linear in the codeword size, at the beginning of each epoch. However, we can use *amortized accounting* to spread this cost over the duration of the epoch and get low amortized complexity. Alternately, the server can spread out the actual computation across the epoch by performing a few steps of it at a time during each access to get low *worst-case* complexity. (This is possible because the database is read-only, and so its contents at the onset of the next epoch are known in advance at the beginning of the current epoch.) The security of this scheme follows from that of the bounded-access PANDA since in each epoch, the read operations are essentially performed using a fresh copy of the bounded-access PANDA (with fresh PRP keys).

PANDA with Public Encoding. Our construction of (unbounded-access) PANDA scheme described above has some nice features beyond what is required by the definition. Specifically, although the server is stateful and its internal state is updated in each epoch, the state can be computed using public information (the FHE encryptions of the PRF keys), the database, and the epoch number.[6] We find it useful to abstract this property further as a *PANDA with public encoding*. Specifically, we think of the PANDA scheme as having a key generation algorithm which doesn't depend on the database, and generates a public-key for the server and secret-keys for each of the clients. The server can then use the public-key to create a fresh encoding of the database with respect to an arbitrary epoch identifier (which can be a number, or an arbitrary bit string). The clients are given the epoch identifier, and can perform read operations which consist of reading some subset of locations from the server. Security holds as long as the number of read operations performed by honest clients with respect to any epoch identifier is bounded by B. Such a scheme can immediately be used to get an unbounded-access PANDA by having the server re-encode the database at the beginning of each epoch with an incremented epoch counter.

Note that our basic security definition considers a semi-honest adversary who corrupts the server and some subset of the clients, but otherwise follows the protocol specification. However, with the above structure, it's also clear that fully malicious clients (who might not follow the protocol) have no affect on the server state, and therefore cannot violate security. A fully malicious server, on the other hand, can lie about the epoch number and cause honest clients to perform too many read operations in one epoch, which would break security. However, if we assume that the epoch number is independently known to honest clients (for example, epochs occur at regular intervals, and clients know the rate at which accesses occur and have synchronized clocks) then this attack is prevented. The only other potential attack for a fully malicious server is to give incorrect values for the locations accessed in the encoded database. We can also prevent this attack by using succinct

[6]For example, the state does not depend on the history of protocol executions with the clients, and is unaffected by client actions. This may be of independent interest even if we downgrade the scheme to the single client setting, and gives the first ORAM scheme we are aware of with this property.

(interactive) arguments to prove that the values were computed correctly.

1.3.2 PANDA with Writes

We also consider PANDA schemes where clients can write to the database, and discuss two PANDA variants in this setting which we call *public-writes* and *secret-writes*.

Public-writes PANDA. In a public-writes PANDA, we consider a setting where the server holds a shared public database which should be accessible to all clients. Clients can write to arbitrary locations in the database but, since the database is public, the locations and the values being written are necessarily public as well. However, we still want to maintain anonymity for the write operations (i.e., the server does not learn which client is performing each write), and both privacy *and* anonymity for the read operations (namely, the server does not learn which client is performing each read, or the locations being read). Our write operation is extremely simple: the client just sends the location and value being written to the server. However, even if we use PANDA with public encoding, the server cannot simply update the encoded database since this would require (at least) linear time.

Instead, we use an idea loosely inspired by hierarchical ORAM [Ost90, GO96]. We will store the database on the server in a sequence of $\log L$ levels, where L is the database size. Each level i consists of a separate instance of a read-only PANDA with public encoding, and will contain at most $L_i = 2^i$ database values. We think of the levels as growing from the top down, namely level-0 (the smallest) is the top-most level, and level-$\log L$ (the largest) is the bottom-most. Initially, all the data is stored in the bottom level $i = \log L$, and all the remaining levels are empty. When a client wants to read some location j of the database, she uses the read-only PANDA for each of the $\log L$ levels to search for location j, and takes the value found in the top-most level that contains it. When a client writes to some location j, the server will place that database value in the top level $i = 0$. The server knows (in the clear) which database values are stored at each level. After every 2^i write operations, the server takes all the values in levels $0, \ldots, i$ and moves them to level $i+1$ by using the public encoding procedure of PANDA and incrementing the epoch counter; level $i+1$ will contain all the values that were previously in levels $\leq i+1$, and levels $0, \ldots, i$ will be emptied.[7] Although the cost of moving all the data to level $i+1$ scales with the data size L_{i+1}, the *amortized* cost is low since this only happens once every 2^i writes.[8]

One subtlety that we need to deal with is that our read-only PANDA was designed as an array data structure which holds L items with addresses $1, \ldots, L$. However, the way we use it in this construction requires a map data structure where the intermediate levels store $L_i \ll L$ items with addresses corresponding to some subset of the values $1, \ldots, L$. We can resolve this using the standard data-structures trick of storing a map in an array by hashing the n addresses into n buckets where each bucket contains some small number of values (to handle collisions). Our final public-writes PANDA scheme can also be thought of as implementing a map data structure, where database entries can be associated with arbitrary bit-strings as addresses, and clients can read/write to the value at any address. We can also allow the total database size to grow dynamically by adding additional levels as needed.

Secret-writes PANDA. In this setting, instead of having a shared public database, we think of each client as having an individual private database which only she can access. We want the

[7]Note that the epoch counters are also incremented, and the encodings are refreshed, when sufficiently many reads occur at that level, just like in the read-only case.

[8]The server complexity can actually be de-amortized using the pipelining trick of Ostrovsky and Shoup [OS97].

clients to be able to read and write to locations in their own database, while maintaining privacy and anonymity so that the server doesn't learn the identity of the client performing each access, the location being accessed, or the content of the data.

Our starting point is the public-writes PANDA scheme, which already guarantees privacy and anonymity of read operations, and anonymity of write operations. The clients can also individually encrypt all their content to ensure that it remains private. Therefore, we only need to modify write operations to provide privacy for the underlying location being written. To achieve this, we rely on the fact that our public-writes PANDA scheme already supports a map data structure, where data can be associated with an arbitrary bit-string as an address. As a first idea, when a client wants to write to some location j in her database, she can use a client-unique PRF, associating the data with the address $PRF(j)$, and then write it using the public-writes scheme. While this partially hides the location j, the server still learns when the same location is written repeatedly. To solve this problem, we also add a counter c, and set the address to be $PRF(j, c)$. Whenever a client wants to read some location j, she uses the read operation of the public-writes PANDA to perform a binary search, and find the largest count c such that there is a value at the address $PRF(j, c)$ in the database. Whenever a client wants to write to location j, she first finds the correct count c (as she would in a read access), and then writes the value to address $PRF(j, c+1)$. This ensures that the address being written reveal no information about the underlying database location. The only downside to this approach is that the server storage grows with the total *number of writes*, rather than the total *data size*. Indeed, since the server cannot correlate different "versions" of the same database location, it cannot delete old copies. Although we view this as a negative, we note that many existing database systems only support "append only" operations, and keep (as a backup) all old versions of the data. Therefore, in such a setting the growth in server storage caused by our scheme does not in fact add any additional overhead.

2 Preliminaries

Throughout the paper λ denotes a security parameter. We use standard cryptographic definitions of Pseudo-Random Permutations (PRPs), Pseudo-Random Functions (PRFs), and Fully Homomorphic Encryption (FHE) (see, e.g., [Gol01, Gol04]). For a vector $\mathbf{a} = (a_1, \ldots, a_n)$, and a subset $S = \{i_1, \ldots, i_s\} \subseteq [n]$, we denote $\mathbf{a}_S = (a_{i_1}, \ldots, a_{i_s})$.

Parameter Names. For all variants of the PANDA problem, we will let n denote the number of clients, L denote the database size, and t denote a bound on the number of corrupted clients colluding with the server.

2.1 Locally Decodable Codes (LDCs).

Locally decodable codes were first formally introduced by [KT00]. We rely on the following definition of smooth LDCs.

Definition 2.1 (Smooth LDC). An s-smooth, k-query *locally decodable code* with message length L, and codeword size M over alphabet Σ, denoted by $(s, k, L, M)_\Sigma$*-smooth LDC*, is a triplet (Enc, Query, Dec) of PPT algorithms with the following properties.

Syntax. Enc is given a message $\mathsf{msg} \in \Sigma^L$ and outputs a codeword $c \in \Sigma^M$,
Query is given an index $\ell \in [L]$ and outputs a vector $\mathbf{r} = (r_1, \ldots, r_k) \in [M]^k$,
and Dec is given $c_\mathbf{r} = (c_{r_1}, \ldots, c_{r_k}) \in \Sigma^k$ and outputs a symbol in Σ.

Local decodability. For every message $\mathsf{msg} \in \Sigma^L$, and every index $\ell \in [L]$,

$$\Pr\left[\mathbf{r} \leftarrow \mathsf{Query}(\ell) \ : \ \mathsf{Dec}\left(\mathsf{Enc}(\mathsf{msg})_{\mathbf{r}}\right) = \mathsf{msg}_\ell\right] = 1.$$

Smoothness. For every index $\ell \in [L]$, the distribution of $(r_1, \ldots, r_k) \leftarrow \mathsf{Query}(\ell)$ is s-wise uniform. In particular, for any subset $S \subseteq [k]$ of size $|S| = s$, the random variables $r_i \ : i \in S$ are uniformly random over $[M]$ and independent of each other.

We will use the Reed-Muller (RM) family of LDCs [Ree54, Mul54] over a finite field $\mathbb{F}$ which, roughly, are defined by m-variate polynomials over $\mathbb{F}$. More specifically, to encode messages in $\mathbb{F}^L$, one chooses a subset $H \subseteq \mathbb{F}$ such that $|H|^m \geq L$. Encoding a message $\mathsf{msg} \in \mathbb{F}^L$ is performed by interpreting the message as a function $\mathsf{msg} : H^m \to \mathbb{F}$, and letting $\widetilde{\mathsf{msg}} : \mathbb{F}^m \to \mathbb{F}$ be the *low degree extension* of msg; i.e., the m-variate polynomial of individual degree $< |H|$ whose restriction to H^m equals msg. The codeword c consists of the evaluations of $\widetilde{\mathsf{msg}}$ at all points if $\mathbb{F}^m$. We can locally decode any coordinate $\ell \in [L]$ of the message by thinking of ℓ as a value in H^m. This is done by choosing a random degree-s curve $\varphi : \mathbb{F} \to \mathbb{F}^m$ such that $\varphi(0) = \ell$, and querying the codeword on $k \geq ms(|H| - 1)$ non-0 points on the curve. The decoder then uses the answers $a_1, \cdots, a_k$ to interpolate the (unique) univariate degree-$(k-1)$ polynomial $\widetilde{\varphi}$ such that $\widetilde{\varphi}(i) = a_i$ for every $1 \leq i \leq k$. It outputs $\widetilde{\varphi}(0)$ as the ℓ'th message symbol. To guarantee that the field contains sufficiently many evaluation points, the field is chosen such that $|\mathbb{F}| \geq k + 1$. The codeword length is $M = |\mathbb{F}|^m$.

Theorem 2.2. *For any constant $\varepsilon > 0$, there exist $(s, k, L, M)_\Sigma$-smooth LDCs with $|\Sigma| = \mathsf{poly}(s, \log L)$, $k = \mathsf{poly}(s, \log L)$ and $M = L^{1+\varepsilon} \cdot \mathsf{poly}(s, \log L)$. Furthermore, the encoding time is $\widetilde{O}(M)$ and the decoding time is $\widetilde{O}(k)$.*

Proof. Let $\hat{s} = \max\{s, \lceil \log L \rceil\}$. For any constant $c \geq 1$, choose H and m such that $|H| = \hat{s}^c$ and $m = \lceil (\log_{\hat{s}} L)/c \rceil$. This satisfies $|H|^m \geq L$ as required. Furthermore $m \leq \hat{s}$. Then we set $k = ms(|H| - 1) \leq \hat{s}^{c+2}$ and we can choose a characteristic-2 field $\Sigma = \mathbb{F}$ of size $k + 1 \leq |\mathbb{F}| \leq 2k \leq \hat{s}^{c+3}$. This gives $M = |\mathbb{F}|^m \leq (\hat{s}^{c+3})^{\lceil (\log_{\hat{s}} L)/c \rceil} \leq (\hat{s}^{c+3})(\hat{s}^{c+3})^{(\log_{\hat{s}} L)/c} \leq L^{1+3/c} \cdot \mathsf{poly}(s, \log L)$. We get the bound in the Theorem by picking $c > 3/\varepsilon$. For the run times, we can rely on FFT techniques for fast multi-variate polynomial interpolation and evaluation. See Appendix A for a further discussion. $\square$

3 Read-Only PANDA

In this section we describe our read-only PANDA scheme. We first formally define this notion. At a high level, a PANDA scheme is run between a server S and n clients $C_1, \cdots, C_n$, and allows clients to *securely* access a database DB, even in the presence of a (semi-honestly) corrupted coalition consisting of the server S and a subset of at most t of the clients. In this section, we focus on the setting of a *read-only, public* database, in which the security guarantee is that read operations of honest clients remain entirely private and anonymous, meaning the corrupted coalition learns nothing about the identity of the client performed the operation, or which location was accessed.

Definition 3.1 (RO-PANDA)**.** A *Read-Only Private Anonymous Data Access (RO-PANDA)* scheme consists of procedures $(\mathsf{Setup}, \mathsf{Read})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^n, 1^t, \mathsf{DB})$ is a function that takes as input a security parameter λ, the number of clients n, a collusion bound t, and a database $\mathsf{DB} \in \{0,1\}^L$, and outputs the initial server state st_S, and client keys $\mathsf{ck}_1, \cdots, \mathsf{ck}_n$. We require that the size of the client keys $|\mathsf{ck}_j|$ is bounded by some fixed polynomial in the security parameter λ, independent of $n, t, |\mathsf{DB}|$.

- Read is a protocol between the server S and a client C_j. The client holds as input an address $\mathsf{addr} \in [L]$ and the client key ck_j, and the server holds its current states st_S. The output of the protocol is a value val to the client, and an updated server state st'_S.

We require the following correctness and security properties.

- **Correctness:** In any execution of the Setup algorithm followed by a sequence of Read protocols between various clients and the server, each client always outputs the correct database value $\mathsf{val} = \mathsf{DB}_{\mathsf{addr}}$ at the end of each protocol.
- **Security:** Any PPT adversary $\mathcal{A}$ has only $\mathsf{negl}(\lambda)$ advantage in the following security game with a challenger $\mathcal{C}$:
  - $\mathcal{A}$ sends to $\mathcal{C}$:
    * The values n, t and the database $\mathsf{DB} \in \{0,1\}^L$.
    * A subset $T \subset [n]$ of corrupted clients with $|T| \le t$.
    * A pair of read sequences $R^0 = \left(j_l^0, \mathsf{addr}_l^0\right)_{1 \le l \le q}, R^1 = \left(j_l^1, \mathsf{addr}_l^1\right)_{1 \le l \le q}$ (for some $q \in \mathbb{N}$) , where $\left(j_l^b, \mathsf{addr}_l^b\right)$ denotes that client $j_l^b \in [n]$ reads address $\mathsf{addr}_l^b \in [L]$.

    We require that $\left(j_l^0, \mathsf{addr}_l^0\right) = \left(j_l^1, \mathsf{addr}_l^1\right)$ for every $l \in [q]$ such that $j_l^0 \in T \vee j_l^1 \in T$.
  - $\mathcal{C}$ performs the following:
    * Picks a random bit $b \leftarrow \{0,1\}$.
    * Initializes the scheme by computing $\mathsf{Setup}\left(1^\lambda, 1^n, 1^t, \mathsf{DB}\right)$.
    * Sequentially executes the sequence R^b of Read protocol executions between the honest server and clients. It sends to $\mathcal{A}$ the views of the server S and the corrupted clients $\{C_j\}_{j \in T}$ during these protocol executions, where the view of each party consists of its internal state, randomness, and all protocol messages received.
  - $\mathcal{A}$ outputs a bit b'.

