Entangled Cloud Storage

Giuseppe Ateniese[1] Özgür Dagdelen[2] Ivan Damgård[3] Daniele Venturi[3]

[1] Sapienza University of Rome, Italy
ateniese@di.uniroma1.it
[2] Technische Universität Darmstadt, Germany
oezguer.dagdelen@cased.de
[3] Aarhus University, Denmark
{ivan,dventuri}@cs.au.dk

Abstract. Entangled cloud storage enables a set of clients $\{P_i\}$ to "entangle" their files $\{f_i\}$ into a single *clew* c to be stored by a (potentially malicious) cloud provider S. The entanglement makes it impossible to modify or delete significant part of the clew without affecting *all* files in c. A clew keeps the files in it private but still lets each client P_i recover his own data by interacting with S; no cooperation from other clients is needed. At the same time, the cloud provider is discouraged from altering or overwriting any significant part of c as this will imply that none of the clients can recover their files.

We provide theoretical foundations for entangled cloud storage, introducing the notion of an *entangled encoding scheme* that guarantees strong security requirements capturing the properties above. We also give a concrete construction based on privacy-preserving polynomial interpolation, along with protocols for using the encoding scheme in practice.

Protocols for cloud storage find application in the cloud setting, where clients store their files on a remote server and need to be ensured that the cloud provider will not delete their data illegitimately. Current solutions, e.g., based on Provable Data Possession and Proof of Retrievability, catch a malicious server "after-the-fact", meaning that the server needs to be challenged regularly to provide evidence that the clients' files are stored *at a given time*.

Entangled storage makes all clients equal and with the same rights: It makes it financially inconvenient for a cloud provider to alter specific files and exclude certain "average" customers, since doing so would undermine all customers in the system, even those considered "important" and, thus, profitable. Therefore, entangled storage schemes offer security "before-the-fact".

1 Introduction

The terminology "cloud computing" refers to a paradigm shift in which applications from a server are executed and managed through a client's web browser, with no installed client version of an application required. This new paradigm—also known as the software as a service paradigm—has generated new intriguing challenges for cryptographers. In this paper we deal with the problem of *cloud storage*, where clients store their files on remote servers. Outsourcing data storage provides several benefits, including improved scalability and accessibility, data replication and backup, and considerable cost saving.

Nevertheless, companies and organizations are still reluctant to outsource their storage needs. Files may contain sensitive information and cloud providers can misbehave. While encryption can help in this case, it is utterly powerless to prevent data corruption, whether intentional or caused by a malfunction. Indeed, it is reasonable to pose the following questions: How can we be certain the cloud provider is storing the entire file intact? What if files that are rarely accessed are altered? Can we detect these changes and possibly recover the original content?

Possible solutions. It turns out that the questions above have been studied extensively in the last few years. Proof-of-storage schemes allow clients to verify that their remote files are still pristine even though they do not possess any local copy of these files. Two basic approaches have emerged: Provable Data Possession (PDP), introduced by Ateniese *et al.* [2], and Proof of Retrievability (PoR), independently introduced by Juels and Kaliski [25] (building on a prior work by Naor and Rothblum [34]). They were later extended in several ways in [39, 15, 3]. In a PDP scheme, file blocks are signed by the clients via authentication tags. During an audit, the remote server is challenged and proves possession of randomly picked file blocks by returning a short proof of possession. The key point is that the response from the server is essentially constant, thanks to the homomorphic property of authentication tags that makes them *compressible* to fit into a short string. Any data alteration or deletion will be detected with high probability. In POR, in addition, error correction codes are included along with remote file blocks. Now, the server provides a proof that the entire file could potentially be recovered in case of hitches.

However, proof-of-storage schemes catch a misbehaving cloud provider after the fact, when targeted files have already been altered. In this paper we consider a different approach that is based on making altering or deleting files extremely inconvenient for the cloud provider. The idea is to have the clients encode all their files into a single digital *clew* c, an "entangled encoding", that can be used as a representation of all files and be stored on remote and untrusted servers. The goal is to ensure that any significant change to c is likely to disrupt the content of all files. This approach of *data entanglement* was proposed by Aspnes *et al.* [1]. Roughly speaking, an entangled encoding should satisfy the following properties: (i) every client which took part to the entanglement generation process should be able to recover its original file from c; (ii) if the server alters c in any way, no clients will be able to retrieve its original file (this requirement is called all-or-nothing-integrity).

Unfortunately, in the original model of Aspnes *et al.* [1], the entanglement is created by a trusted authority. In addition, files can only be retrieved through the trusted authority. Thus, schemes within their framework are not suitable for cloud computing. In this paper we focus on storage schemes where the entanglement is collectively created by all clients and files can also be retrieved without interacting with any trusted entity. We will refer to our framework as *entangled storage*.

In a sense, entangled storage makes all clients equal and with the same rights: It is financially inconvenient for a cloud provider to alter specific files or target and exclude certain "average" customers since doing so would undermine all customers in the system, even those considered "important" and, thus, profitable.

1.1 Our Contributions

The main contribution of this paper is to provide theoretical foundations for data entanglement. More in detail, our contributions and techniques are outlined below.

Simulation-based security. Our definition of entangled storage uses a strong simulation-based definition, capturing all possible security concerns. In the simulation-based paradigm, security of a cryptographic protocol is defined by comparing an execution in the real world, where the scheme is deployed, with an execution in an ideal world, where all the clients give their inputs to a trusted party which then computes the output for them.[1]

[1] Essentially, with regard to privacy, the model of [1] can be viewed as the ideal world in our security definitions.

Roughly, the ideal functionality $\mathcal{I}_{\mathsf{ESS}}$ that we introduce captures the following security requirements (see also the discussion in Section 4): (1) *Privacy of entanglement:* The entanglement process does not leak information on the file f_i of client P_i, neither to other (possibly malicious) clients nor to the (possibly malicious) server; (2) *Privacy of recovery:* At the end of each recovery procedure, the confidentiality of all files is still preserved; (3) *All-or-nothing integrity:* A malicious server overwriting a significant part of the entanglement is not able to answer recovery queries from any of the clients.