  The advantage $\mathsf{Adv}_{\mathcal{A}}(\lambda)$ of $\mathcal{A}$ in the security game is defined as: $\mathsf{Adv}_{\mathcal{A}}(\lambda) = |\Pr[b' = b] - \frac{1}{2}|$.

Efficiency Goals. Since a secure PANDA scheme can be trivially obtained by having the client store the entire database locally, or having the server send the entire database to the client in every read request, the *efficiency* of the scheme is our main concern. We focus on minimizing the client storage and the client/server run-time during each Read protocol. At the very least, we require these to be $t \cdot o(|\mathsf{DB}|)$.

Bounded-Access PANDA. We will also consider a weaker notion of a *bounded-access* RO-PANDA scheme, for which security is only guaranteed to hold as long as the total number of read operations q is a-priori bounded. Such schemes will be useful building blocks for designing RO-PANDA schemes with full-fledged security.

Definition 3.2 (B-access RO-PANDA)**.** Let B be an access bound. We say that $(\mathsf{Setup}, \mathsf{Read})$ is a *B-access RO-PANDA scheme* if the security property of Definition 3.1 is only guaranteed to hold for PPT adversaries that are restricted to choose read sequences R^0, R^1 of length $q \le B$.

Remark on Adaptive Security. Note that, for simplicity, our definition is selective, where the adversary chooses the entire read sequences R^0, R^1 ahead of time. We could also consider a stronger adaptive security definition where the adversary chooses the sequence of reads *adaptively* as the protocol progresses. Although our constructions are also secure in the stronger setting (with minimal modifications to the proofs), we chose to present our results in the selective setting to keep them as simple as possible.

3.1 A Bounded-Access Read-Only PANDA Scheme

As a first step, we now show how to construct a bounded-access RO-PANDA scheme, yielding the following theorem.

Theorem 3.3 (B-access RO-PANDA)**.** *Assuming one-way functions exist, for any constant $\varepsilon > 0$ there is a B-bounded access RO-PANDA where, for n clients with t collusion bound and database size L:*

- *The client and server complexity during each* Read *protocol is $t \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The client storage is $t \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The server storage is $\alpha \leq t \cdot L^{1+\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The access bound is $B = \alpha/(t \cdot \mathsf{poly}(\lambda, \log L))$.*

Note that in the above theorem we can increase the access-bound B arbitrarily by artificially inflating the database size L to increase α. However, we will mainly be interested in having a small ratio α/B while keeping α as small as possible.

Construction Outline. As outlined in the introduction, our idea is inspired by the recent works of Canetti et al. [CHR17] and Boyle et al. [BIPW17] on DEPIR. We rely on an s-smooth, k-query LDC where $s = \lambda$ is set to be the security parameter. We think of the server S as consisting of $k' = k^2 t$ different "virtual servers", where t is the collusion bound. Each virtual server contains a permuted copy of the LDC codeword under a fresh PRP. Each client is assigned a random committee consisting of k out of k' of the virtual servers and gets the corresponding PRP keys. To retrieve an entry from the database, the client runs the LDC local decoding algorithm, which requests to see k codeword locations, and reads these locations using the virtual servers on its committee by applying the corresponding PRPs. It also reads uniformly random locations from the $k' - k$ virtual servers that are not on its committee.

Construction 3.4 (B-Access RO-PANDA)**.** The scheme uses the following building blocks:

- An $(s, k, L, M)_\Sigma$-smooth LDC $(\mathsf{Enc}_{\mathsf{LDC}}, \mathsf{Query}_{\mathsf{LDC}}, \mathsf{Dec}_{\mathsf{LDC}})$ (see Definition 2.1, Theorem 2.2).
- A CPA-secure symmetric encryption scheme $\left(\mathsf{KeyGen}_{\mathsf{sym}}, \mathsf{Enc}_{\mathsf{sym}}, \mathsf{Dec}_{\mathsf{sym}}\right)$.
- A pseudorandom permutation (PRP) family $P : \{0,1\}^\lambda \times [M] \to [M]$ where for every $K \in \{0,1\}^\lambda$ the function $P(K, \cdot)$ is a permutation.

The scheme consists of the following procedures:

- $\mathsf{\textbf{Setup}}(1^\lambda, 1^n, 1^t, \mathsf{\textbf{DB}})$**:** Recall that n denotes the number of clients, t is the collusion bound, and $\mathsf{DB} \in \{0,1\}^L$. Instantiate the LDC with message size L and smoothness $s = \lambda$, and let k be the corresponding number of queries, M be the corresponding codeword size and Σ be the alphabet. Set $k' = k^2 t$ to be the number of virtual servers. Proceed as follows.

- Database encoding. Generate the codeword $\widetilde{\mathsf{DB}} = \mathsf{Enc}_{\mathsf{LDC}}(\mathsf{DB})$ with $\widetilde{\mathsf{DB}} \in \Sigma^M$.
- Virtual server generation. For every $1 \le i \le k'$:
  - Generate a PRP key $K^i_{\mathsf{PRP}} \leftarrow \{0,1\}^\lambda$, and an encryption key $K^i_{\mathsf{sym}} \leftarrow \mathsf{KeyGen}_{\mathsf{sym}}(1^\lambda)$.
  - Let $\widehat{\mathsf{DB}}^i \in \Sigma^M$ be a permuted database which satisfies $\widehat{\mathsf{DB}}^i_{P(K^i_{\mathsf{PRP}},j)} = \widetilde{\mathsf{DB}}_j$ for all $j \in [M]$.
  - Let $\widetilde{\mathsf{DB}}^i$ be the encrypted-permuted database with $\widetilde{\mathsf{DB}}^i_j = \mathsf{Enc}_{\mathsf{sym}}\left(K^i_{\mathsf{sym}}, \widehat{\mathsf{DB}}^i_j\right)$.
- Committee generation. For every $j \in [n]$, pick a random size-k subset $\mathsf{S}_j \subseteq [k']$.
- Output. For each client C_j, set the client key $\mathsf{ck}_j = (\mathsf{S}_j, \{K^i_{\mathsf{PRP}}, K^i_{\mathsf{sym}} : i \in \mathsf{S}_j\})$ to consist of the description of the committee and the PRP and encryption keys of the virtual servers on the committee. Set the server state $\mathsf{st}_S = \{\widetilde{\mathsf{DB}}^i : i \in [k']\}$ to consist of the encrypted-permuted databases of every virtual server.

- **The Read protocol.** To read database entry at location $\mathsf{addr} \in [L]$ from the server S, a client C_j with key $\mathsf{ck}_j = (\mathsf{S}_j, \{K^i_{\mathsf{PRP}}, K^i_{\mathsf{sym}} : i \in \mathsf{S}_j\})$ operates as follows.
  - *(Query.)* Denote $\mathsf{S}_j = \{v_1, \ldots, v_k\} \subseteq [k']$. Sample $(r_{v_1}, \cdots, r_{v_k}) \leftarrow \mathsf{Query}_{\mathsf{LDC}}(\mathsf{addr})$, and for each $v \in \mathsf{S}_j$ set $\hat{r}_v = P(K^v_{\mathsf{PRP}}, r_v)$ to be the query to virtual server v. For every $v \in [k'] \setminus \mathsf{S}_j$, pick $\hat{r}_v \in_R [M]$ uniformly.
  - *(Recover).* Send $(\hat{r}_1, \cdots, \hat{r}_{k'})$ to the server S and obtain the answers $\left(\widetilde{\mathsf{DB}}^1_{\hat{r}_1}, \cdots, \widetilde{\mathsf{DB}}^{k'}_{\hat{r}_{k'}}\right)$. For every $v = v_h \in \mathsf{S}_j$, decrypt $a_h = \mathsf{Dec}_{\mathsf{sym}}\left(K^v_{\mathsf{sym}}, \widetilde{\mathsf{DB}}^v_{\hat{r}_v}\right)$, and output $\mathsf{Dec}_{\mathsf{LDC}}(a_1, \cdots, a_k)$.

Remark. Note that in the above construction the server is completely static and stateless. Indeed the Read protocol simply consists of the client retrieving some subset of the locations from the server.

3.1.1 Proof of Security

We prove the following claim about the above construction.

Claim 3.5. *Assuming the security of all of the building blocks, Construction 3.4 is B-bounded-access RO-PANDA for $B = M/(2k^2)$.*

Claim implies Theorem. It's easy to see that Claim 3.5 immediately implies Theorem 3.3 by plugging in the LDC parameters from Theorem 2.2. In particular, for n clients, t collusion bound and database size L:

- The client/server run-time is $k' \log|\Sigma| = tk^2 \log|\Sigma| = t \cdot \mathsf{poly}(\lambda, \log L)$.
- The client storage is $k' \cdot (\mathsf{poly}(\lambda) + \log L) = t \cdot \mathsf{poly}(\lambda, \log L)$.
- The server storage is $\alpha = k' \cdot M \cdot \log|\Sigma| = tk^2 M \cdot \log|\Sigma| = t \cdot L^{1+\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$.
- The bound B is $B = M/(2k^2) = \alpha/(t \cdot \mathsf{poly}(\lambda, \log L))$.

Background Lemmas. To show that the construction is secure, we rely on two lemmas. The first lemma comes from the work of Canetti et al. [CHR17].

Lemma 3.6 (Lemma 1 in [CHR17]). *Let $X = (X_1, \cdots, X_m), Y = (Y_1, \cdots, Y_m)$ be l-wise independent random variables such that for every $1 \leq i \leq m$, X_i, Y_i are identically distributed. Assume also that there is a value $\star$ such that $\Pr[X_i = \star] \geq 1 - \delta$. Then $\mathsf{SD}(X, Y) \leq (m\delta)^{l/2} + m^{l-1}\delta^{l/2-1} \leq 2m^{l-1}\delta^{l/2-1} \leq 2m(m^2\delta)^{l/2-1}$.*

The second lemma deals with the intersection size of random sets.

Lemma 3.7. *Let $T \subseteq [n]$ be an arbitrary set of size $|T| \leq t$. Let $S_1, \cdots, S_n$ be chosen as random subsets $S_j \subseteq [k']$ of size $|S_j| = k$, where $k' = k^2 t$. Then, for all $\rho > 2e$, the probability that there exists some $j \in [n] \setminus T$ such that $|(\cup_{i \in T} S_i) \cap S_j| \geq \rho$ is at most $n \cdot 2^{-\rho}$.*

Proof. Define $E = \bigcup_{i \in T} S_i$ and let $\ell = |E|$ so that $\ell \leq tk$. For any fixed $j \in [n] \setminus T$ the set S_j is random and independent of E and therefore we have

$$\begin{aligned}\Pr[|S_j \cap E| \geq \rho] = \frac{\binom{\ell}{\rho}\binom{k'-\rho}{k-\rho}}{\binom{k'}{k}} &\leq \left(\frac{e\ell}{\rho}\right)^{\rho} \cdot \frac{\binom{k'-\rho}{k-\rho}}{\binom{k'}{k}} \\ &\leq \left(\frac{e\ell}{\rho}\right)^{\rho} \cdot \frac{k(k-1)\cdots(k-\rho+1)}{k'(k'-1)\cdots(k'-\rho+1)} \\ &\leq \left(\frac{e\ell}{\rho}\right)^{\rho}\left(\frac{1}{\ell}\right)^{\rho} \\ &\leq \left(\frac{e}{\rho}\right)^{\rho} \leq \left(\frac{1}{2}\right)^{\rho}\end{aligned}$$

were the third line follows since $\frac{k-x}{k'-x} \leq \frac{k}{k'} \leq \frac{1}{\ell}$ for $0 \leq x \leq \rho - 1$. The lemma follows by a union bound over all $j \in [n]$. $\square$

Proof of Claim. We are now ready to prove Claim 3.5.

Proof of Claim 3.5. The correctness of the scheme follows directly from the correctness of the LDC and the symmetric encryption scheme. We now argue security.

Let $\mathcal{A}$ be a PPT adversary corrupting the server and a subset T of at most t clients. Let R^0, R^1 be the two sequences of read operations of length $q \leq B$ which $\mathcal{A}$ chooses in the security game. Without loss of generality, we can assume that R^0, R^1 do not contain read operations by corrupted clients since $\mathcal{A}$ can generate the corresponding accesses itself (and it does not affect the server state in any way). Let $S_1, \ldots, S_n$ be the random committees chosen during Setup and let $E = \bigcup_{i \in T} S_i$. We proceed via a sequence of hybrids.

$\mathbf{H_1}$ Hybrid H_1 is the security game as in Definition 3.1.

$\mathbf{H_2}$ In hybrid H_2, for all $i \notin E$, we replace the encrypted database $\widetilde{\mathsf{DB}}^i$ by a dummy encryption (e.g.,) of the all 0 string.

Hybrids H_1 and H_2 are computationally indistinguishable by CPA security of the encryption scheme.

$\mathbf{H_3}$ In hybrid H_3, for all $i \notin E$ we replace all calls to the PRP $P(K^i_{\mathsf{PRP}}, \cdot)$ during the various executions of the Read protocol with a truly random permutation $\pi^i : [M] \to [M]$.

Hybrids H_2 and H_3 are computationally indistinguishable by PRP security. Here we rely on the fact that in both hybrids the encrypted database $\widetilde{\mathsf{DB}}^i$ for $i \notin E$ is independent of the permutation.

$\mathbf{H_4}$ In hybrid H_4, if during the committee selection in the Setup algorithm it occurs that there exists some $j \in [n] \setminus T$ such that $|E \cap S_j| \geq s/2$, then the game immediately halts.

Hybrids H_3 and H_4 are statistically indistinguishable by Lemma 3.7, where we set $\rho = s/2$. Recall that $s = \lambda$ and therefore $n \cdot 2^{-\rho} = \mathsf{negl}(\lambda)$.

$\mathbf{H_5}$ In hybrid H_5, we replace the queries $(\hat{r}_1, \cdots, \hat{r}_{k'})$ created during the execution of each Read protocol with truly random values $(u_1, \cdots, u_{k'}) \leftarrow [M]^{k'}$.

The main technical difficulty is showing that hybrids H_4 and H_5 are (statistically) indistinguishable, which we do below. Once we do that, note that hybrid H_5 is independent of the challenge bit b and therefore in hybrid H_5 we have $\Pr[b = b'] = \frac{1}{2}$. Since hybrids H_1 and H_5 are indistinguishable, it means that in hybrid H_1 we must have $|\Pr[b = b'] - \frac{1}{2}| = \mathsf{negl}(\lambda)$ which proves the claim.

We are left to show that hybrids H_4 and H_5 are statistically indistinguishable. We do this by showing that for every Read protocol execution, even if we fix the entire view of the adversary prior to this protocol, the queries sent during the protocol in hybrid H_4 are statistically close to uniform. The protocol is executed by some honest client j with committee $S_j = \{v_1, \ldots, v_k\}$ and we know that $|S_j \cap E| \leq s/2$. Let $(\hat{r}_1, \ldots, \hat{r}_{k'})$ be the distribution on the client queries in the protocol.

(i) For all $v \notin S_j$ the values $\hat{r}_v$ are chosen uniformly at random and independently by the client.

(ii) For $v \in S_j \cap E$, the values $\hat{r}_v = P(K^v_{\mathsf{PRP}}, r_v)$ are uniformly random by the s-wise independence of $\{r_v\}_{v \in S_j}$ and the fact that $|S_j \cap E| \leq s/2$.

(iii) For $v \in S_j \setminus E$, we want to show that the values $\hat{r}_v = \pi^v(r_v)$ are statistically close to uniform, even if we condition on (i),(ii). Note that the values $\{r_v\}_{v \in S_j \setminus E}$ are $s/2$-wise independent even conditioned on the above, and therefore so are the values $\{\hat{r}_v\}_{v \in S_j \setminus E}$. For each v, let $\mathcal{Z}_v \subseteq [M]$ be the set of values $\pi^v(r)$ that were queried in some prior protocol execution by some client. Then $|\mathcal{Z}_v| \leq B$. Note that if $\hat{r}_v = \pi^v(r_v) \notin \mathcal{Z}_v$ then $\hat{r}_v$ is simply uniform over $[M] \setminus \mathcal{Z}_v$ by the randomness of the permutation π^v. We can define random variables X_v where $X_v = \hat{r}_v$ when $\hat{r}_v \in \mathcal{Z}_v$ and $X_v = \star$ otherwise. We can then think of sampling $\{\hat{r}_v\}_{v \in S_j \setminus E}$ by sampling $\{X_v\}_{v \in S_j \setminus E}$ and defining $\hat{r}_v = X_v$ when $X_v \neq \star$ and sampling $\hat{r}_v$ uniformly at random over $[M] \setminus \mathcal{Z}_v$ otherwise. Note that $\{X_v\}_{v \in S_j \setminus E}$ is a set of $|S_j \setminus E| \leq k$ variables which are $s/2$-wise independent and $\Pr[X_v = \star] \geq 1 - \delta$ where $\delta \leq |\mathcal{Z}_v|/M \leq B/M$. Therefore, by applying Lemma 3.6, the variables $\{X_v\}_{v \in S_j \setminus E}$ are statistically close to truly independent variables $\{Y_v\}_{v \in S_j \setminus E}$ such that each Y_v has the same marginal distribution as X_v, where the statistical distance is $2k(k^2 B/M)^{s/4-1} \leq 2k(1/2)^{s/4-1} = \mathsf{negl}(\lambda)$. Replacing the variables $\{X_v\}$ by $\{Y_v\}$ is equivalent to replacing the values $\{\hat{r}_v\}_{v \in S_j \setminus E}$ by truly uniform and independent values.

□

3.2 Public-Encoding PANDA

In this section we describe a *public-encoding* variant of bounded-access RO-PANDA schemes, which will be used in the following sections to construct an unbounded-access RO-PANDA as well as

PANDA schemes that support writes. At a high level, a public-encoding bounded-access PANDA scheme contains a key-generation algorithm KeyGen that generates a public key pk and a set $\{\mathsf{ck}_j\}$ of client secret keys. Any database owner can *locally* encode the database using only the public key. The scheme guarantees privacy and anonymity, even if the adversary obtains a subset of the secret keys, as long as the honest clients make at most B accesses to the database. Furthermore, we allow the server to create many encodings of the same, or different, databases with respect to some labels lab, and the clients can generate accesses using the corresponding label lab. As long as the clients make at most B accesses with respect to any *one* label, security is maintained.

Definition 3.8 (Public-Encoding PANDA). A *public-encoding PANDA (PE-PANDA)* consists of a tuple of algorithms $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Query}, \mathsf{Recover})$ with the following syntax.

- $\mathsf{KeyGen}(1^\lambda, 1^n, 1^t, 1^L)$ is a PPT algorithm that takes as input a security parameter λ, the number of clients n, and the collusion bound t, and a database size L. It outputs a public key pk, and a set of client secret keys $\{\mathsf{ck}_j\}_{j\in[n]}$.
- $\mathsf{Encode}(\mathsf{pk}, \mathsf{DB}, \mathsf{lab})$ is a deterministic algorithm that takes as input a public-key pk, a database DB, and a label lab, and outputs an encoded database $\widetilde{\mathsf{DB}}$.
- $\mathsf{Query}(\mathsf{ck}_j, \mathsf{addr}, \mathsf{lab})$ is a PPT algorithm that takes as input a secret-key ck_j, an address addr in a database, and a label lab, and generates a list $(q_1, \cdots, q_{k'})$ of coordinates in the encoded database.
- $\mathsf{Recover}\left(\mathsf{ck}_j, \mathsf{lab}, \left(\widetilde{\mathsf{DB}}_{q_1}, \cdots, \widetilde{\mathsf{DB}}_{q_{k'}}\right)\right)$ is a deterministic algorithm that takes as input a secret-key ck, a label lab, and a list $\left(\widetilde{\mathsf{DB}}_{q_1}, \cdots, \widetilde{\mathsf{DB}}_{q_{k'}}\right)$ of entries in an encoded database, and outputs a database value val.

We require that it satisfies the following correctness and security properties.

- **Correctness:** For every $\lambda, n, t, L \in \mathbb{N}$, every $\mathsf{DB} \in \{0,1\}^L$, every label $\mathsf{lab} \in \{0,1\}^*$, every address $\mathsf{addr} \in [L]$, and every client $j \in [n]$:

$$\Pr\left[\begin{array}{c} \left(\mathsf{pk}, \{\mathsf{ck}_j\}_{j\in[n]}\right) \leftarrow \mathsf{KeyGen}\left(1^\lambda, 1^n, 1^t, 1^L\right) \\ \widetilde{\mathsf{DB}} = \mathsf{Encode}\left(\mathsf{pk}, \mathsf{DB}, \mathsf{lab}\right) \\ (q_1, \cdots, q_{k'}) \leftarrow \mathsf{Query}\left(\mathsf{ck}_j, \mathsf{addr}, \mathsf{lab}\right) \\ \mathsf{val} = \mathsf{Recover}\left(\mathsf{ck}_j, \mathsf{lab}, \left(\widetilde{\mathsf{DB}}_{q_1}, \cdots, \widetilde{\mathsf{DB}}_{q_{k'}}\right)\right) \end{array} : \mathsf{val} = \mathsf{DB}_{\mathsf{addr}}\right] = 1.$$

- **B-Bounded-Access Security:** Every PPT adversary $\mathcal{A}$ has only $\mathsf{negl}(\lambda)$ advantage in the following security game with a challenger $\mathcal{C}$:
  - $\mathcal{A}$ sends to $\mathcal{C}$ values n, t, L, and a subset $T \subset [n]$ of size $|T| \leq t$.
  - $\mathcal{C}$ executes $\left(\mathsf{pk}, \{\mathsf{ck}_j\}_{j\in[n]}\right) \leftarrow \mathsf{KeyGen}\left(1^\lambda, 1^n, 1^t, 1^L\right)$ and sends pk and $\{\mathsf{ck}_j\}_{j\in T}$ to $\mathcal{A}$. Additionally, $\mathcal{C}$ picks a random bit b.
  - $\mathcal{A}$ is given access to the oracle $\mathsf{Query}^b_{\{\mathsf{ck}_j\}}$ that on input $(j_0, j_1, \mathsf{addr}_0, \mathsf{addr}_1, \mathsf{lab})$ such that $j_0, j_1 \notin T$, outputs $\mathsf{Query}\left(\mathsf{ck}_{j_b}, \mathsf{addr}_b, \mathsf{lab}\right)$.
    We restrict $\mathcal{A}$ to make at most B queries to the oracle with any given label lab, but allow it to make an unlimited number of queries in total.
  - $\mathcal{A}$ outputs a bit b'.

The advantage $\mathsf{Adv}_{\mathcal{A}}(\lambda)$ of $\mathcal{A}$ in the security game is defined as: $\mathsf{Adv}_{\mathcal{A}}(\lambda) = |\Pr[b' = b] - \frac{1}{2}|$.

Next, we construct a public-encoding PANDA scheme, based on our bounded-access PANDA scheme (Construction 3.4 in Section 3.1). The high-level idea is to use fresh PRP keys for every label, by creating them via a PRF applied to the label. The public key of the server contains FHE encryptions of the PRF keys. This enables the server to create the encoded-permuted databases for each virtual server, as in Construction 3.4, by operating on the PRF keys under FHE.

Construction 3.9 (Public-Encoding PANDA)**.** The scheme uses the same building blocks as Construction 3.4. In addition we rely on:

- A pseudo-random function $F : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$.
- The symmetric-key encryption scheme in Construction 3.4 will be replaced by a symmetric-key leveled FHE scheme $(\mathsf{KeyGen}_{\mathsf{FHE}}, \mathsf{Enc}_{\mathsf{FHE}}, \mathsf{Dec}_{\mathsf{FHE}}, \mathsf{Eval}_{\mathsf{FHE}})$.

The scheme consists of the following algorithms:

- $\mathbf{KeyGen}\left(1^\lambda, 1^n, 1^t, 1^L\right)$ operates as follows:
  - Let the parameters s, k, M, k' be chosen the same way as in Construction 3.4.
  - For every virtual server $i \in [k']$:
    - Generates a random FHE key $K^i_{\mathsf{FHE}} \leftarrow \mathsf{KeyGen}_{\mathsf{FHE}}\left(1^\lambda\right)$. We use a leveled FHE that can evaluate circuits up to some fixed polynomial depth $d = \mathsf{poly}(\lambda, \log M)$ specified later.
    - Generates a random PRF key $K^i_{\mathsf{PRF}} \leftarrow \{0,1\}^\lambda$.
    - Encrypts the PRF key: $\widetilde{K}^i_{\mathsf{PRF}} \leftarrow \mathsf{Enc}_{\mathsf{FHE}}\left(K^i_{\mathsf{FHE}}, K^i_{\mathsf{PRF}}\right)$.
  - Generates the random size-k committee $\mathsf{S}_j \subseteq [k']$ for every $1 \le j \le n$.
  - Outputs the public key $\mathsf{pk} = \left(L, \left\{\widetilde{K}^i_{\mathsf{PRF}}\right\}_{i\in[k']}\right)$, and the secret keys $\left\{\mathsf{ck}_j = \left(\mathsf{S}_j, L, \left\{K^i_{\mathsf{PRF}}, K^i_{\mathsf{FHE}} : i \in \mathsf{S}_j\right\}\right)\right\}_{j\in[n]}$.
- $\mathbf{Encode}\left(\mathbf{pk} = \left(L, \left\{\widetilde{K}^i_{\mathbf{PRF}}\right\}_{i\in[k']}\right), \mathbf{DB}, \mathsf{lab}\right)$ operates as follows:
  - Let $\widetilde{\mathsf{DB}} = \mathsf{Enc}_{\mathsf{LDC}}(\mathsf{DB})$ using and LDC with parameters s, k, L, M as in the Setup algorithm of Construction 3.4.
  - For every $i \in [k']$:
    - Generates an encrypted key $\widetilde{K}^i_{\mathsf{PRP}} = \mathsf{Eval}_{\mathsf{FHE}}\left(C_{F,\mathsf{lab}}(\cdot), \widetilde{K}^i_{\mathsf{PRF}}\right)$, where $C_{F,x}(\cdot)$ is the circuit that on input K computes $F(K, x)$.
    - Generate an encrypted-permuted database $\widetilde{\mathsf{DB}}^i = \mathsf{Eval}_{\mathsf{FHE}}\left(C_{P,\widetilde{\mathsf{DB}}}(\cdot), \widetilde{K}^i_{\mathsf{PRP}}\right)$, where $C_{P,\widetilde{\mathsf{DB}}}(\cdot)$ is the circuit that on input K computes the permuted database $\widehat{\mathsf{DB}}$ which satisfies $\widehat{\mathsf{DB}}^i_{P(K,j)} = \widetilde{\mathsf{DB}}_j$ for all $j \in [M]$.
  - Outputs $\left(\widetilde{\mathsf{DB}}^1, \cdots, \widetilde{\mathsf{DB}}^{k'}\right)$.
- **Query**, **Recover**. These algorithms work the same way as the two stages of the Read protocol in Construction 3.4 where the client sets $K^i_{\mathsf{PRP}} := F\left(K^i_{\mathsf{PRF}}, \mathsf{lab}\right)$ and $K^i_{\mathsf{sym}} := K^i_{\mathsf{FHE}}$ for $i \in \mathsf{S}_j$.

Leveled FHE Remark. In the above construction we set parameter d representing the maximum circuit depth for the leveled FHE to be the combined depth of the circuits $C_{F,x}(\cdot)$ and $C_{P,\widetilde{\mathsf{DB}}}(\cdot)$ defined above. Since we can use a permutation network which permutes data of size M in depth $\log M$, so we have $d = \mathsf{poly}(\lambda, \log M)$. We assume that the leveled FHE scheme allows us to compute circuits C of depth d in time $|C| \cdot \mathsf{poly}(\lambda, d)$.

Claim 3.10. *Assuming the security of all of the building blocks, Construction 3.9 is a B-bounded-access secure public-encoding PANDA scheme for $B = M/(2k^2)$.*

Proof. Correctness follows directly from the correctness of Construction 3.4, and the correctness of the underlying LDC and FHE scheme. We now argue that the scheme is secure via a sequence of hybrids. Let $\mathcal{A}$ be a PPT adversary and let T be the subset of at most t clients that it chooses to corrupt. Let $\mathsf{S}_1, \ldots, \mathsf{S}_n$ be the random committees chosen during Setup and let $E = \bigcup_{i \in T} \mathsf{S}_i$. We proceed via a sequence of hybrids which mirrors the proof of Claim 3.5.

$\mathbf{H_1}$ Hybrid H_1 is the security game as in Definition 3.8.

$\mathbf{H_2}$ In hybrid H_2, for all $i \notin E$, we replace $\widetilde{K}^i_{\mathsf{PRF}}$ by an FHE encryption of the all-0 string.

Hybrids H_1 and H_2 are computationally indistinguishable by security of the FHE encryption scheme.

$\mathbf{H_3}$ In hybrid H_3, for all $i \notin E$, whenever the Query algorithm computes $K^i_{\mathsf{PRP}} := F\left(K^i_{\mathsf{PRF}}, \mathsf{lab}\right)$ and uses $P(K^i_{\mathsf{PRP}}, \cdot)$ we instead replace this by a truly random permutation $\pi^{i,\mathsf{lab}}[M] \to [M]$.

Hybrids H_2 and H_3 are computationally indistinguishable by the PRF and PRP security.

$\mathbf{H_4}$ In hybrid H_4, if during the committee selection in the Setup algorithm it occurs that there exists some $j \in [n] \setminus T$ such that $|E \cap \mathsf{S}_j| \geq s/2$ then the game immediately halts.

Hybrids H_3 and H_4 are statistically indistinguishable by Lemma 3.7, where we set $\rho = s/2$. Recall that $s = \lambda$ and therefore $n \cdot 2^{-\rho} = \mathsf{negl}(\lambda)$.

$\mathbf{H_5}$ In hybrid H_5, we replace the values $(q_1, \cdots, q'_k)$ generated by $\mathsf{Query}\left(\mathsf{ck}_{j_b}, \mathsf{addr}_b, \mathsf{lab}\right)$ by uniformly random values $(u_1, \cdots, u_{k'}) \leftarrow [M]^{k'}$.

To show hybrids H_4 and H_5 are indistinguishable, we use a sequence of intermediate hybrids where we do the switch for each distinct label lab queried by the adversary one at a time. Performing each such switch follows the exact same argument as in Claim 3.5 by relying on the fact that each label corresponds to a fresh and independent permutation $\pi^{i,\mathsf{lab}}$.

Note that hybrid H_5 is independent of the challenge bit b and therefore in hybrid H_5 we have $\Pr[b = b'] = \frac{1}{2}$. Since hybrids H_1 and H_5 are indistinguishable, it means that in hybrid H_1 we must have $|\Pr[b = b'] - \frac{1}{2}| = \mathsf{negl}(\lambda)$ which proves the claim. □

Theorem 3.11 (Public-Encoding PANDA)**.** *Suppose leveled FHE exists. Then for any constant $\varepsilon > 0$ there is a PE-PANDA scheme with B-bounded access security, for n clients, t collusion bound and database size L where:*

- *The complexity of Query and $\mathsf{Recover}$ procedures is $t \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The server public key and the client secret keys are each of size $t \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The complexity of the encoding procedure and the size of the encoded database is $\alpha \leq t \cdot L^{1+\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$.*

- *The access bound is* $B = \alpha/(t \cdot \mathsf{poly}(\lambda, \log L))$.