An abstract framework to realize $\mathcal{I}_{\mathsf{ESS}}$. As a stepping stone towards the construction of entangled storage, we introduce the notion of an *entangled encoding scheme*, which we believe it is of independent interest. Informally, such an encoding Encode takes as input n strings $f_1, \ldots, f_n$ (together with a certain amount of randomness $r_1, \ldots, r_n$) and outputs a single codeword c which "entangles" all the input strings. The encoding is efficiently decodable, i.e. there exists an efficient algorithm Decode that takes as input (c, r_i, i) and outputs f_i. Since only r_i is required to retrieve f_i (we don't need $r_j, j \neq i$) we refer to this as "local decodability". In addition, the encoding satisfies two main security properties. First off, it is *private* in the sense that even if an adversary already knows a subset of the input strings and randomness used to encode them, the resulting encoding reveals no additional information about any of the other input strings other than what can be derived by the knowledge of this subset. Second, it is *all-or-nothing* in the sense that whenever an adversary has "large" uncertainty about c (i.e., a number of bits linear in the security parameter), he cannot design a function that will answer any decoding query correctly. See Section 3 for a precise definition.

Now, given an entangled encoding scheme $(\mathsf{Encode}, \mathsf{Decode})$, we describe a general framework to realize entangled storage in the cloud, with the goal of implementing the ideal functionality $\mathcal{I}_{\mathsf{ESS}}$. Consider n (possibly malicious) clients $P_1, \ldots, P_n$ each holding a file f_i and a (possibly malicious) server S. During the so-called "entanglement phase", the clients interact in a multi-party computation protocol Π_{ETG} together with the server. Each client inputs file f_i and learns nothing at the end of the execution; on the other hand the server inputs nothing and outputs $\mathsf{Encode}(f_1, \ldots, f_n)$. Later on, in the so-called "recovery phase" a client P_i can run a two-party protocol Π_{RCV} together with S. The client inputs some trapdoor (whose length is constant and independent of the length of f_i) and outputs file f_i; on the other hand the server inputs the entanglement and outputs nothing.

Technical difficulties in proving security. With such a framework in hand, we would like to prove that this securely realizes functionality $\mathcal{I}_{\mathsf{ESS}}$ in the presence of malicious adversaries, whenever the underlying protocols Π_{ETG} and Π_{RCV} are secure against malicious adversaries. The intuition for this would be that a simulator could simply encode random data for the honest clients (which is fine by privacy of the underlying entangled encoding). Then, if a corrupt S overwrites a large part of its storage, the simulator should tell the functionality $\mathcal{I}_{\mathsf{ESS}}$ that the server has misbehaved. The functionality will answer any recovery query with "fail" after this point, but this is fine: by the all-or-nothing integrity property the server cannot answer any queries in the real protocol either.

However, turning this intuition into a proof turns out to be a challenging task. The all-or-nothing-integrity property does guarantee a very useful information theoretic statement: Whenever there is enough uncertainty about the encoding c, it is impossible to answer any recovery query with good probability. In the protocol execution, however, we need efficient simulation and this means that the simulator must be able to efficiently decide when a corrupt server is in a state where its uncertainty is large. This is not easy to tell. Even if the server overwrites part of the memory where c is stored, information may still be encoded in some strange way in the state of the server.

A solution to this kind of issue is to restrict the way a malicious server can manipulate and interact with the data it is storing. We do this by requiring that the entangled data is stored on a memory tape $\mathcal{M}$ to which the server is given a restricted form of access. In particular, we model an adversary $\mathcal{A}$ controlling the server as a Turing machine (with its own memory tape) and we allow $\mathcal{A}$ the following interaction with $\mathcal{M}$. Whenever a recovery query from one of the clients arrives, $\mathcal{A}$ must specify a Turing machine $T(\cdot)$ which is applied to the entangled data in order to compute the answer to that query. The machine $T(\cdot)$ must be read-only, so that with this kind of queries $\mathcal{A}$ can decide to answer honestly or not to a specific query, without necessarily overwriting the data. Furthermore, at any point in time, $\mathcal{A}$ can issue a command that will overwrite some number α of bits of the encoding. Lastly, we always allow $\mathcal{A}$ to maintain a backup copy of the original encoding and restore it at any point in time. This models a cloud provider making a backup copy before overwriting part of the entanglement.

We call such an adversary $\mathcal{A}$ an $(\mathcal{M}, \alpha)$-limited adversary. Notice that in this model a simulator is able to understand whenever $\mathcal{A}$ is in a position such that he has large uncertainty on the encoding he was asked to store. This will allow the

simulation to go through, and in fact we are able to show that the paradigm sketched above for data entanglement in the cloud (based on entangled encoding) works, i.e. $(\Pi_{\mathsf{ETG}}, \Pi_{\mathsf{RCV}})$ securely realizes $\mathcal{I}_{\mathsf{ESS}}$ in the presence of malicious $(\mathcal{M}, \alpha)$-limited adversaries corrupting the server and (part of) the clients.

An instantiations based on polynomials. We then provide a concrete instantiation of the above framework relying on an entangled encoding scheme based on polynomials over a finite field $\mathbb{F}$. Here, the clients encode file f_i by choosing a random pair of elements $(s_i, x_i) \in \mathbb{F}^2$ and defining a point $(x_i, y_i = f_i + s_i)$. The entanglement of $(f_1, \ldots, f_n)$ is now the unique polynomial $c(\cdot)$ of degree $n-1$ interpolating all of (x_i, y_i). In Section 3 we show that if the field $\mathbb{F}$ is large enough, this encoding has $(k, negl(k))$ all-or-nothing integrity for security parameter k and a proper choice of the other parameters.