Proof. All of the parameters are set the same way as in Construction 3.4 and Theorem 3.3. We only have to analyze the complexity of the encoding procedure. Here we rely on the fact that $\mathsf{Enc}_{\mathsf{LDC}}(\mathsf{DB})$ can be computed in time $\widetilde{O}(M) = L^{1+\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$. We also rely on the fact that performing operations under leveled FHE only introduces $\mathsf{poly}(\lambda, d)$ multiplicative overhead where $d = \mathsf{poly}(\lambda, \log M) = \mathsf{poly}(\lambda, \log L)$. □

3.3 Read-Only PANDA with Unbounded Accesses

In this section we use the public-encoding PANDA scheme of Section 3.2, which has B-bounded-access security, to obtain a read-only PANDA scheme that is secure against any unbounded number of accesses. Specifically, we will prove the following:

Theorem 3.12 (Read-Only PANDA)**.** *Suppose leveled FHE exists. Then for any constant* $\varepsilon > 0$ *there is a read-only PANDA, for* n *clients,* t *collusion bound and database size* L *where:*

- *The client/server complexity during each* Read *protocol is* $t \cdot \mathsf{poly}(\lambda, \log L)$.
- *The client keys are of size* $t \cdot \mathsf{poly}(\lambda, \log L)$.
- *The server state is of size* $t \cdot L^{1+\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$.

The high-level idea for our construction is conceptually simple: after every B operations, the server re-encodes the database with a fresh label. We think of these sequences of B consecutive accesses as "epochs", and the label is simply a counter indicating the current epoch. The clients get the current epoch number by reading it from the server before performing an access.

Construction 3.13 (Read-only PANDA)**.** The scheme uses a PE-PANDA scheme $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Query}, \mathsf{Recover})$ with B-bounded-access security as a building block. We define the following procedures.

- **Setup**$(1^\lambda, 1^n, 1^t, \mathbf{DB})$**.** Takes as input a security parameter λ, the number of clients n, a collusion bound t, and a database $\mathsf{DB} \in \{0,1\}^L$. It does the following.
  - Counter initialization. Initializes an *epoch counter* count_e, and a *step counter* count_s, to 0.
  - Generating keys. Runs $\left(\mathsf{pk}, \{\mathsf{ck}_j\}_{j\in[n]}\right) \leftarrow \mathsf{KeyGen}\left(1^\lambda, 1^n, 1^t, 1^L\right)$.
  - Encoding the database. Runs $\widetilde{\mathsf{DB}} = \mathsf{Encode}\left(\mathsf{pk}, \mathsf{DB}, \mathsf{count}_e\right)$.
  - Output. For each client $C_j, 1 \leq j \leq n$ set the client key to $\mathsf{ck}_j := \mathsf{ck}_j$. For the server S set $\mathsf{st}_S := (\mathsf{pk}, \widetilde{\mathsf{DB}}, \mathsf{count}_e, \mathsf{count}_s)$.
- **The Read Protocol.** To read the data block at address addr from the server, a client C_j and the server S run the following protocol.
  - The client reads the epoch counter count_e from S.
  - The client runs $(q_1, \cdots, q_{k'}) \leftarrow \mathsf{Query}\left(\mathsf{ck}_j, \mathsf{addr}, \mathsf{count}_e\right)$, and sends $(q_1, \cdots, q_{k'})$ to S.
  - The server computes $a_i = \widetilde{\mathsf{DB}}_{q_i}$ and sends back the values $(a_1, \cdots, a_{k'})$ to the client.
  - The client recovers $\mathsf{DB}_{\mathsf{addr}} = \mathsf{Recover}\left(\mathsf{ck}_j, \mathsf{count}_e, (a_1, \cdots, a_{k'})\right)$.

– The server S updates its state as follows: if $\mathsf{count}_s < B-1$, S updates $\mathsf{count}_s := \mathsf{count}_s + 1$. Otherwise, S updates $\mathsf{count}_s := 0, \mathsf{count}_e := \mathsf{count}_e + 1$, and replaces $\widetilde{\mathsf{DB}} := \mathsf{Encode}\left(\mathsf{pk}, \mathsf{DB}, \mathsf{count}_e\right)$. If the complexity of the computation $\mathsf{Encode}\left(\mathsf{pk}, \mathsf{DB}, \mathsf{count}_e\right)$ is c_{Encode}, the server performs c_{Encode}/B steps of this computation during each protocol execution so that it is completed by the end of the epoch.

Claim 3.14. *Assuming we start with a PE-PANDA scheme that has B-bounded-access security as a building block, then Construction 3.13 is a secure RO-PANDA scheme.*

Proof. The correctness of Construction 3.13 follows directly from the correctness of the underlying PE-PANDA scheme. As for security, since the encoded database is updated every B read requests, and there are at most $\mathsf{poly}\left(\lambda\right)$ read requests overall, any PPT adversary $\mathcal{A}$ in the security game of Definition 3.1 obtains at most B queries to each of $\mathsf{poly}\left(\lambda\right)$ encoded databases (each generated with a unique label). Therefore, security follows directly from the security of the PE-PANDA scheme. □

The proof of Theorem 3.12 now follows.

Proof of Theorem 3.12. We instantiate Construction 3.13 with the PE-PANDA scheme of Theorem 3.11. The client complexity of any read operation entails running Query and $\mathsf{Recover}$ once (and reading the counter), which is $t \cdot \mathsf{poly}(\lambda, \log L)$. The server complexity additionally entails performing $c_{\mathsf{Encode}}/B = \alpha/B \leq t \cdot \mathsf{poly}(\lambda, \log L)$ steps per read operation to re-encode the database once per epoch (here, c_{Encode} is the complexity of the Encode algorithm, and α is the complexity of the encoding procedure of the PE-PANDA scheme). The server storage and the client key sizes are the same as in the PE-PANDA. □

Remark 3.15. We note that when Construction 3.13 is instantiated with the PE-PANDA scheme of Construction 3.9, the concrete server complexity can be improved: instead of freshly encoding the database in each epoch (which requires computing an LDC encoding and then permuting it under FHE) we can just re-permute the LDC-encoded database under the FHE scheme. Since Construction 3.9 uses the RM-LDC, which is quasilinear-time encodable, this modification does not improve the asymptotic complexity of the resultant PANDA scheme. However, it could potentially allow one to replace the RM-LDC with a different LDC that is not quasilinear-time encodable.

4 PANDA With Writes

In this section we extend the read-only scheme of Section 3 to support writes. We first construct (Section 4.1) a PANDA scheme in the *public* database setting. We then design (Section 4.2) a PANDA scheme that supports writes in the *private* database setting.

4.1 A Public-Writes PANDA Scheme

We construct a PANDA scheme for public databases that supports write operations, but only guarantee privacy of read operations, a primitive we call *public-writes PANDA (PW-PANDA)*. Notice that this is the "best possible" security guarantee when there is (even) a (single) corrupted client. (Indeed, as the database is public, a corrupted coalition can always learn what values were written to which locations by simply reading the entire database after every operation.) We note that it suffices to consider this weaker security guarantee *when all clients are honest*, since any public-writes PANDA scheme can be generically transformed into a PANDA scheme which guarantees the privacy of write operations *when all clients are honest*. Indeed, one can implement a (standard) single-client ORAM scheme on top of the public-writes PANDA scheme, for which all clients know the private client key. (We note that the transformation might require FHE-encrypting the PANDA, to allow the server to perform operations on the PANDA which are caused by client operations on the ORAM.)

We now formally define the notion of a public-writes PANDA scheme.

Definition 4.1 (Public-Writes PANDA (PW-PANDA))**.** A *public-writes PANDA (PW-PANDA)* scheme consists of procedures $(\mathsf{Setup}, \mathsf{Read}, \mathsf{Write})$, where $\mathsf{Setup}, \mathsf{Read}$ have the syntax of Definition 3.1, and Write has the following syntax. It is a protocol between the server S and a client C_j. The client holds as input an address $\mathsf{addr} \in [L]$, a value v, and the client key ck_j, and the server holds its current states st_S. The output of the protocol is an updated server state st'_S.

We require the following correctness and security properties.

- **Correctness:** In any execution of the Setup algorithm followed by a sequence of Read and Write protocols between various clients and the server, where the Write protocols were executed with a sequence Q of values, the output of each client in a read operation is the value it would have read from the database if (the prefix of) Q (performed before the corresponding Read protocol) was performed directly on the database.

- **Security:** Any PPT adversary $\mathcal{A}$ has only $\mathsf{negl}(\lambda)$ advantage in the following security game with a challenger $\mathcal{C}$:

  – $\mathcal{A}$ sends to $\mathcal{C}$:
    * The values n, t, and the database $\mathsf{DB} \in \{0,1\}^L$.
    * A subset $T \subset [n]$ of corrupted clients with $|T| \leq t$.
    * A pair of access sequences $Q^0 = \left(\mathsf{op}_l, \mathsf{val}_l^0, j_l^0, \mathsf{addr}_l^0\right)_{1\leq l\leq q}, Q^1 = \left(\mathsf{op}_l, \mathsf{val}_l^1, j_l^1, \mathsf{addr}_l^1\right)_{1\leq l\leq q}$, where $\left(\mathsf{op}_l, \mathsf{val}_l^b, j_l^b, \mathsf{addr}_l^b\right)$ denotes that client j_l^b performs operation op_l at address addr_l^b with value val_l^b (which, if $\mathsf{op}_l = \mathsf{read}$, is $\perp$).

    We require that $\left(\mathsf{op}_l, \mathsf{val}_l^0, j_l^0, \mathsf{addr}_l^0\right) = \left(\mathsf{op}_l, \mathsf{val}_l^1, j_l^1, \mathsf{addr}_l^1\right)$ for every $l \in [q]$ such that $j_l^0 \in T \vee j_l^1 \in T$; and $\left(\mathsf{val}_l^0, \mathsf{addr}_l^0\right) = \left(\mathsf{val}_l^1, \mathsf{addr}_l^1\right)$ for every $l \in [q]$ such that $\mathsf{op}_l = \mathsf{write}$ (in particular, write operations differ only in the identity of the client performing the operation).

  – $\mathcal{C}$ performs the following:
    * Picks a random bit $b \leftarrow \{0,1\}$.
    * Initializes the scheme by computing $\mathsf{Setup}\left(1^\lambda, 1^n, 1^t, \mathsf{DB}\right)$.
    * Sequentially executes the sequence Q^b of Read and Write protocol executions between the honest server and clients. It sends to $\mathcal{A}$ the views of the server S and the corrupted clients $\{C_j\}_{j\in T}$ during these protocol executions, where the view of each party consists of its internal state, randomness, and all protocol messages received.

  – $\mathcal{A}$ outputs a bit b'.

  The advantage $\mathsf{Adv}_{\mathcal{A}}(\lambda)$ of $\mathcal{A}$ in the security game is defined as: $\mathsf{Adv}_{\mathcal{A}}(\lambda) = |\Pr[b' = b] - \frac{1}{2}|$.

Construction Outline. As outlined in the introduction, the public-writes PANDA scheme consists of $\log L$ levels of increasing size (growing from top to bottom), each containing size-λ "buckets" that hold several data blocks, and implemented with a B-bounded-access PE-PANDA scheme. To initialize our PANDA scheme, we generate PE-PANDA public- and secret-keys for every level. Initially, all levels are empty, except for the lowest level, which consists of a PE-PANDA for the database DB. read operations will look for the data block in all levels (returning the top-most copy),[9] whereas write operations will write to the top-most level, causing a reshuffle at predefined intervals to prevent levels from overflowing. We note that adding a *new copy* of the data block (instead of updating the existing data block wherever it is located) allows us to change *only* the content of the top level. This is crucial to obtaining a non-trivial scheme, since levels are implemented using a *read-only* PANDA, and so can only be updated by generating a *new* scheme for the *entire* content of the level, which might be expensive (and so must not be performed too often for lower levels).

Notice that since the levels are implemented using a PE-PANDA scheme (which, in particular, is only secure against a bounded number of accesses), security is guaranteed only as long as each level is accessed at most an a-priori bounded number of times. To guarantee security against *any* (polynomial) number of accesses, we "regenerate" each level when the number of times it has been accessed reaches the bound. This regeneration is performed by running the Encode algorithm of the PE-PANDA scheme with a new label, consisting of the epoch number of the current level and the number of regeneration operations performed during the current epoch (this guarantees that every label is used at most once in each level). In summary, each level can be updated in one of two forms: (1) through a reshuffle operation that merges an upper level into it; or (2) through a regenerate operation, in which the PE-PANDA of the level is updated (but the actual data blocks stored in it do not change). We note that (unlike standard hierarchical ORAM) the reshuffling and regeneration need not be done obliviously, since the server knows the contents of all levels.

As explained in the introduction, we associate a public hash function with each level, which is used to map data blocks into buckets, thus overcoming the issue that the PE-PANDA scheme is designed for an array structure (in particular, reading a certain data block requires knowing its index in the array), whereas the hierarchical structure causes the structure implemented in each level to be a *map*, since levels contain a subset of (not necessarily consecutive) data blocks. (In particular, since this subset depends on previous write operations performed on the PANDA, a client does not know the map structure of the levels, and consequently will not know in which location to look for the desired data block.)

We now formally describe the construction. We assume for simplicity of the exposition that B is a multiple of λ.

Construction 4.2 (Public-writes PANDA)**.** The scheme uses the following building blocks:

- A PE-PANDA scheme $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Query}, \mathsf{Recover})$.
- A hash function family h (used to map data blocks to buckets).

We define the following protocols.

- **Setup**$(1^\lambda, 1^n, 1^t, \mathsf{DB})$**:** Recall that n denotes the number of clients, t is the collusion bound, and $\mathsf{DB} \in \{0,1\}^L$. It does the following.
    - Counter initialization. Initialize a counter count_W to 0. (count_W counts the total number of writes performed so far.)

[9] We note that in standard hierarchical ORAM, once the data block was found, the client should make "dummy" random accesses to lower levels. However, since in our construction each level is implemented as a PE-PANDA scheme which anyway hides the identity of the read operation, we can simply continue looking for the data block in the "right" locations at all levels.

- Generating level counters and keys. For every $1 \leq i \leq \ell$, where $\ell = \log L$ is the number of levels:
  - Run $\left(\mathsf{pk}^i, \left\{\mathsf{ck}_j^i\right\}_{j\in[n]}\right) \leftarrow \mathsf{KeyGen}\left(1^\lambda, 1^n, 1^t, 1^{2^i\cdot\lambda}\right)$.
  - Pick a random hash function h^i for level i.
  - Initialize a *write-epoch counter* count_W^i, a *read-epoch counter* count_R^i, and a *step counter* count_s^i, to 0.[10]
- Initializing level ℓ. Generate an encoded database using the $\mathsf{InitLevel}$ procedure of Figure 1:

$$\widetilde{\mathsf{DB}}^\ell \leftarrow \mathsf{InitLevel}\left(\ell, \mathsf{pk}^\ell, h^\ell, \mathsf{count}_W^\ell, \mathsf{count}_R^\ell, \mathsf{DB}'\right)$$

  where $\mathsf{DB}' = ((1, b_1), \ldots, (L, b_L))$,[11] and set level ℓ to be $\mathsf{L}^\ell = \left(\mathsf{DB}', \widetilde{\mathsf{DB}}^\ell\right)$.
- Output. For each client C_j, set the client key $\mathsf{ck}_j = \left(\left\{\mathsf{ck}_j^i\right\}_{i\in[\ell]}, \left\{h^i\right\}_{i\in[\ell]}\right)$ to consist of its secret keys, and the hash functions, for all levels. Set the server state

$$\mathsf{st}_S = \left(\mathsf{count}_W, \left\{\mathsf{count}_W^i, \mathsf{count}_R^i, \mathsf{count}_s^i\right\}_{i\in[\ell]}, \left\{\mathsf{pk}^i\right\}_{i\in[\ell]}, \left\{h^i\right\}_{i\in[\ell]}, \mathsf{L}^\ell\right)$$

  to consist of all counters, its public keys and the hash functions of all levels, and the contents of level ℓ.

- **The Read protocol.** To read the database value at location $\mathsf{addr} \in [L]$ from the server S, a client C_j with key $\left(\left\{\mathsf{ck}_j^i\right\}_{i\in[\ell]}, \left\{h^i\right\}_{i\in[\ell]}\right)$ and the server S run the following protocol.
  - The client C_j initializes an output value val to $\perp$.
  - C_j performs the following for every non-empty level i from ℓ to 1:
    - Obtaining database label. Read $\mathsf{count}_W^i, \mathsf{count}_R^i$ from S.
    - Computing bucket index. Computes $l = h^i(\mathsf{addr})$. (If addr appears in level i, it will be in bucket Buc_l.)
    - Looking for data block addr in level i. Reads Buc_l from level i, namely for every $(l-1)\cdot\lambda + 1 \leq m \leq l\cdot\lambda$:
      - · Runs $(q_1, \ldots, q_z) \leftarrow \mathsf{Query}\left(\mathsf{ck}_j^i, m, \left(\mathsf{count}_W^i, \mathsf{count}_R^i\right)\right)$, sends $(q_1, \ldots, q_z)$ to S, and obtains answers $(a_1, \ldots, a_z)$.
      - · Runs $\left(\mathsf{addr}', \mathsf{val}'\right) = \mathsf{Recover}\left(\mathsf{ck}_j^i, \left(\mathsf{count}_W^i, \mathsf{count}_R^i\right), (a_1, \ldots, a_z)\right)$.
      - · If $\mathsf{addr}' = \mathsf{addr}$ then set $\mathsf{val} := \mathsf{val}'$.
  - The server S updates its state as follows: if $\mathsf{count}_s^i < B_i - \lambda$, S updates $\mathsf{count}_s^i \leftarrow \mathsf{count}_s^i + \lambda$.[12] Otherwise, S updates $\mathsf{count}_s^i = 0, \mathsf{count}_R^i \leftarrow \mathsf{count}_R^i + 1$, and sets $\widetilde{\mathsf{DB}}^i := \mathsf{Encode}\left(\mathsf{pk}^i, \mathsf{DB}^i, \left(\mathsf{count}_W^i, \mathsf{count}_R^i\right)\right)$ (where $\mathsf{L}^i = \left(\mathsf{DB}^i, \widetilde{\mathsf{DB}}^i\right)$).

[10] count_W^i represents the number of times the level was reshuffled into a lower level, i.e., the number of level-i epochs; count_R^i represents the number of times the underlying PE-PANDA scheme was refreshed, i.e., re-initialized, in the current level-i epoch; and count_s^i represents the number of read operations performed in level i since its underlying PE-PANDA was last refreshed. We note that though count_W^i can be computed from count_W, for simplicity of the exposition we choose to included it explicitly in the description of the scheme.

[11] This guarantees that each data block contains also the logical address of the block, which will be needed when blocks are mapped to buckets.

[12] This is where we use the assumption that λ divides B, otherwise a regeneration of level i mights be needed *while* Buc_l is being read.

- **The Write protocol.** To write value val at location $\mathsf{addr} \in [L]$ on the server S, a client C_j with key $\left(\left\{\mathsf{ck}_j^i\right\}_{i\in[\ell]}, \left\{h^i\right\}_{i\in[\ell]}\right)$ and the server S run the following protocol.
  - The client C_j generates a "dummy" level 0 which contains a single data block $(\mathsf{addr}, \mathsf{val})$, and sends it to the server.
  - The server S updates its state as follows:
    - $\mathsf{count}_W := \mathsf{count}_W + 1$.
    - For $i = 0, 1, \ldots, \ell$ such that 2^i divides count_W, S reshuffles level i into level $i+1$ using the $\mathsf{ReShuffle}$ procedure of Figure 1, namely executes

      $$\mathsf{ReShuffle}\left(i, \mathsf{count}_W^i, \mathsf{count}_R^i, \mathsf{count}_s^i, \mathsf{count}_W^{i+1}, \mathsf{count}_R^{i+1}, \mathsf{count}_s^{i+1}, \mathsf{pk}^{i+1}, h^{i+1}, \mathsf{L}^i, \mathsf{L}^{i+1}\right)$$

      where $\mathsf{L}^i, \mathsf{L}^{i+1}$ are the contents of levels i and $i+1$ (respectively).
      If before executing $\mathsf{ReShuffle}$ for level i, L^{i+1} is empty (following a previous reshuffle, or because it has not yet been initialized), then S first sets $\mathsf{L}^{i+1} := \left(\mathsf{DB}_\emptyset, \widetilde{\mathsf{DB}}^{i+1}\right)$ where $\mathsf{DB}_\emptyset$ is the empty database, and $\widetilde{\mathsf{DB}}$ is generated using the $\mathsf{InitLevel}$ procedure of Figure 1: $\widetilde{\mathsf{DB}}^{i+1} := \mathsf{InitLevel}\left(i+1, \mathsf{pk}^{i+1}, h^{i+1}, \mathsf{count}_W^{i+1}, \mathsf{count}_R^{i+1}, \mathsf{DB}_\emptyset\right)$.

We prove the following claim about Construction 4.2.

Claim 4.3. *Assuming we start with a PE-PANDA scheme that has B-bounded-access security as a building block, and the hash functions are random functions, then Construction 4.2 is a secure PW-PANDA scheme.*

Proof. Correctness follows from the correctness of the underlying PE-PANDA scheme, and the definition of the Read protocol (it returns the top-most copy of the data block, which is also the most recent copy). We now argue security.

Let $\mathcal{A}$ be a PPT adversary corrupting the server and a subset T of at most t clients and let $Q^0 = \left(\mathsf{op}_l, \mathsf{val}_l^0, j_l^0, \mathsf{addr}_l^0\right)_{1\le l\le q}, Q^1 = \left(\mathsf{op}_l, \mathsf{val}_l^1, j_l^1, \mathsf{addr}_l^1\right)_{1\le l\le q}$ denote the sequences of accesses which $\mathcal{A}$ chose in the game. We assume all write operations in Q^0, Q^1 are identical, i.e., are also performed by the same clients. This is without loss of generality, since for corrupted clients this is the case (by the restrictions of Definition 4.1), and write operations of honest clients (may) differ only in the identity of the client performing the operation, which in any case does not influence the operations performed by the Write algorithm.

Let $z_1, \ldots, z_m, m \le q$ be the indices of all accesses in Q^0, Q^1 in which an honest client performs a read access (by the restrictions on Q^0, Q^1 imposed by Definition 4.1, and by our assumption, Q^0, Q^1 differ only in these indices). We proceed via a sequence of hybrids.

$H_{1,1}$ is the security game of Definition 4.1 with $b = 1$.

$H_{1,l}, 2 \le l \le \ell$ is the security game of Definition 4.1, in which the challenger performs Q^0 in all accesses up to (not including) the z_1'st access, reads location $\mathsf{addr}_{z_1}^0$ in the PE-PANDAs of levels $1, \ldots, l-1$, reads location $\mathsf{addr}_{z_1}^1$ in the PE-PANDAs of levels $l, \ldots, \ell$, and performs Q^1 in all remaining accesses.

$H_{k,l}, 2 \le k \le m, 1 \le l \le \ell$**:** is the security game of Definition 4.1, in which the challenger performs Q^0 in all accesses up to (not including) the z_k'th access, reads location $\mathsf{addr}_{z_k}^0$ in the PE-PANDAs of levels $1, \ldots, l-1$, reads location $\mathsf{addr}_{z_k}^1$ in the PE-PANDAs of levels $l, \ldots, \ell$, and performs Q^1 in all remaining accesses.

The $\mathsf{InitLevel}$ procedure

Inputs:

i**:** the index of a level to initialize.

$\mathsf{pk}^i, h^i, \mathsf{count}^i_W, \mathsf{count}^i_R$**:** the public key, hash function, and counters of level i.

DB: a database (of size at most 2^i), consisting of entries of the form $(\mathsf{addr}, \mathsf{val})$.

Operation:

- For every entry $(\mathsf{addr}, \mathsf{val}) \in \mathsf{DB}$, add $(\mathsf{addr}, \mathsf{val})$ to bucket $\mathsf{Buc}^i_{h^i(\mathsf{addr})}$.[a]
- Fill every bucket to size λ using "dummy" blocks of the form $(0, 0)$.[b]
- Run $\widetilde{\mathsf{DB}}^i \leftarrow \mathsf{Encode}\left(\mathsf{pk}^i, \left(\mathsf{Buc}^i_1, \ldots, \mathsf{Buc}^i_{2^i}\right), \left(\mathsf{count}^i_W, \mathsf{count}^i_R\right)\right)$, and output $\widetilde{\mathsf{DB}}^i$.

The $\mathsf{ReShuffle}$ procedure

Inputs:

i**:** the index of a level to reshuffle.

$\mathsf{count}^j_W, \mathsf{count}^j_R, \mathsf{count}^j_s, j \in \{i, i+1\}$**:** the counters of levels $i, i+1$.

$\mathsf{pk}^{i+1}, h^{i+1}$**:** the public-key and hash function of level $i+1$.

$\mathsf{L}^j = \left(\mathsf{DB}^j, \widetilde{\mathsf{DB}}^j\right), j \in \{i, i+1\}$**:** the contents of levels $i, i+1$.

Operation:

- Removing duplicate entries. Merge $\mathsf{DB}^i, \mathsf{DB}^{i+1}$ into a single database DB, where if $(\mathsf{addr}, \mathsf{val}) \in \mathsf{DB}^i$, $\left(\mathsf{addr}, \mathsf{val}'\right) \in \mathsf{DB}^{i+1}$, then DB contains only $(\mathsf{addr}, \mathsf{val})$.[c]
- Update level i. Set level i to be empty, and update $\mathsf{count}^i_W := \mathsf{count}^i_W + 1$, and $\mathsf{count}^i_R = \mathsf{count}^i_s = 0$.
- Update level $i+1$. Run

$$\widetilde{\mathsf{DB}}^{i+1\prime} = \mathsf{InitLevel}\left(i+1, \mathsf{pk}^{i+1}, h^{i+1}, \mathsf{count}^{i+1}_W, \mathsf{count}^{i+1}_R, \mathsf{DB}\right).$$

  Set the new level $i+1$ to be $\mathsf{L}^{i+1} := \left(\mathsf{DB}, \widetilde{\mathsf{DB}}\right)$, and update $\mathsf{count}^{i+1}_R = \mathsf{count}^{i+1}_s = 0$.

[a] We implicitly assume here that no bucket overflows. If a bucket overflows then the server simply aborts the computation. As we show below, this happens only with negligible probability.

[b] We note that "dummy" blocks are not required to be indistinguishable from real blocks, but rather all parties should be able to distinguish between the two. Here, we implicitly assume that "0" is not a valid address, but it could be replaced with any other non-valid address. Alternatively, one could concatenate a bit to every entry, indicating whether it is a real or "dummy" entry.

[c] This can be done in time $O\left(\left|\mathsf{DB}^i\right| + \left|\mathsf{DB}^{i+1}\right|\right)$ by putting both databases in a hash table, and then scanning the table for collisions, and checking, for every collision, whether it is due to multiple copies of the same data block.

Figure 1: Procedures used in Construction 4.2

$H_{m,\ell+1}$: is the security game of Definition 4.1 with $b = 0$.

We now show that every pair of adjacent hybrids are computationally indistinguishable, and consequently so are $H_{1,1}, H_{m,\ell+1}$. Therefore, $\mathsf{Adv}_{\mathcal{A}}(\lambda) = |\Pr[b' = b] - \frac{1}{2}| = \frac{1}{2}|\Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0]| = \mathsf{negl}(\lambda)$.

For $k \in [m], l \in [\ell - 1]$, $H_{k,l}, H_{k,l+1}$ differ only in the z_k'th read operation on the PE-PANDA of level l. In particular, we can fix the public- and secret-keys of the PE-PANDAs, the contents of the PE-PANDAs, and the accesses to these PE-PANDAs, in all levels $l' \neq l$ (this is possible because the keys were generated for each level independently; the contents of the databases in each level are identical in both cases; and each level is accessed independently). After this conditioning, we are left with the security game of Definition 3.8 when $\mathcal{A}$ queries at most $\mathsf{poly}(\lambda)$ PE-PANDAs with keys $\mathsf{pk}^l, \mathsf{ck}_j^l, 1 \leq j \leq n$ (these PE-PANDAs were generated with different labels for database size $2^l \cdot \lambda$). Moreover, each PE-PANDA is accessed at most B times with each label (because the PE-PANDA is refreshed, by running Encode, every B operations). Therefore, indistinguishability follows from the security of the underlying PE-PANDA scheme. The same argument shows that $H_{m,\ell}, H_{m,\ell+1}$ are indistinguishable.

For $k \in [m-1]$, $H_{k,\ell}, H_{k+1,1}$ differ only in the z_k'th read operation on the PE-PANDA of level ℓ (which in $H_{k,\ell}$ is to $\mathsf{addr}_{z_k}^1$ and in $H_{k+1,1}$ is to $\mathsf{addr}_{z_k}^0$). Then $H_{k,\ell} \approx H_{k+1,1}$ by the same argument as above. This completes the security proof.

□

We now use Claim 4.3 to prove the following theorem.

Theorem 4.4 (Public-writes PANDA)**.** *Suppose leveled FHE exists. Then for any constant $\varepsilon > 0$ there is a PW-PANDA, for n clients, t collusion bound and database size L, where:*

- *The client/server complexity during each* Read *protocol is $t \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The client complexity during each* Write *protocol is $O(\log L)$, and the amortized server complexity is $t \cdot L^{\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The client keys are of size $t \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The server state is $t \cdot L^{1+\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$.*

Remark on De-amortization. We note that using a technique of Ostrovsky and Shoup [OS97], the server complexity in Theorem 4.4 can be de-amortized, by slightly modifying Construction 4.2 to allow the server to *spread-out* the reshuffling process. More specifically, we only need to modify the order in which reshuffles are performed in the Write algorithm, such that the operations needed for reshuffle can be executed over multiple accesses to the PANDA. (We note that the server complexity caused by Encode operations in the Read algorithm can be de-amortized as in Construction 3.13.)

Concretely, at time T, let $i \in [\ell] \cup \{0\}$ be the largest such that 2^i divides T (in particular, at time T every level $j \in \{1, \ldots, i\}$ is reshuffled into the next level), then we first reshuffle level i into level $i+1$, then reshuffle level $i-1$ into level i, etc. To see why this "bottom-up" reshuffle strategy is useful, notice that if at time T level j is reshuffled into level $j+1$ (which requires merging levels $j, j+1$, and re-encoding the PE-PANDA of level $j+1$) then the data blocks involved in the reshuffle have been known for the last 2^{j-2} time units. Indeed, level j has not been changed in the last 2^{j-1} time units (since the last time level $j-1$ was reshuffled into it), and level $j-1$ has not been changed in the last 2^{j-2} time units (since the last time level $j-2$ was reshuffled into it). Therefore, the operations needed to performed the reshuffle of level j into level $j+1$ (which in Theorem 4.4 requires at most $t \cdot \left(2^j \cdot \lambda\right)^{1+\epsilon} \cdot \mathsf{poly}(\lambda, \log L)$ operations) can be spread out over 2^{j-2} operations.

Proof of Theorem 4.4. We instantiate Construction 4.2 with the PE-PANDA schemes of Theorem 3.11, and instantiate the hash functions using a PRF F (i.e., h^i is defined by picking a random PRF key K_i and setting $h^i(\cdot) = F(K_i, \cdot)$).

The client complexity of any read operation entails running Query and $\mathsf{Recover}$ λ times for each level (as well as reading the counters, and computing the hash), which is $t \cdot \mathsf{poly}(\lambda, \log L)$. The server complexity additionally entails updating $O(\log L)$ counters, and performing (amortized) $c_{\mathsf{Encode}}/B = \alpha/B \leq t \cdot \mathsf{poly}(\lambda, \log L)$ steps per read operation to re-encode the database once every B read operations (here, c_{Encode} is the complexity of the Encode algorithm, and α is the complexity of the encoding procedure of the PE-PANDA scheme). The client complexity of any write operation is $O(\log L)$ (the client simply generates an entry, which includes an address of size $\log L$). The server complexity additionally entails reshuffling level-i into level-$(i+1)$ every 2^i write operations, performing $c_{\mathsf{Encode}}\left(2^i \cdot \lambda, B\right) = t \cdot \left(2^i \cdot \lambda\right)^{1+\varepsilon} \cdot \mathsf{poly}(\lambda, \log\left(2^i \cdot \lambda\right))$ operations, so the amortized complexity per write operation is $t \cdot \left(2^i\right)^{\varepsilon} \cdot \mathsf{poly}(\lambda, \log L) \leq t \cdot L^{\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$. The server storage and the client key sizes are $\log L$ times the storage and key sizes of the PE-PANDA (the server also stores $\log L$ counters of size at most $\mathsf{poly}(\lambda, \log L)$), i.e., have size $t \cdot L^{1+\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$ and $t \cdot \mathsf{poly}(\lambda, \log L)$, respectively.