We finally construct secure protocols $(\Pi_{\mathsf{ETG}}, \Pi_{\mathsf{RCV}})$ for the above encoding scheme, using standard cryptographic building blocks, and we show they securely realize $\mathcal{I}_{\mathsf{ESS}}$ in the model of universal composability [10]. Protocol Π_{ETG} can be seen as an instantiation of the well studied problem of *privacy-preserving polynomial interpolation* (3PI), where a bunch of clients is given as input a point (x_i, y_i) (one for each client) and they want to compute the polynomials $c(\cdot)$ interpolating the points while keeping privacy of the points itself. On the other hand, protocol Π_{RCV} allows a client holding x_i to retrieve $c(x_i) = y_i$ (and nothing more) together with a proof of correctness about the recovered value. We call this a protocol for *secure polynomial evaluation* (SPE). These protocols are described in Section 5.1 and 5.2. Our final construction is a bit more complicated than this, since we need to deal with the fact that the clients cannot store the whole value x_i (otherwise they could just store the file f_i). We refer to Section 5.3 for the details.

1.2 Other Related Work

Data entanglement also appears in the context of censorship-resistant publishing systems; see, e.g, Dagster [42] and Tangler [44]. The notion of SPE is related to oblivious polynomial evaluation (OPE), introduced by Naor and Pinkas [32, 33] and studied also in [11, 47, 19, 24]. A detailed comparison of our protocols with existing solutions is given in Section 5.

The notion of all-or-nothing integrity is inspired by the all-or-nothing transform introduced by Rivest et al. [38], and later generalized in [14]. The standard definition of all-or-nothing transform requires that it should be hard to reconstruct a message if not all the bits of its encoding are known.

2 Preliminaries

Notation. Given an integer n, we denote $[n] = \{1, \ldots, n\}$. If $n \in \mathbb{R}$, we write $\lceil n \rceil$ for the smallest integer greater than n. If x is a string, we denote its length by $|x|$; if $\mathcal{X}$ is a set, $|\mathcal{X}|$ represents the number of elements in $\mathcal{X}$. When x is chosen randomly in $\mathcal{X}$, we write $x \xleftarrow{\$} \mathcal{X}$. When A is an algorithm, we write $y \leftarrow \mathsf{A}(x)$ to denote a run of A on input x and output y; if A is randomized, then y is a random variable and $\mathsf{A}(x; \omega)$ denotes a run of A on input x and random coins ω.

Throughout the paper, we denote the security parameter by k. A function $negl(k)$ is negligible in k (or just negligible) if it decreases faster than the inverse of every polynomial in k. A machine is said to be probabilistic polynomial time (PPT) if it is randomized, and its number of steps is polynomial in the security parameter.

Let $X = \{X_k\}_{k \in \mathbb{N}}$ and $Y = \{Y_k\}_{k \in \mathbb{N}}$ be two distribution ensembles. We say X and Y are ϵ-computationally indistinguishable if for every polynomial time distinguisher $\mathcal{A}$ there exists a function ϵ such that $|Pr(\mathcal{A}(X) = 1) - Pr(\mathcal{A}(Y) = 1)| \leq \epsilon(k)$. If $\epsilon(k)$ is negligible, we simply say X and Y are (computationally) indistinguishable (and we write $X \approx Y$).

The statistical distance of two distributions X, Y is defined through $SD(X, Y) = \sum_a |Pr(X = a) - Pr(Y = a)|$. The min-entropy of a random variable X is $H_\infty(X) = -\max_x \log(Pr(X = x))$.

Succinct argument systems. Let $R \subset \{0,1\}^* \times \{0,1\}^*$ be a polynomial-time relation with language $L_R = \{x : \exists w \text{ s.t. } (x, w) \in R$. A succinct argument system (P, V) for $L \in \mathcal{NP}$ is a pair of probabilistic polynomial-time machines such that the following properties are satisfied: (i) (succinctness) the total length of all messages exchanged during an execution of (P, V) is only polylogarithmic in the instance and witness sizes; (ii) (completeness) for any $x \in L$ we have that $(\mathsf{P}(w), \mathsf{V})(x)$ outputs 1 with overwhelming probability; (iii) (computational soundness) for any $x \notin L$ and any computationally bounded prover P^* we have that $(\mathsf{P}^*, \mathsf{V})(x) = 1$ only with negligible probability; (iv) (argument of knowledge) for any

$x \notin L$ and any computationally bounded prover P^* such that $(\mathsf{P}^*, \mathsf{V})(x)$ outputs 1 there exists a polynomial time extractor $\mathsf{E}_{\mathsf{P}^*}$ outputting a witness w that satisfies $(x, w) \in R$ with overwhelming probability. See for instance [45, 5].

Succinct interactive argument systems for $\mathcal{NP}$ exists in 4 rounds based on the PCP theorem, under the assumption that collision-resistant function ensembles exists [28, 45]. Succinct non-interactive argument systems, also called SNARG, are impossible under any falsifiable cryptographic assumption [23] but are known to exists in the random oracle model [29] or under non-falsifiable cryptographic assumptions [5].

3 Entangled Encoding Schemes

In this section, we introduce the notion of an entangled encoding scheme and show a construction based on polynomial interpolation. Intuitively, an entangled encoding scheme encodes an arbitrary number of input strings $f_1, \ldots, f_n$ into a single output string using random strings $r_1, \ldots, r_n$ (one for each input string). We assume that all input strings have the same length ℓ. The following definition captures an entangled encoding scheme formally.