Finally, we show that a bucket overflows only with negligible probability. For simplicity of the analysis, we model the PRF as a random function. Whenever $\mathsf{InitLevel}$ is called for some level $i \in [\ell]$, the corresponding database contains at most 2^i data blocks, which are then allocated to the 2^i size-λ buckets. Therefore, in expectation every specific bucket Buc will contain a single data block, and so by Chernoff's bound the probability that Buc overflows is at most

$$\frac{e^{\lambda-1}}{\lambda^{\lambda}} \leq 2^{-\lambda}$$

where the inequality holds for a large enough $\lambda \geq 2e$. Since the total number of buckets in all levels is at most $L \cdot \log L = \mathsf{poly}(\lambda)$, the probability that *any* bucket overflows is negligible. □

4.1.1 A Public-Writes PANDA Scheme for Map Structures and Dynamic Databases

We show that the PW-PANDA of Construction 4.2 can be extended to support map structures (in which the data blocks are not necessarily consecutive), and dynamic databases that grow throughout the execution. This will be useful when constructing PANDA schemes in the private-database setting in the following section.

Construction Outline. Though Construction 4.2 was described for databases with an array structure, it can actually be used for databases given as map structures, because the data is organized in each level using a hash function. Therefore, it remains to describe how to modify the construction to support dynamic databases. Let DB be the initial database of size L, and $L_{\mathsf{max}} \in \mathbb{N}$ be an upper bound on its maximal size during the execution (we assume that $L_{\mathsf{max}} = \mathsf{poly}(\lambda)$)[13]. Construction 4.2 is instantiated with the size bound L_{max}, and initialization generates level counters and keys for levels $1, \ldots, \log L_{\mathsf{max}}$ (i.e., for all levels that *might* be needed throughout the execution), where *all levels are initially empty.* Additionally, the construction generates a RO-PANDA scheme for DB. A new data block is added to the PANDA in the same manner that an existing block is written to the database, and new levels are initialized when needed, just as levels $1, \ldots, \log L$ are

[13]This assumption is only needed to bound the probability a bucket overflows. In fact, L_{max} can even be taken to be 2^{λ}, as long as we are guaranteed that the database size throughout the execution is always $\mathsf{poly}(\lambda)$. Setting $L_{\mathsf{max}} = 2^{\lambda}$ gives a scheme in which there is no a-priori polynomial bound on the database size (i.e., it can grow to any polynomial size).

generated in Construction 4.2. To read a data block, one looks for it both in the RO-PANDA, and in the hierarchical structure.

The parameters of the resultant scheme are summarized in the next theorem, where we say that DB is a *dynamic database of size at most L_{max}* if it is a dynamic database (i.e., to which data blocks may be added) whose size never exceeds L_{max}.

Theorem 4.5 (PW-PANDA for map structures and dynamic databases)**.** *Suppose leveled FHE exists. Then for any constant $\varepsilon > 0$ and database size bound L_{max} there is a PW-PANDA scheme for dynamic databases of size at most L_{max}, for n clients, and t collusion bound, where when the database has size L:*

- *The client and server complexity during each Read protocol is $\mathsf{poly}(\log L_{\mathsf{max}}) + t \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The client complexity during each Write protocol is $O(\log L + \log L_{\mathsf{max}})$.*
- *The amortized server complexity during each Write protocol is $\mathsf{poly}(\log L_{\mathsf{max}}) + t \cdot L^{\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$.*[14]
- *The client keys are of size $t \cdot \mathsf{poly}(\lambda, \log L_{\mathsf{max}})$.*
- *The server state is $t \cdot \mathsf{poly}(\lambda, \log L_{\mathsf{max}}) + t \cdot L^{1+\epsilon} \cdot \mathsf{poly}(\lambda, \log L)$.*

Proof. We first describe the scheme $(\mathsf{Setup}', \mathsf{Read}', \mathsf{Write}')$ for dynamic databases, which uses the protocols $(\mathsf{Setup}, \mathsf{Read}, \mathsf{Write})$ of Construction 4.2 (which, in turn, uses the PE-PANDA scheme of Theorem 3.11), and a RO-PANDA scheme $(\mathsf{Setup}^R, \mathsf{Read}^R)$. (We note that instead of a RO-PANDA scheme, one can use Construction 4.2, which is also a RO-PANDA scheme. However, using a different RO-PANDA scheme might yield a more efficient construction.)

- $\mathsf{Setup}'(1^\lambda, 1^n, 1^t, 1^L, L_{\mathsf{max}}, \mathsf{DB})$ runs $\mathsf{Setup}(1^\lambda, 1^n, 1^t, 1^{L_{\mathsf{max}}}, \mathsf{DB})$,[15] but does not initialize the lowest level. (In particular, after initialization all levels of the PW-PANDA are empty.) Additionally, it runs $\mathsf{Setup}^R(1^\lambda, 1^n, 1^t, \mathsf{DB})$ to generate a RO-PANDA for DB. It outputs to each client C_j a key $\mathsf{ck}'_j = (\mathsf{ck}^R_j, \mathsf{ck}_j)$ consisting of its keys $\mathsf{ck}^R_j, \mathsf{ck}_j$ for the RO-PANDA and PW-PANDA (respectively), and a server state $\mathsf{st}'_S = (\mathsf{st}^R_S, \mathsf{st}_S)$ consisting of its states $\mathsf{st}^R_S, \mathsf{st}_S$ in the RO-PANDA and PW-PANDA, respectively.
- $\mathsf{Read}'(\mathsf{addr}, \mathsf{ck}'_j = (\mathsf{ck}^R_j, \mathsf{ck}_j), \mathsf{st}'_S = (\mathsf{st}^R_S, \mathsf{st}_S))$ Runs $\mathsf{Read}^R(\mathsf{addr}, \mathsf{ck}^R_j, \mathsf{st}^R_S)$, $\mathsf{Read}(\mathsf{addr}, \mathsf{ck}_j, \mathsf{st}_S)$ to look for addr in the RO-PANDA and the PW-PANDA. If the data block was found in both, then the copy in the PW-PANDA is returned (since it is newer).
- $\mathsf{Write}'(\mathsf{addr}, \mathsf{val}, \mathsf{ck}'_j = (\mathsf{ck}^R_j, \mathsf{ck}_j), \mathsf{st}'_S = (\mathsf{st}^R_S, \mathsf{st}_S))$ simply runs $\mathsf{Write}(\mathsf{addr}, \mathsf{val}, \mathsf{ck}_j, \mathsf{st}_S)$ to write value val to address addr in the database stored in the PW-PANDA.

We now show that $(\mathsf{Setup}', \mathsf{Read}', \mathsf{Write}')$ has the desired properties. Correctness follows directly from the correctness of Construction 4.2 (Theorem 4.4), and the correctness of the RO-PANDA scheme (which can be initialized using Construction 4.2).

Regarding security, let $\mathcal{A}$ be a PPT adversary corrupting the server and a subset T of at most t clients, and let $Q^0 = (\mathsf{op}_l, \mathsf{val}^0_l, j^0_l, \mathsf{addr}^0_l)_{1 \le l \le q}, Q^1 = (\mathsf{op}_l, \mathsf{val}^1_l, j^1_l, \mathsf{addr}^1_l)_{1 \le l \le q}$ denote the sequences of accesses which $\mathcal{A}$ chose in the security game of Definition 4.1. We proceed through a sequence of hybrids.

[14] The server complexity can be de-amortized similarly to Theorem 4.4.

[15] We note that if L_{max} is super-polynomial then even passing $1^{L_{\mathsf{max}}}$ to Setup cannot be done in $\mathsf{poly}(\lambda)$ time. However, Setup' can emulate Setup without actually writing down $1^{L_{\mathsf{max}}}$, and since the emulation only instantiates $\log L_{\mathsf{max}}$ empty levels, the emulation would be efficient.

H_1^b is the security game of Definition 4.1 (with the modification that new entries may be written to the database) with bit b.

H_2^b is the same as H_1^b, except that all calls to Read^R (during the execution of the Read' or Write' protocols) induced by read or write operations of honest clients, are replaced with read requests to address 1 (or any other arbitrary address).

$H_1^b \approx H_2^b$ by the security of the RO-PANDA scheme.

To complete the proof, we show that the views of $\mathcal{A}$ in H_2^0, H_2^1 are computationally indistinguishable (and so its advantage in the security game is negligible). This follows from the security of the PW-PANDA scheme: both views contain the same write operations, and the same Read^R calls (for calls induced by read operations of honest clients this holds because they read address 1; for calls induces by read operations of corrupted clients this holds because these operations read the same address, by the restrictions in the security game of Definition 4.1). Therefore, the views differ only in read accesses of honest clients to the PW-PANDA, or in the identity of the client in write operations of honest clients on the PW-PANDA, which are indistinguishable by the security of the PW-PANDA scheme.

As for the complexity of the scheme, we follow the analysis in the proof of Theorem 4.4. (For the RO-PANDA scheme, we can use another copy of the scheme of Construction 4.2, which will at most double the complexity.) Regarding the Read complexity, each execution of Read' requires performing $\mathsf{poly}(\log L_{\mathsf{max}})$ operations for each of the $\log L_{\mathsf{max}}$ levels (to read the counters etc.), but the client only needs to perform read operations *on the first* $\log L$ *levels* (all other levels are empty), so the client and server complexity during the Read protocol is $\mathsf{poly}(\log L_{\mathsf{max}}) + t \cdot \mathsf{poly}(\lambda, \log L)$. Similarly, the client and server complexities during Write are the same as in Construction 4.2 (since all operations need to be performed only on the first L levels), except that they also need to perform $O(1)$ operations on counters (of size at most $\log L_{\mathsf{max}}$), so the complexities are $O(\log L + \log L_{\mathsf{max}})$ and $\mathsf{poly}(\log L_{\mathsf{max}}) + t \cdot L^{\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$, respectively. Regarding server state and key sizes, the client holds $\log L_{\mathsf{max}}$ keys of the PE-PANDA scheme, so the key size is $t \cdot \mathsf{poly}(\lambda, \log L_{\mathsf{max}})$, and the server stores public keys for all $\log L_{\mathsf{max}}$ levels, but only the first $\log L$ levels are non-empty, so the server state has size $t \cdot \mathsf{poly}(\lambda, \log L_{\mathsf{max}}) + t \cdot L^{1+\epsilon} \cdot \mathsf{poly}(\lambda, \log L)$. We note that bucket overflows still happen with negligible probability since $L_{\mathsf{max}} = \mathsf{poly}(\lambda)$.

□

4.2 A Secret-Writes PANDA Scheme

In this section we focus on the the *private-database* setting, in which the database consists of the *private* data of different clients (and each client is allowed to access only its own data). In this setting, write privacy can be obtained for honest clients. Our goal is to design a PANDA scheme (supporting writes) in this setting, and for this we use a PW-PANDA scheme for dynamic databases (Section 4.1.1). We first formally define the primitive.

Definition 4.6 (Secret-Writes PANDA (SW-PANDA))**.** A *Secret-Writes Private Anonymous Data Access (SW-PANDA)* scheme consists of procedures $(\mathsf{Setup}, \mathsf{Read}, \mathsf{Write})$ with the syntax of Definition 4.1, and the following correctness and security properties.

- **Correctness:** In any execution of the Setup algorithm followed by a sequence of Read and Write protocols between various clients and the server, in which every client only accessed its own data, and where the Write protocols were executed with a sequence Q of values, the output of each client in a read operation is the value it would have read from the database if (the prefix of) Q (performed before the corresponding Read protocol) was performed directly on the database.

- **Security:** Any PPT adversary $\mathcal{A}$ has only $\mathsf{negl}(\lambda)$ advantage in the following security game with a challenger $\mathcal{C}$:
  - $\mathcal{A}$ sends to $\mathcal{C}$:
    - The values n, t, and a pair of databases $\mathsf{DB}^0, \mathsf{DB}^1 \in \{0,1\}^L$.
    - A subset $T \subset [n]$ of corrupted clients with $|T| \leq t$.
    - A pair of access sequences $Q^0 = \left(\mathsf{op}_l, \mathsf{val}_l^0, j_l^0, \mathsf{addr}_l^0\right)_{1 \leq l \leq q}, Q^1 = \left(\mathsf{op}_l, \mathsf{val}_l^1, j_l^1, \mathsf{addr}_l^1\right)_{1 \leq l \leq q}$, where $\left(\mathsf{op}_l, \mathsf{val}_l^b, j_l^b, \mathsf{addr}_l^b\right)$ denotes that client j_l^b performs operation op_l at address addr_l^b in database DB^b with value val_l^b (which, if $\mathsf{op}_l = \mathsf{read}$, is $\perp$).

    We require that $\left(\mathsf{op}_l, \mathsf{val}_l^0, j_l^0, \mathsf{addr}_l^0\right) = \left(\mathsf{op}_l, \mathsf{val}_l^1, j_l^1, \mathsf{addr}_l^1\right)$ for every $l \in [q]$ such that $j_l^0 \in T \vee j_l^1 \in T$ (in particular, Q^0, Q^1 differ only on accesses of honest clients), and that $\mathsf{DB}^0, \mathsf{DB}^1$ agree on all blocks that belong to clients in T.
  - $\mathcal{C}$ performs the following:
    - Picks a random bit $b \leftarrow \{0,1\}$.
    - Initializes the scheme by computing $\mathsf{Setup}\left(1^\lambda, 1^n, 1^t, \mathsf{DB}^b\right)$.
    - Sequentially executes the sequence Q^b of Read and Write protocol executions between the honest server and clients. It sends to $\mathcal{A}$ the views of the server S and the corrupted clients $\{C_j\}_{j \in T}$ during these protocol executions, where the view of each party consists of its internal state, randomness, and all protocol messages received.
  - $\mathcal{A}$ outputs a bit b'.

  The advantage $\mathsf{Adv}_{\mathcal{A}}(\lambda)$ of $\mathcal{A}$ in the security game is defined as: $\mathsf{Adv}_{\mathcal{A}}(\lambda) = |\Pr[b' = b] - \frac{1}{2}|$.

Notice that Definition 4.6 does not hide which operations were read and which were write operations. A stronger notion of security which also hides the type of operations performed, can be generically achieved as follows: in every (read or write) operation, first read the desired address, and then write to that address, either writing-back the read value (when performing a read), or writing the new desired value (when performing a write).

Construction Overview. Recall from the introduction that we use a PW-PANDA scheme for dynamic databases (Section 4.1.1), and the high-level idea is for write operations to add *new copies* of the data blocks to the PW-PANDA. To prevent the adversary from correlating different copies of the same data block (which would allow it to deduce the number of times a specific data block was written), we replace the actual address of the written value with an "effective" address which is obtained by applying a PRF (with a client-specific key) to the logical address *and the copy number*. Since this prevents the server from removing duplicates from the underlying PANDA, the server storage grows proportionally to the number of writes. Moreover, the PANDA scheme may now contain multiple copies of the same data block, so operations need to first determine the current copy number. For read operations, this is achieved by performing a binary search over all possible copy numbers. For write operations, this is obtained by first performing a read for the data block (which locates the most recent copy), and then setting the copy number of the new copy accordingly.

We now formally describe our construction. For simplicity of the exposition, we assume that all clients know the location of their blocks in DB.[16] We use a bound L_{max} (whose value is set below) on the maximal size of the database.

Construction 4.7 (Secret-Writes PANDA)**.** The scheme uses the following building blocks:

- A PW-PANDA scheme for dynamic databases $\left(\mathsf{Setup}^{\mathsf{PW}}, \mathsf{Read}^{\mathsf{PW}}, \mathsf{Write}^{\mathsf{PW}}\right)$.
- A symmetric encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with pseudorandom ciphertexts[17] (used to encrypt the contents of data blocks).
- A PRF F (used to generate the "effective" block addresses).

We define the following protocols.

- **$\mathsf{Setup}(1^\lambda, 1^n, 1^t, \mathsf{DB})$:** Recall that n denotes the number of clients, t is the collusion bound, and $\mathsf{DB} \in \{0,1\}^L$. It does the following.
  - Counter initialization. Initialize a write counter count_W to 0.[18]
  - Generating client keys. For every $1 \leq j \leq n$, generate a random PRF key $K_j \leftarrow \{0,1\}^\lambda$ for F, and an encryption key $K_j^{\mathsf{sym}} \leftarrow \mathsf{KeyGen}\left(1^\lambda\right)$, for client C_j.
  - Initializing the database.
    - $*$ For every data block $b_i, 1 \leq i \leq L$ which belongs to client C_j, compute $\widetilde{b}_i \leftarrow \mathsf{Enc}\left(K_j^{\mathsf{sym}}, b_i\right)$, and let $\widetilde{\mathsf{DB}} = \left(\left((1,1), \widetilde{b}_1\right), \ldots, \left((L,1), \widetilde{b}_L\right)\right)$. (Intuitively, we attach to each data block its logical address, and the copy number "1".)
    - $*$ Run $\mathsf{Setup}^{\mathsf{PW}}\left(1^\lambda, 1^n, 1^t, 1^L, L_{\mathsf{max}}, \widetilde{\mathsf{DB}}\right)$ to generate a public-writes PANDA scheme for dynamic databases for $\widetilde{\mathsf{DB}}$. $\mathsf{Setup}^{\mathsf{PW}}$ outputs a server state $\mathsf{st}_S^{\mathsf{PW}}$, and client keys $\mathsf{ck}_1^{\mathsf{PW}}, \ldots, \mathsf{ck}_n^{\mathsf{PW}}$.
  - Output. For each client C_j, set the client key $\mathsf{ck}_j = \left(K_j, K_j^{\mathsf{sym}}, \mathsf{ck}_j^{\mathsf{PW}}\right)$ to consist of its PRF and encryption keys, as well as its key in the PW-PANDA scheme. Set the server state $\mathsf{st}_S = \left(\mathsf{count}_W, \mathsf{st}_S^{\mathsf{PW}}\right)$ to consist of the counter, and the server state in the PW-PANDA scheme.
- **The Read protocol.** To read the database value at location $\mathsf{addr} \in [L]$ from the server S, a client C_j with key $\mathsf{ck}_j = \left(K_j, K_j^{\mathsf{sym}}, \mathsf{ck}_j^{\mathsf{PW}}\right)$ and the server S run the following protocol.
  - Obtaining counter information. C_j reads count_W from S. (Notice that there are at most count_W copies of the data block addr in the PW-PANDA.)
  - Locating the most recent copy in the PW-PANDA. C_j performs a binary search to find the most recent copy (i.e., the copy with largest index) of data block addr, in the range $1, \ldots, \mathsf{count}_W$ (if such a copy exists). The binary search is executed as follows: whenever

[16] This assumption can be removed by implementing DB with a structure similar to the levels in Construction 4.2, namely using buckets to which data blocks are mapped using a public hash.

[17] A symmetric encryption scheme with pseudorandom ciphertexts is a symmetric encryption scheme in which the ciphertexts generated by Enc are pseudorandom strings (see, e.g., [ABSV15]).

[18] We note that when the underlying PW-PANDA is the scheme obtained from Construction 4.2, then one can use the write counter of the underlying scheme. However, we cannot generally assume that the underlying scheme keeps such a counter.

the algorithm requires reading (i.e., searching for) copy m from the PANDA, the client and server run $\mathsf{Read}^{\mathsf{PW}}$ for address $\mathsf{addr}_m = F(K_j, (\mathsf{addr}, m))$ in the PW-PANDA. Let $\left((\mathsf{addr}, m), \widetilde{\mathsf{val}}\right)$ denote the output of the binary search.

   – Recovering data. C_j decrypts $\mathsf{val} = \mathsf{Dec}\left(K_j^{\mathsf{sym}}, \widetilde{\mathsf{val}}\right)$.

- **The Write protocol.** To write value val at location $\mathsf{addr} \in [L]$ on the server S, a client C_j with key $\mathsf{ck}_j = \left(K_j, K_j^{\mathsf{sym}}, \mathsf{ck}_j^{\mathsf{PW}}\right)$ and the server S run the following protocol.

   – Locating the most recent copy. The client and server execute the Read protocol for data block addr. (Notice that from this execution, C_j learns the index m of the most recent copy of block addr.)

   – Adding a new block copy. C_j encrypts $\widetilde{\mathsf{val}} \leftarrow \mathsf{Enc}\left(K_j^{\mathsf{sym}}, \mathsf{val}\right)$, and computes the *effective* address of the new copy as $\mathsf{addr}' = F(K_j, (\mathsf{addr}, m+1))$. Then, the client and server run $\mathsf{Write}^{\mathsf{PW}}$ to add the block at address addr' with value $\widetilde{\mathsf{val}}$.

   – The server S updates its state by setting $\mathsf{count}_W := \mathsf{count}_W + 1$.

We prove the following claim about Construction 4.7.

Claim 4.8. *Assuming the security of all the building blocks, Construction 4.7 is a secure SW-PANDA scheme.*

Proof. The correctness of the scheme follows directly from the correctness of the underlying building blocks. (In particular, when the range of F is sufficiently large, then except with negligible probability no two effective addresses in the execution are the same, and so no client overwrites another client's data blocks.) We now argue security.

Let $\mathcal{A}$ be a PPT adversary corrupting the server and a subset T of at most t clients. Let $\mathsf{DB}^0, \mathsf{DB}^1$ be any pair of databases that agree on all blocks that belong to clients in T, and let $Q^0 = \left(\mathsf{op}_l, \mathsf{val}_l^0, j_l^0, \mathsf{addr}_l^0\right)_{1\le l\le q}, Q^1 = \left(\mathsf{op}_l, \mathsf{val}_l^1, j_l^1, \mathsf{addr}_l^1\right)_{1\le l\le q}$ denote the sequences of accesses which $\mathcal{A}$ chose in the game. We proceed through a sequence of hybrids.

H_1^b: H_1^b is the security game of Definition 4.6 with bit b.

H_2^b: In H_2^b, the value of all data blocks of honest clients (in DB^b or in write operations) are replaced with encryptions of 0.

$H_1^b \approx H_2^b$ by the security of the encryption scheme. (Notice that the pseudorandom ciphertexts of the encryption scheme guarantee indistinguishability even if a client owns multiple data blocks, or writes the same value to a data block multiple times.)

H_3^b: In H_3^b, we replace the effective addresses computed (during the run of the Read or Write protocols) during an access of an honest client with the output of a random function (instead of the outputs of F).

$H_2^b \approx H_3^b$ by the security of the PRF.

To finish the proof, we show that that the views of $\mathcal{A}$ in H_3^0, H_3^1 are computationally indistinguishable (and so its advantage in the security game is negligible). This follows from the security of the PW-PANDA scheme. Indeed, in both cases write operations of honest clients write encryptions of 0 to *random* addresses (since the address is the output of a random function on a *unique* value, due to the use of increasing copy numbers). Moreover, write operations of different honest clients are indistinguishable due to the pseudorandom ciphertexts of the encryption scheme (which guarantees that given two ciphertexts, one cannot determine whether they were generated with the same encryption key, or with different encryption keys, nor learn anything about the key; thus one learns nothing about the identity of the client who performed the operation). Furthermore, in both views the database contains encryptions of the same values for data blocks of corrupted clients (since these were initialized with the same values, and all operations of corrupted clients are the same in Q^0, Q^1), and encryptions of 0 in the data blocks of honest clients. Finally, the read operations of corrupted clients are the same in both views (by the restrictions of Definition 4.6). Consequently, H_0^3, H_1^3 differ only in read accesses of honest clients to the PW-PANDA, and the identity of the honest clients performing write operations on the PW-PANDA, and so H_0^3, H_1^3 are computationally indistinguishable by the security of the PW-PANDA scheme.

□

We now use Claim 4.8 to prove the following theorem.

Theorem 4.9 (Secret-Writes PANDA scheme)**.** *Suppose leveled FHE exists. Then for any constant $\varepsilon > 0$ there is a SW-PANDA, for n clients, t collusion bound and database size L, where after W write operations:*

- *The client and server complexity during each Read protocol is $t \cdot \mathsf{poly}(\lambda, \log L, \log W)$.*
- *The client complexity during each Write protocol is $t \cdot \mathsf{poly}(\lambda, \log L, \log W)$, and the amortized server complexity is $t \cdot (L + W)^{\epsilon} \cdot \mathsf{poly}(\lambda, \log L, \log W)$.*[19]
- *The client keys are of size $t \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The server state is $t \cdot (L + W)^{1+\epsilon} \cdot \mathsf{poly}(\lambda, \log L, \log W)$.*

Proof. We instantiate Construction 4.7 with the PW-PANDA scheme of Theorem 4.5 for dynamic databases of size at most $L_{\mathsf{max}} = 2^{\lambda}$, any PRF with $\mathsf{poly}(\lambda)$ overhead (and sufficiently large range), and any secure symmetric encryption scheme with pseudorandom ciphertexts (see, e.g., [Gol04]) with $\mathsf{poly}(\lambda)$ overhead. Then correctness and security follow directly from Claim 4.8.

Regarding the complexity of the scheme, every execution of the Read protocol consists of a binary search in the range $1, \ldots, W$, where each read of the binary search calls $\mathsf{Read}^{\mathsf{PW}}$ once on a database of size $L + W$ (so this step costs $t \cdot \mathsf{poly}(\lambda, \log L, \log W)$); and additionally requires computing the PRF once and decrypting one value (both take $\mathsf{poly}(\lambda)$ time), so the client and server complexity during the Read protocol is $t \cdot \mathsf{poly}(\lambda, \log L, \log W)$. Every execution of the Write protocol consists of a single execution of the Read protocol ($t \cdot \mathsf{poly}(\lambda, \log L, \log W)$ time); computing one PRF image and one encryption ($\mathsf{poly}(\lambda)$ time); and a single execution of the $\mathsf{Write}^{\mathsf{PW}}$ protocol ($\mathsf{poly}(\lambda)$ time for the client, $t \cdot (L + W)^{\epsilon} \cdot \mathsf{poly}(\lambda, \log L, \log W)$ amortized time for the server), so the client (server) complexity during the Write protocol is $t \cdot \mathsf{poly}(\lambda, \log L, \log W)$ ($t \cdot (L + W)^{\epsilon} \cdot \mathsf{poly}(\lambda, \log L, \log W)$). Regarding the client keys, they consist of a PW-PANDA key, and a PRF and encryption key, so the client key has size $t \cdot \mathsf{poly}(\lambda, \log L)$. As for the server state, the server stores a single PW-PANDA scheme for dynamic databases of size at most $L_{\mathsf{max}} = 2^{\lambda}$, whose actual size is $L + W$, and a counter (which requires at most $O(\log W)$ bits), so the server storage is $t \cdot (L + W)^{1+\epsilon} \cdot \mathsf{poly}(\lambda, \log L, \log W)$. □

[19]The server complexity can be de-amortized similarly to Theorem 4.4.

5 Extensions and Additional Results

5.1 Adaptive Security

The adversarial model in the various PANDA notions considered in previous sections is selective, where the adversary chooses the access pattern at the onset of the game. One could also consider a stronger adversarial model in which the adversary chooses the access pattern *adaptively* during the game. (We chose to define the weaker selective notion since it simplifies the definitions.) All our constructions are also secure against adaptive adversaries. Concretely, for bounded RO-PANDA (Construction 3.4), essentially the same proof as that of Claim 3.5 also shows adaptive security. The notion of PE-PANDA is defined with an adaptive adversary (Definition 3.8). Finally, (unbounded) RO-PANDA, PW-PANDA and SW-PANDA are adaptively secure since they are constructed using PE-PANDA, which is adaptively secure.

5.2 Maliciously-Corrupted Parties

In previous sections we have focused on constructing PANDA schemes that are secure against semi-honest coalitions consisting of the server and a (small) fraction of the clients, all of whom follow the protocol honestly but try to learn sensitive information from their view of the protocol execution. We now generalize our constructions to the setting of *malicious* corruptions, in which parties may arbitrarily deviate from the protocol.

5.2.1 Read-Only PANDA

We begin by considering the case of unbounded-access read-only PANDA from Section 3.3.

Maliciously-corrupted clients. We first consider the setting in which the server remains *semi-honest* but colludes with *maliciously-corrupted* clients. In particular, the server is still guaranteed to follow the protocol, but clients may arbitrarily deviate from it. We note that our scheme remains secure in this setting. Indeed, in our scheme, the actions of the clients do not have any effect on the server state. Therefore, malicious behavior by the corrupted clients has no impact on the protocol executions of the honest clients, or the view of the adversary. In other words, in our scheme there is no real difference between semi-honest and fully malicious clients.

Maliciously-corrupted server. Next, we consider the setting in which the server is also *malicious* (i.e., can arbitrarily deviate from the protocol), and may collude with *malicious* clients. There are only two possible points during the protocol in which the server may deviate from the specification and we discuss these separately.

Firstly, the server may lie about the epoch number, causing honest clients to perform too many accesses in a single epoch. This would violate the security requirements of the underlying bounded-access PANDA scheme, and would indeed lead to a loss of security. Recall that in our scheme the clients simply ask the server for the epoch number. Since (honest) clients do not communicate with each other, they have no way of knowing how many accesses were performed so far by other honest clients, and therefore cannot easily verify the current epoch number on their own. However, we may be able to assume some independent mechanism through which the number of accesses performed so far, and therefore also the current epoch number, could be known. For example, we may assume that some third party service can implement a "public counter" which clients can anonymously increment after each access, and anybody can read to see how many accesses were performed so far.[20] Alternately, we may assume that we know the rate at which accesses occur and set the epochs to occur at regularly timed intervals, so that clients with synchronized clocks can individually determine the current epoch number on their own. In such settings we do not need to trust the server to provide the correct epoch number, and therefore resolve this problem.

Secondly, the server might send wrong values in the encoded database in response to client queries. Not only could this cause the clients to recover incorrect data but, by seeing whether clients succeed or fail in recovering the data, it also allows the server to perform Chosen-Ciphertext Attacks (CCA) against the FHE scheme, and potentially break the privacy or anonymity of the PANDA scheme. To solve this problem, whenever the client sends a query $(q_1, \ldots, q_l)$, we can have the server send the corresponding positions of the encoded database $a_i = \widetilde{\mathsf{DB}}_{q_i}$, and also *prove* that these values were computed correctly using a succinct interactive argument. One subtlety is that, even if we use succinct arguments which are efficient for the client (who plays the role of the verifier), we also need these arguments to be equally efficient for the server (who plays the role of the prover). It turns out that we can make this work as described below.