Definition 3.1 (Entangled Encoding Scheme) *An entangled encoding scheme is a triplet of algorithms* $(\mathsf{Setup}, \mathsf{Encode}, \mathsf{Decode})$ *defined as follows.*

Setup. Setup *is a probabilistic algorithm which, on input a security parameter k, the number of strings to encode n, and the length parameter ℓ, outputs public parameters $(\mathcal{F}, \mathcal{R}, \mathcal{C})$. We call $\mathcal{F}$ the input space, $\mathcal{R}$ the randomness space and $\mathcal{C}$ the entanglement space.*

Encoding. Encode *is a deterministic algorithm which, on input strings $f_1, \ldots, f_n \in \mathcal{F}$ and auxiliary inputs $r_1, \ldots, r_n \in \mathcal{R}$, outputs an encoding $c \in \mathcal{C}$.*

(Local) Decoding. Decode *is a deterministic algorithm which, on input an encoding $c \in \mathcal{C}$ and input $r_i \in \mathcal{R}$ together with index i, outputs string $f_i \in \mathcal{F}$.*

Correctness of decoding requires that for all security parameter k and length ℓ, public parameters $(\mathcal{F}, \mathcal{R}, \mathcal{C}) \leftarrow \mathsf{Setup}(1^k, n, \ell)$, input strings $f_1, \ldots, f_n \in \mathcal{F}$ and auxiliary inputs $r_1, \ldots, r_n \in \mathcal{R}$, we have $f_i = \mathsf{Decode}(\mathsf{Encode}(f_1, \ldots, f_n, r_1, \ldots, r_n), r_i, i)$ for all $i \in [n]$. We let F_i and R_i for $i = 1, \ldots, n$ be random variables representing the choice of f_i and r_i, respectively. We make no assumption on the distributions of F_i and R_i. We let F_{-i} (f_{-i}) denote the set of all variables (values) *except* F_i (f_i). Similar notation is used for R_i and r_i. An entangled encoding scheme satisfies two main security properties.

Privacy: Even if an adversary already knows a subset of the input strings and randomness used to encode them, the resulting encoding reveals no additional information about any of the other input strings other than what can be derived by the knowledge of this subset. More precisely, let U denote some arbitrary subset of the pairs $(F_j, R_j)_{j=1\ldots n}$, and let $\mathcal{D}_u$ denote the distribution of C given $U = u$ where $C = \mathsf{Encode}(F_1, \ldots, F_n, R_1, \ldots, R_n)$. Let V be the set of (F_i, R_i) not included in U, i.e., $V = (F_{-U}, R_{-U})$. An entangled encoding scheme is private if for any u, $\mathcal{D}_u$ is statistically close to $\mathcal{D}_v$ for any v drawn from V, i.e. $SD(\mathcal{D}_u, \mathcal{D}_v) \leq negl(k)$.

All-Or-Nothing Integrity: Roughly speaking, if an adversary has a large amount of uncertainty about the encoding $C = \mathsf{Encode}(F_1, \ldots, F_n, R_1, \ldots, R_n)$, he cannot design a function that will answer decoding queries correctly. More precisely, let U be defined as under privacy, and define a random variable C'_U that is obtained by applying an arbitrary (possibly probabilistic) function $g(\cdot)$ to U and C. Now the adversary plays the following game: he is given that $C'_U = c'$ for any value c' and then specifies a function Decode_{Adv}. We say that the adversary wins at position i if F_i is not included in U and $\mathsf{Decode}_{Adv}(R_i, i) = \mathsf{Decode}(C, R_i, i)$. The encoding has (α, β) all-or-nothing integrity if $H_\infty(C|C'_U = c') \geq \alpha$ implies that for each i, the adversary wins at position i with probability at most β.

Definition 3.2 ((α, β) All-or-Nothing Integrity) *We say that an entangled encoding scheme* $(\mathsf{Setup}, \mathsf{Encode}, \mathsf{Decode})$ *has (α, β) all-or-nothing integrity if the following holds for all probabilistic polynomial time adversaries $\mathcal{A}$ and for all $i \in [n]$:*

$$Pr\left(\mathsf{Decode}_{Adv}(R_i, i) = \mathsf{Decode}(C, R_i, i) : \begin{array}{l}(\mathcal{F}, \mathcal{R}, \mathcal{C}) \leftarrow \mathsf{Setup}(1^k, n, \ell), C = \mathsf{Encode}(F_1, \ldots, F_n; R_1, \ldots, R_n), \\ U \subset \{(F_j, R_j)\}_{j=1\ldots n}, g(\cdot) \leftarrow \mathcal{A}((F_u, R_u)_{u \in U}), C'_U = g(C, U), \\ \mathsf{Decode}_{Adv} \leftarrow \mathcal{A}(C'_U), H_\infty(C|C'_U = c') \geq \alpha, (F_i, R_i) \notin U\end{array}\right) \leq \beta,$$

where the probability is taken over the choices of the random variables F_i, R_i and the coin tosses of $\mathcal{A}$.

Note that β in the definition of all-or-nothing integrity will typically depend on both α and the security parameter k, and we would like that β is negligible in k, if α is large enough. We cannot ask for more than this, since if α is small, the adversary can guess the correct encoding and win with large probability.

We now design an encoding scheme that has the properties we are after. As a first attempt, we consider the following. We choose a finite field $\mathbb{F}$, say of characteristic 2, large enough that we can represent values of F_i as field elements. We then choose $x_1, \ldots, x_n$ uniformly in $\mathbb{F}$ and define the encoding to be c, where c is the polynomial of degree at most $n-1$ such that $c(x_i) = f_i$ for all i. Decoding is simply evaluating c. Furthermore, the all-or-nothing property is at least intuitively satisfied: c has degree at most n and we may think of n as being much smaller than the size of $\mathbb{F}$. Now, if an adversary has many candidates for what c might be, and wants to win the above game, he has to design a single function that agrees with many of these candidates in many input points. This seems hard since the candidates pairwise can only agree in at most n points. We give a more precise analysis later.

Privacy, however, is not quite satisfied: we are given the polynomial c and we want to know how much this tells us about $c(x_i)$ where x_i is uniformly chosen. Note that it does not matter if we are given x_j for $j \neq i$, since all x_j are independent. We answer this question by the following lemma:

Lemma 3.3 *Given a non-constant polynomial c of degree at most n, the distribution of $c(R)$, where R is uniform in $\mathbb{F}$, has min-entropy at least* $\log|\mathbb{F}| - \log(n)$.

Proof. The most likely value of $c(R)$ is the value y for which $c^{-1}(y)$ is of maximal size. This is equivalent to asking for the number of roots in $c(X) - y$ which is at most n, since $c(X) - y$ is not 0 and has degree at most n. Hence $Pr(c(R) = y) \leq n/|\mathbb{F}|$, and the lemma follows by definition of min-entropy. □

It is reasonable to assume that c will not be constant, but even so, we see that distribution of $c(R)$ is not uniform as we would like, but only close (if $n \ll |\mathbb{F}|$). In some applications, a loss of $\log n$ bits in entropy may be acceptable, but it is also easy to fix this by simply one-time pad encrypting the actual data before they are encoded. This leads to the final definition of our encoding scheme:

Setup: Given as input the length ℓ of the n data items to be encoded and the security parameter k, define $\mathcal{F} = \mathbb{F} = GF(2^{max(\ell, 3k+\log n+\log\log n)})$, $\mathcal{R} = \mathbb{F}^2$ and $\mathcal{C} = \mathbb{F}^n$.

Encoding: Given $f_1, \ldots, f_n$ to encode, choose $x_i, s_i \in \mathbb{F}$ uniformly at random, and set $r_i = (x_i, s_i)$. Now define $\mathsf{Encode}(f_1, \ldots, f_n, r_1, \ldots, r_n) = c$, where c is the polynomial of degree at most $n-1$ such that $c(x_i) = f_i + s_i$ for $i = 1, \ldots, n$.

Decoding: We define $\mathsf{Decode}(c, r_i, i) = \mathsf{Decode}(c, (x_i, s_i), i) = c(x_i) + s_i$.

It is trivial to see that decoding works, and that the encoding is private. In fact, by the uniformly random choice of s_i, given an arbitrary subset of the pairs $(F_j, R_j)_{j=1\ldots n}$ the distribution $\mathcal{D}_u$ of C given $U = u$ is independent on the distribution $\mathcal{D}_v$ for $V = (F_{-U}, R_{-U})$ and any $V = v$. For all-or-nothing integrity, we have the theorem below. Its conclusion may seem a bit complicated at first, but in fact, reflects in a natural way that the adversary has two obvious strategies when playing the game from the definition: he can try to guess the correct encoding, which succeeds with probability exponentially small in α, or he can try to guess the correct field element that is computed at the end of the game (by making his function be constant). However, the latter strategy succeeds with probability exponentially small in $|\mathbb{F}|$. The theorem says that, up to constant factor losses in the exponent, these are the only options open to the adversary.

However, before coming to the theorem, we need the following lemma:

Lemma 3.4 *Let U, C'_U be as in the definition of all-or-nothing integrity and suppose the pair $(F_i, R_i) = (F_i, (X_i, S_i))$ is not included in U. Then for the encoding scheme defined above, and for any c', we have $H_\infty(X_i | \, C'_U = c') \geq \log|\mathbb{F}| - \log n$.*

Proof. Suppose first that we are given values for *all* F_j, R_j where $j \neq i$ and also for C and F_i, i.e., we are given the polynomial c, all f_j and all (x_j, s_j), except (x_i, s_i). Let V be a variable representing all this. Before a value of V is given, x_i, s_i are uniformly random and independent of the f_j's and of the (x_j, s_j) where $j \neq i$. It follows that when we are given a value of V, the only new constraint this introduces is that $c(x_i) = s_i + f_i$ must hold. Now, if c is constant, this gives no information at all about x_i, so assume c is not constant. Then for each value s_i, it must be the case that x_i is in a set consisting

of at most n elements, since c has degree at most $n-1$. Therefore we can specify the distribution of x_i induced by this as follows. The set of all x_i is split into at least $|\mathbb{F}|/n$ subsets. Each subset is equally likely (since s_i is uniform a priori), and the elements inside each subset are equally likely (since x_i is uniform a priori). Each subset is, therefore, assigned probability at most $n/|\mathbb{F}|$, and thus, also the largest probability we can assign to an x_i value (if the subset has size 1). Therefore, the conditional min-entropy of X_i is at least $\log|\mathbb{F}| - \log n$.

Now observe that the variable C'_U can be obtained by processing V using a (possibly randomized) function. If we assume that a value of C'_U is given, the conditional min-entropy of X_i is at least as large as when V is given. This actually requires an argument, since it is not the case in general that the min-entropy does not decrease if we are given less information. In our case, however, if we are given $U = u$, the resulting distribution of X_i will be a weighted average computed over the distributions of X_i given values of V that map to $U = u$. But all these distributions have min-entropy at least $\log|\mathbb{F}| - \log n$ and hence so does any weighted average. □

Theorem 3.5 *The above encoding scheme has* $(\alpha, max(2^{-k+2}, 2^{-(\alpha-3)/2}))$ *all-or-nothing integrity.*

Proof. We assume that the distribution $\mathcal{D}$ of the polynomial c in the view of the adversary has min-entropy at least α, so that the maximal probability occurring in the distribution is at most $2^{-\alpha}$. The adversary now submits his function Decode_{Adv}, and he wins if $c(x_i) + s_i = \mathsf{Decode}_{Adv}(x_i, s_i)$ for an arbitrary but fixed $i \in [n]$. We want to bound the adversary's success probability.

Consider any fixed value of s_i and define $g(x_i) = \mathsf{Decode}_{Adv}(x_i, s_i) + s_i$. It is now clear that we may as well bound the probability ϵ that $g(x_i) = c(x_i)$ for a g chosen by the adversary, where c is chosen according to $\mathcal{D}$ and x_i has large min-entropy as shown in Lemma 3.4 above.