We first augment the Setup procedure of the unbounded-access PANDA from Construction 3.13 to have each client store a collision-resistant hash $\sigma = H(\mathsf{DB})$ of the database DB as well as the public key pk of the underlying public-encoding PANDA. During the Read protocol, the client sends queries $(q_1, \ldots, q_l)$, and the server sends the corresponding positions $a_i = \widetilde{\mathsf{DB}}_{q_i}$ of the encoded database. In addition, the server sends $\tilde{\sigma} = MH(\widetilde{\mathsf{DB}})$, where MH denotes a Merkle-Tree Hash. It then gives the path in the Merkle-Tree to succinctly show that $a_i = \widetilde{\mathsf{DB}}_{q_i}$ is indeed the value in position q_i of the hashed $\widetilde{\mathsf{DB}}$. Finally the server uses a succinct interactive argument of knowledge to prove that it knows some DB such that $\sigma = H(\mathsf{DB})$ and $\tilde{\sigma} = MH(\mathsf{Encode}(\mathsf{pk}, \mathsf{DB}, \mathsf{count}_e))$, where count_e is the current epoch number. Since the adversary cannot break the collision resistance of H, MH, this implies that a_i must be the correctly computed value. Note that the statement and witness in the interactive argument do not depend on a_i, and are therefore the same for all protocol executions throughout the epoch. For efficiency, we need a succinct interactive argument of knowledge where, for a statement x in an NP-relation that can be verified in time T:

1. The complexity of the verifier is $\mathsf{poly}(\lambda, |x|, \log T)$.
2. The prover can perform a one-time pre-processing in time $T \cdot \mathsf{poly}(\lambda, |x|, \log T)$, after which it can run arbitrarily many executions of the interactive argument with a verifier, where the complexity of each execution is $\mathsf{poly}(\lambda, |x|, \log T)$.

Using such an argument system, the server would perform the pre-processing step of the argument system at the beginning of each epoch, as well as compute the Merkle-Tree corresponding to $\tilde{\sigma} = H(\widetilde{\mathsf{DB}})$. It would then execute proofs efficiently during each subsequent read access. It turns out that such interactive proofs can be constructed using the paradigm of Kilian [Kil92] combined with quasi-linear probabilistically checkable proofs (PCP), e.g., [BCGT13]. The pre-processing would compute a PCP for the computation $\widetilde{\mathsf{DB}} = \mathsf{Encode}(\mathsf{pk}, \mathsf{DB}, \mathsf{count}_e), \sigma = H(\mathsf{DB}), \tilde{\sigma} = MH(\widetilde{\mathsf{DB}})$, and store a Merkle-Tree Hash of this PCP. Since the approach of Kilian only relies on collision-resistant hashing which is implied by FHE, this does not introduce any additional assumptions, nor does it increase the asymptotic efficiency of our read-only solution.

[20] Note that there are no privacy requirements for this counter, and the only security requirement is that an adversary cannot decrement it. The adversary could always artificially increment the count, but this does not violate security.

5.2.2 PANDA with Writes

We now consider malicious security for PANDA schemes with public and secret writes. In both cases, we only consider the case of a semi-honest server colluding with fully malicious clients, and we leave it as an open problem to consider a fully malicious server as well.

Public-Writes PANDA. In a public-writes PANDA, the Write protocol solely consists of clients sending to the server, in the clear, the address and value being written. Therefore, there is no difference between a fully malicious client, and a semi-honest client who follows the protocol with some sequence of writes. Consequently, malicious client behavior does not affect the privacy or anonymity of the honest clients. Note, however, that malicious clients can completely overwrite all of the data in the database with arbitrary values! (Indeed, we want to allow clients to write arbitrarily, so an honest client can do that as well.) Whether clients should be allowed to write arbitrarily depends on the specific application scenario, but it is easy to imagine that this would not be desirable. We can easily add restrictions to the allowed write operations by having the server enforce them. For example, the server can only allow certain locations of the database to be writable, while others are read-only; since the server sees the locations being written to, it can simply ignore write requests to illegal locations.

Secret-Writes PANDA. In a secret-writes PANDA, each client has an individual database and we certainly do not want malicious clients to be able to overwrite the data of honest clients. Recall that in our construction, the Write protocol consists of the clients giving an (address, value) pair, but this time the address consists of a PRF applied to the actual location being written and a counter. There are two potential issues if malicious clients choose the addresses adversarially.

Firstly, a malicious client could potentially choose an address that "belongs" to some honest client. To solve this, the server can simply check that every address ever being written is fresh (which for honest clients will anyway be the case with overwhelming probability due to the pseudorandomness of the PRF).

Secondly, a malicious client colluding with a semi-honest server could potentially choose addresses in an adversarial way to cause an overflow in some bucket.[21] To solve this problem, we require that each time a level is initialized, the server picks completely fresh hash functions (by sampling randomness on the fly) that map addresses to buckets. That way, even if malicious clients collude with the server and maliciously select the address, they will not know which hash function will be used to map this address to buckets later on. We now need to modify the Read procedure slightly so that the server also gives the client the description of the hash functions used at every level at the start of every Read protocol.

5.3 Alternative Solutions Without Anonymity

When anonymity is not required, our techniques can be used to obtain simpler alternative constructions of multi-client read-only private database access schemes (without client anonymity). In this setting, we can get a solution based only on one-way functions (OWFs) without relying on fully homomorphic encryption (FHE).

A trivial solution in this case would be for each client to have its own ORAM stored on the server (for which only the client knows the secret key), where to access the database the client only needs to access her own ORAM scheme. (Note that this would reveal the identity of the client, but

[21]This can also occur in the public-writes setting, but in that case clients can anyway overwrite all the data so it does not yield any additional attack.

is not problematic when anonymity is not required.) Therefore, in the trivial solution the server storage is linear in the number of clients, even though all clients access one shared, public, read-only database.

Our techniques can be used to reduce server storage, making it sublinear in the number of clients. The high-level idea is to store an LDC-encoding of the database at multiple virtual servers, as in our bounded-access RO-PANDA scheme (Construction 3.4), but to implement each virtual server using a standard ORAM scheme (instead of permuting the codeword using PRPs). As in Construction 3.4, we will use an LDC with data-size L, smoothness $s = \lambda$ and corresponding locality $k = \mathsf{poly}(\lambda, \log L)$, and codeword size $M = L^{1+\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$. We will have $k' = k^2 t$ different virtual servers, where each one consists of a separate ORAM containing the LDC codeword. Each client will have a random committee consisting of k out of k' virtual servers associated with her, for which she will know the secret ORAM keys. To read from the database, the client generates the queries of the LDC decoder (as in Construction 3.4), and reads these locations from the ORAMs of the virtual servers on her committee (each ORAM is used to answer a single decoder query). Even if the adversary corrupts t of the clients, and therefore implicitly corrupts all of the virtual servers on their committees, with overwhelming probability every honest client's committee will have at most s corrupted virtual servers on it. Therefore, every honest client's read remains private since the adversary only sees the queries to the corrupted virtual server (by the security of the ORAM) and these are uniformly random and independent of the location being read (by the smoothness of the LDC).

The above solutions gives us a multi-client "private *non-anonymous* data access scheme" in the read-only setting using only OWFs. Using an ORAM with poly-logarithmic overhead based on OWFs (e.g., [GO96, SvDS+13]), for any $\varepsilon > 0$ we can get a scheme where for n clients, with database size L, and collusion bound t:

- The client/server run-time for each read operation is $t \cdot \mathsf{poly}(\lambda, \log L)$.
- The client storage is $t \cdot \mathsf{poly}(\lambda, \log L)$.
- For any $\varepsilon > 0$, the server storage is $t \cdot L^{1+\varepsilon} \cdot \mathsf{poly}(\lambda, \log L)$.

5.4 Non-Client Writes

Our public-writes PANDA scheme (Construction 4.2) has the property that anyone can write to the database even if they're not one of the designated clients in the scheme. Still, only the designated set of clients can privately read from the database. This is because writing to the PANDA scheme requires no secret key, whereas reading from it requires knowing a secret key that was generated (for a particular client) during the setup phase. Therefore, our public-writes PANDA scheme can be used to implement a public bulletin board to which everyone can write, but from which only a designated group of clients can read. This can be useful in several scenarios, e.g., when a company wishes to implement an anonymous "suggestion box" to which everyone in the company can add suggestions, but only the management can read.

Acknowledgments

We thank Mary Wootters for helpful conversations regarding the efficiency of encoding/decoding for RM codes, and for pointing out the work of [KU08]. The last author would also like to acknowledge Mariana Raykova for introducing him to the problem of multi-client ORAM.

Research supported in part by NSF grants 1619348, CNS-1413964, CNS-1413964, and CNS-1314722. This work was support in part by DARPA, US-Israel BSF grant 2012366, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. The views expressed are those of the authors and do not reflect position of the Department of Defense or the U.S. Government. The third author was supported in part by The Eric and Wendy Schmidt Postdoctoral Grant for Women in Mathematical and Computing Sciences.

References

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In *CRYPTO'15*, pages 657–677, 2015.

[BCGT13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete efficiency of probabilistically-checkable proofs. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 585–594. ACM, 2013.

[BG12] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 263–280, 2012.

[BHKP16] Michael Backes, Amir Herzberg, Aniket Kate, and Ivan Pryvalov. Anonymous RAM. In *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I*, pages 344–362, 2016.

[BIM00] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. In *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, pages 55–73, 2000.

[BIPW17] Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? *IACR Cryptology ePrint Archive (to appear in TCC'17)*, 2017:567, 2017.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995*, pages 41–50, 1995.

[Cha03] David Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. In *Secure Electronic Voting*, pages 211–219. 2003.

[CHR17] Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient private information retrieval. *IACR Cryptology ePrint Archive (to appear in TCC'17)*, 2017:568, 2017.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 303–320, 2004.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, 1996.

[Gol87] Oded Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *STOC'87*, pages 182–194, 1987.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques.* Cambridge University Press, 2001.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications.* Cambridge University Press, 2004.

[HO08] Brett Hemenway and Rafail Ostrovsky. Public-key locally-decodable codes. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, pages 126–143, 2008.

[HOSW11] Brett Hemenway, Rafail Ostrovsky, Martin J. Strauss, and Mary Wootters. Public key locally decodable codes with short keys. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 14th International Workshop, APPROX 2011, and 15th International Workshop, RANDOM 2011, Princeton, NJ, USA, August 17-19, 2011. Proceedings*, pages 605–615, 2011.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 723–732. ACM, 1992.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 364–373, 1997.

[KPK16] Nikolaos P. Karvelas, Andreas Peter, and Stefan Katzenbeisser. Blurry-ORAM: A multi-client oblivious storage architecture. *IACR Cryptology ePrint Archive*, 2016:1077, 2016.

[KT00] Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 80–86, 2000.

[KU08] Kiran S. Kedlaya and Christopher Umans. Fast modular composition in any characteristic. In *FOCS'08*, pages 146–155, 2008.

[LPDH17] Hemi Leibowitz, Ania M. Piotrowska, George Danezis, and Amir Herzberg. No right to remain silent: Isolating malicious mixes. *IACR Cryptology ePrint Archive*, 2017:1000, 2017.

[MBN15] Travis Mayberry, Erik-Oliver Blass, and Guevara Noubir. Multi-user oblivious RAM secure against malicious servers. *IACR Cryptology ePrint Archive*, 2015:121, 2015.

[MMRS15] Matteo Maffei, Giulio Malavolta, Manuel Reinert, and Dominique Schröder. Privacy and access control for outsourced personal records. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 341–358, 2015.

[Mul54] David E. Muller. Application of boolean algebra to switching circuit design and to error detection. *Trans. I.R.E. Prof. Group on Electronic Computers*, 3(3):6–12, 1954.

[OS97] Rafail Ostrovsky and Victor Shoup. Private information storage (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 294–303, 1997.

[Ost90] Rafail Ostrovsky. Efficient computation on oblivious RAMs. In *STOC'90*, pages 514–523, 1990.

[RAD78] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[Ree54] Irving S. Reed. A class of multiple-error-correcting codes and the decoding scheme. *Trans. of the IRE Professional Group on Information Theory (TIT)*, 4:38–49, 1954.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009.

[SvDS$^+$13] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 299–310, 2013.

[WY05] David P. Woodruff and Sergey Yekhanin. A geometric approach to information-theoretic private information retrieval. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*, pages 275–284, 2005.

[ZZQ16] Jinsheng Zhang, Wensheng Zhang, and Daji Qiao. MU-ORAM: dealing with stealthy privacy attacks in multi-user data outsourcing services. *IACR Cryptology ePrint Archive*, 2016:73, 2016.

A The Complexity of Reed-Muller Encoding

We show that Reed-Muller (RM) codes [Ree54, Mul54] can be encoded in quasilinear time (in the codeword length, up to a $\mathsf{poly}(\lambda)$ blowup). The analysis follows that used by Kedlaya and Umans [KU08] to analyze the complexity of (the more complicated case of) multipoint evaluation of multivariate polynomials in arbitrary characteristics. (More specifically, we use the second step in the algorithm used to prove [KU08, Theorem 3.1].)

Recall from Section 2 that in RM codes we choose a subset $H \subseteq \mathbb{F}$, the message DB is interpreted as an m-variate function $\mathsf{DB} : H^m \to \Sigma$, and the codeword is the evaluation, on all $\mathbb{F}^m$, of the low degree extension $\widetilde{\mathsf{DB}} : \mathbb{F}^m \to \mathbb{F}$ of DB of individual degree$< |H|$. Therefore, encoding a message DB requires: (1) interpolating the low-degree extension from its values on H^m; and (2) evaluating the low-degree extension on all points in $\mathbb{F}^m$.

First, we claim that if $\mathbb{F}$ has prime order $q \in \mathbb{N}$, and H is chosen to be the set of $|H|$'th roots of unity in $\mathbb{F}$, then interpolation can be done in time $\widetilde{O}\left(|\mathbb{F}|^m\right) = \widetilde{O}\left(\left|\widetilde{\mathsf{DB}}\right|\right)$. We prove this by induction on m. For the base case, the univariate polynomial (of degree$< q$) can be evaluated in time $q\mathsf{polylog}\left(q\right)$ using Fast Fourier Transform (FFT). For the step, assume the claim holds for $(m-1)$-variate polynomials, and given an m-variate polynomial $P\left(x_1, \ldots, x_m\right)$ (of individual degree$< q$), we write it as $P\left(x_1, \ldots, x_m\right) = \sum_{i=0}^{q-1} x_1^i \cdot P_i\left(x_2, \ldots, x_m\right)$ (if P has degree lower than $q-1$ in x, then the leading coefficients are set to 0). The induction hypothesis guarantees that the coefficients of each of $P_0, \ldots, P_{q-1}$ can be computed in time $q^{m-1}\mathsf{polylog}\left(q\right)$, and these give all the coefficients of P, so overall the coefficients of P are computable in time $q^m\mathsf{polylog}\left(q\right)$.

Second, we claim that if $\mathbb{F}$ has prime order $q \in \mathbb{N}$ then evaluating an m-variate polynomial (of individual degree$< q$) on all $\mathbb{F}^m$ can be done in time $m \cdot q^m \cdot \mathsf{polylog}\left(q\right)$. Again we prove the claim by induction on m. The base case with $m = 1$ follows from the fact that univariate multipoint evaluation can be done in time $q\mathsf{polylog}\left(q\right)$ using FFT. For the step, assume the claim holds for $(m-1)$-variate polynomials, and again we write the polynomial P as $P\left(x_1, \ldots, x_m\right) = \sum_{i=0}^{q-1} x_1^i \cdot P_i\left(x_2, \ldots, x_m\right)$. For every $\beta \in \mathbb{F}^{m-1}$, $P_\beta\left(x\right) = \sum_{i=0}^{q-1} x_1^i \cdot P_i\left(\beta\right)$ is a univariate polynomial and so using FFT can be evaluated on all $\mathbb{F}$ in time $q\mathsf{polylog}\left(q\right)$. Using the induction hypothesis, each P_i can be evaluated on all $\beta \in \mathbb{F}^{m-1}$ in time $(m-1) \cdot q^{m-1} \cdot \mathsf{polylog}\left(q\right)$. Consequently, one can evaluated P on all $\mathbb{F}^m$ in time

$$q \cdot (m-1) \cdot q^{m-1} \cdot \mathsf{polylog}\left(q\right) + q\mathsf{polylog}\left(q\right) \cdot q^{m-1} = m \cdot q^m \cdot \mathsf{polylog}\left(q\right).$$

In summary, overall the RM encoding time is $m \cdot q^m\mathsf{polylog}\left(q\right) = \widetilde{O}(M)$.