Let ϵ_c be the probability that $g(x_i) = c(x_i)$ for a fixed c, then $\epsilon = \sum_c q_c \epsilon_c$ where q_c is the probability assigned to c by $\mathcal{D}$. A standard argument shows that $Pr(\epsilon_c \geq \epsilon/2) \geq \epsilon/2$ since otherwise the average $\sum_c q_c \epsilon_c$ would be smaller than ϵ.

Consider now the distribution $\mathcal{D}'$ which is $\mathcal{D}$ restricted to the c's for which $\epsilon_c \geq \epsilon/2$. The maximal probability in this new distribution is clearly at most $2^{-\alpha+1}/\epsilon$. It follows that $\mathcal{D}'$ assigns non-zero probability to at least $\epsilon 2^{\alpha-1}$ polynomials. We now define $\mathcal{C}'$ be a subset of these polynomials. There are two cases: 1) if $\epsilon 2^{\alpha-1} \leq \sqrt[3]{|\mathbb{F}|/n}$, we set $\mathcal{C}'$ to be all the $\epsilon 2^{\alpha-1}$ polynomials in question; 2) otherwise, we set $\mathcal{C}'$ to be an arbitrary subset of $\sqrt[3]{|\mathbb{F}|/n}$ polynomials.

We now define a modified game, which is the same as the original, except that the polynomial c is chosen uniformly from $\mathcal{C}'$. By construction, we know that the adversary can win with probability $\epsilon/2$ by submitting the function g.

Now define, for $c_i, c_j \in \mathcal{C}'$, the set $\mathcal{X}_{ij} = \{x \in \mathbb{F} \mid c_i(x) = c_j(x)\}$. And let $\mathcal{X} = \cup_{i,j}\mathcal{X}_{ij}$. Since all polynomials in $\mathcal{C}'$ have degree at most $n-1$, it follows that $|\mathcal{X}| \leq n|\mathcal{C}'|^2$. Note that if $x \notin \mathcal{X}$, then $c(x)$ is different for every $c \in \mathcal{C}'$ and one needs to guess c to guess $c(x)$. We can now directly bound the probability we are interested in:

$$\begin{aligned} Pr(g(x) = c(x)) &= Pr(g(x) = c(x) \mid x \in \mathcal{X}) \cdot Pr(x \in \mathcal{X}) + Pr(g(x) = c(x) \mid x \notin \mathcal{X}) \cdot Pr(x \notin \mathcal{X}) \\ &\leq Pr(x \in \mathcal{X}) + Pr(g(x) = c(x) \mid x \notin \mathcal{X}) \\ &\leq \frac{|\mathcal{C}'|^2 n \log n}{|\mathbb{F}|} + \frac{1}{|\mathcal{C}'|}, \end{aligned}$$

where the last inequality follows from Lemma 3.4. Since we already know that there is a way for the adversary to win with probability $\epsilon/2$, we have $\epsilon/2 \leq \frac{|\mathcal{C}'|^2 n \log n}{|\mathbb{F}|} + \frac{1}{|\mathcal{C}'|}$. In case 1), this implies $\epsilon \leq 2^{-(\alpha-3)/2}$, in case 2) we get $\epsilon \leq 2^{-k+3}$. The theorem follows. □

4 Entangled Storage of Data

For reasons of clarity, we define data entanglement for clients each holding only a single file f_i of length ℓ. However, all our definitions and constructions can be easily extended to cover an arbitrary number of files (of arbitrary length) for each party by either encoding multiple files in a single one or by allowing to put in as many files as desired.

4.1 Ideal Implementation of Data Entanglement

We now define an ideal implementation/functionality $\mathcal{I}_{\mathsf{ESS}}$ of an entangled storage scheme. The functionality is shown in Figure 1. The main security requirements captured by $\mathcal{I}_{\mathsf{ESS}}$ are discussed below.

Functionality $\mathcal{I}_{\mathsf{ESS}}$

The functionality $\mathcal{I}_{\mathsf{ESS}}$ is parameterized by the security parameter k, entanglement size n and file space $\mathcal{F}$. Initialize boolean bad as *false*. The interaction with an ordered set of (possibly corrupted) clients $\mathcal{P} = \{P_1, \ldots, P_n\}$, a (possibly corrupted) server S, and ideal adversary $\mathcal{Z}$ is enabled via the following queries:

- On input $(\texttt{Entangle}, P_i, f_i)$ from party P_i, if $f_i \notin \mathcal{F}$, ignore the input; Else, record (P_i, f_i). Ignore any subsequent queries $(\texttt{Entangle}, P_i, *)$ from party P_i. If there are already n recorded tuples (P_j, f_j), send `Entangled` to all parties in $\mathcal{P}$, server S, and adversary $\mathcal{Z}$. Mark session as `Entangled`.
- On input $(\texttt{Overwrite})$ from adversary $\mathcal{Z}$, set bad to *true*.
- On input $(\texttt{Reset})$ from adversary $\mathcal{Z}$, set bad to *false*.
- On input $(\texttt{Recover}, P_i)$ from P_i, check if session is `Entangled`. If not ignore the input. In case bad is *true* output $\perp$ to P_i. Otherwise, record $(\texttt{Pending}, P_i)$ and send $(\texttt{Recover}, P_i)$ to S and $\mathcal{Z}$.
- On input $(\texttt{Recover}, S, i)$ from S or $\mathcal{Z}$, check if session is `Entangled` and record $(\texttt{Pending}, P_i)$ exists. If not, ignore the input. Otherwise:
  - If S is corrupted output $\perp$ to P_i.
  - Else, if P_i is corrupted hand $(\texttt{Cheat}, P_i)$ to $\mathcal{Z}$. Upon receiving $(\texttt{Cheat}, P_i, f_i')$ from $\mathcal{Z}$, output f_i' to P_i.
  - Otherwise, output f_i to P_i.

  Delete record $(\texttt{Pending}, P_i)$.

Figure 1: Ideal functionality $\mathcal{I}_{\mathsf{ESS}}$ for an entangled storage scheme

Privacy of entanglement. We argue that the ideal functionality captures privacy of the entanglement process. That is, the client's files are private to both the other clients and the server. This is represented by the fact that in the ideal execution, at the end of the entanglement process, all participants receive only a message `Entangled` as acknowledgement that the input files are indeed entangled.

Privacy of recovery. The inputs of clients $P_1, \ldots, P_n$ need to remain private also during the recovery process run by client P_i together with S. This is satisfied in the ideal implementation because the server does not get any output when running the recovery process. Furthermore, client P_i queries $(\texttt{Recover}, P_i)$ to the ideal functionality. If client P_i decides to cheat, the adversary simply chooses an arbitrary value f_i' as P_i's output, and thus P_i learns nothing new. Therefore, clients cannot learn about other client's files after the entanglement *and* the recovery process.

All-or-nothing integrity. The server should be unable to deny a given client (or a subset of the clients) access to the entanglement without also impairing all other clients. In other words, if the server modifies the entanglement, *nobody* should be able to recover its original file anymore. In the ideal implementation, the server can send a message $(\texttt{Overwrite})$ to the functionality, indicating that it wants to "forget" part of the data it was supposed to store (e.g., because S wants to sell part of its storage to other clients). Whenever this happens, a boolean value bad is set to *true* and, from this point on, any honest client wishing to recover its file will receive value $\perp$ at the end of the recovery procedure. Of course, a corrupted server can also decide to answer incorrectly to a recovery query without necessary overwriting its memory.

The server is also allowed to restore the original entanglement (by sending a message $(\texttt{Restore})$ to the functionality), in which case bad is reset to *false* and honest clients can recover their file again.

4.2 Entangled Storage Scheme

In a nutshell, an entangled storage scheme $\mathcal{ESS} = (\mathsf{Gen}, \Pi_{\mathsf{ETG}}, \Pi_{\mathsf{RCV}})$ implements an entangled encoding scheme (cf. Definition 3.1) in the cloud setting. More precisely, $\mathcal{ESS}$ enables n clients, in a set $\mathcal{P} = \{P_1, \ldots, P_n\}$, to send their data in a privacy-friendly way to a server S. Assume client P_i holds a file f_i of length ℓ; the server will store a piece of information c which "entangles" all files f_i's while preserving their confidentiality, and such that: (i) the server is unable to alter f_i without harming all others files and (ii) every client can recover its own file f_i (and nothing more).

Definition 4.1 (Entangled Storage Scheme) *Let* $(\mathsf{Setup}, \mathsf{Encode}, \mathsf{Decode})$ *be an entangled encoding scheme. Consider* n *clients in a set* $\mathcal{P} = \{P_1, \ldots, P_n\}$ *and a server* S*. An entangled storage scheme is a tuple* $\mathcal{ESS} = (\mathsf{Gen}, \Pi_{\mathsf{ETG}}, \Pi_{\mathsf{RCV}})$*, defined as follows.*

Parameters Generation. *Upon input a security parameter* k*, the length parameter* ℓ *and the number of clients* n*, the PPT algorithm* Gen *outputs a pair* $(pp_i, sp_i) \leftarrow \mathsf{Gen}(1^k, n)$ *for each client* P_i *where* pp_i *and* sp_i *are the public and secret parameters of client* $i \in [n]$*. In addition,* Gen *runs the* Setup *algorithm of the underlying encoding scheme, yielding a description of the input space* $\mathcal{F}$*, the randomness space* $\mathcal{R}$ *and the entanglement space* $\mathcal{C}$*.*

Entanglement. *The (possibly interactive) protocol* Π_{ETG} *is run by the clients and the server, and computes a functionality* $((f_1, r_1, pp_1, sp_1), \ldots, (f_n, r_n, pp_n, sp_n), -) \mapsto (-, \ldots, -, c)$*, where* $r_i \leftarrow \mathcal{R}$ *and* $c \in \mathcal{C}$ *is an entanglement of the files* $f_i \in \mathcal{F}$*, i.e.* c *is such that* $c = \mathsf{Encode}(f_1, \ldots, f_n, r_1, \ldots, r_n)$*.*

Recovery. *The interactive protocol* Π_{RCV} *is a two-party protocol between a client* P_i *and the server* S*, computing a functionality* $((pp_i, sp_i), c) \mapsto (f_i, -)$*.*

Correctness of $\mathcal{ESS}$ demands that for all $(pp_i, sp_i) \xleftarrow{\$} \mathsf{Gen}(1^k, n)$, for all $(\mathcal{F}, \mathcal{R}, \mathcal{C}) \leftarrow \mathsf{Setup}(1^k, n, \ell)$ and all files $f_1, \ldots, f_n \in \mathcal{F}$, it holds that $c \leftarrow \Pi_{\mathsf{ETG}}((f_1, r_1, pp_1, sp_1), \ldots, (f_n, r_n, pp_n, sp_n), -)$ satisfies $c = \mathsf{Encode}(f_1, \ldots, f_n, r_1, \ldots, r_n)$ and $(f_i, -) \leftarrow \Pi_{\mathsf{RCV}}((pp_i, sp_i), c)$ satisfies $f_i = \mathsf{Decode}(c, r_i, i)$.

Adversarial model. In order to realize the ideal functionality of Figure 1, we need to model a malicious server able to overwrite and/or modify part of the entanglement. This is necessary, because otherwise there is no way to distinguish whether the server has modified the entanglement "on the fly", i.e. only for a given query, or it has actually overwritten part of it. Hence, we will assume that, the entanglement corresponding to a set of files is stored in a memory module $\mathcal{M}$; an adversary $\mathcal{A}$ controlling the server S is modeled as a polynomial-time Turing machine (with its own memory tape) having some form of black-box access to $\mathcal{M}$.

After an entanglement has been established, the adversary can choose to answer a recovery query by modifying arbitrarily the content of $\mathcal{M}$. Furthermore, the adversary can overwrite (at any time) part of the modulo content. Note that we do allow the adversary to have a backup copy of the entanglement and restore its original content after an overwrite query.

Definition 4.2 ($(\mathcal{M}, \alpha)$-limited Adversary) *Let* $\mathcal{M} = (\mathcal{M}_{\text{active}}, \mathcal{M}_{\text{backup}})$ *be memory tapes initially containing a bit-string* $\mathbf{c} = (\mathbf{c}[1], \ldots, \mathbf{c}[l])$ *for some integer* $l \in \mathbb{N}$*. An active adversary* $\mathcal{A}$ *with black-box access to* $\mathcal{M}$ *is called* $(\mathcal{M}, \alpha)$*-limited if it is allowed the following queries to* $\mathcal{M}$*:*

Modify query. *Upon input* $(\texttt{Modify}, T(\cdot))$ *from* $\mathcal{A}$*, where* $T(\cdot)$ *is the description of a polynomial-time* read-only *Turing machine, replace the content of* $\mathcal{M}_{\text{active}}$ *by* $T(\mathbf{c})$*.*

Overwrite query. *Upon input* $(\texttt{Overwrite}, \{i_1, \ldots, i_{\alpha'}\}, (\mathbf{c}'[1], \ldots, \mathbf{c}'[\alpha']))$ *from* $\mathcal{A}$*, where* $\alpha' \geq \alpha$*, replace the content of* $\mathcal{M}_{\text{active}}$ *by* $\mathbf{c}'$ *such that* $\mathbf{c}'[i_j] = \mathbf{c}'[j]$ *for all* $j \in [\alpha']$ *and* $\mathbf{c}'[i] = \mathbf{c}[i]$ *for all other indexes* $i \in [l] \setminus \{i_1, \ldots, i_{\alpha'}\}$*.*

Reset query. *Upon input* $(\texttt{Reset})$ *from* $\mathcal{A}$*, copy the content of* $\mathcal{M}_{\text{backup}}$ *in* $\mathcal{M}_{\text{active}}$*.*

Note that the machine specified in a modify query must be read-only, otherwise a modify query could collapse in an overwrite query. However, $T(\cdot)$ can have its own memory tape. Also, notice that to answer a query honestly the adversary can specify an appropriate Turing machine.[2] The lower bound on the number of bits to be overwritten models the fact that we are interested in the case where the adversary "forgets" a non-negligible fraction of the entanglement. In practice, we can think of α having order of magnitude comparable to the size of one of the files part of the entanglement.

[2] In case the module contains an entanglement in shared form, e.g. the shares of the coefficient of a polynomial to interpolate, the machine $T(\cdot)$ must first reconstruct the shared values.

4.3 The Security Definition

As in the standard static modeling, a corrupted party is either passively or actively controlled by an adversarial entity. In the passive case (a.k.a. semi-honest case) a corrupted party follows the protocol's instructions and tries to gain additional information about the honest parties' inputs from its view; in the active case (a.k.a. malicious case) a corrupted party is allowed to follow an arbitrary polynomial-time strategy.

Formally, denote by $\mathbf{IDEAL}_{\mathcal{I}_{\mathsf{ESS}},\mathcal{Z}(z)}(k,(f_1,\ldots,f_n,-))$ the output of an ideal adversary $\mathcal{Z}$, server S and clients $P_1,\ldots,P_n$ in the above ideal execution of $\mathcal{I}_{\mathsf{ESS}}$ upon inputs $(f_1,\ldots,f_n,-)$ and auxiliary input z given to $\mathcal{Z}$. The functionality $\mathcal{I}_{\mathsf{ESS}}$ is implemented via an entangled storage scheme $\mathcal{ESS} = (\mathsf{Gen}, \Pi_{\mathsf{ETG}}, \Pi_{\mathsf{RCV}})$, as defined in Definition 4.1. We denote by $\mathbf{REAL}_{\mathcal{ESS},\mathcal{A}(z)}(k,(f_1,\ldots,f_n,-))$ the output of adversary $\mathcal{A}$, server S and clients $P_1,\ldots,P_n$ in a real execution of $\mathcal{ESS}$ upon inputs $(f_1,\ldots,f_n,-)$ and auxiliary input z given to $\mathcal{A}$.

Definition 4.3 (Security of Entangled Storage) *We say that $\mathcal{ESS}$* securely implements *$\mathcal{I}_{\mathsf{ESS}}$, if for any probabilistic polynomial time $(\mathcal{M},\alpha)$-limited real adversary $\mathcal{A}$ there exists a probabilistic polynomial-time ideal adversary (simulator) $\mathcal{Z}$ such that for any tuple of inputs $(f_1,\ldots,f_n)$ and auxiliary input z,*

$$\{\mathbf{IDEAL}_{\mathcal{I}_{\mathsf{ESS}},\mathcal{Z}(z)}(k,(f_1,\ldots,f_n,-))\}_{k\in\mathbb{N}} \approx \{\mathbf{REAL}_{\mathcal{ESS},\mathcal{A}(z)}(k,(f_1,\ldots,f_n,-)\}_{k\in\mathbb{N}}.$$

5 Construction

Our construction of $\mathcal{ESS}$ is based on two primitives that we use as building blocks: Privacy-Preserving Polynomial Interpolation (3PI) and Secure Polynomial Evaluation (SPE). For each primitive, we provide both an abstract definition and concrete instantiations. The schemes in this sections are analyzed in the case of standalone security. An extension to the UC-setting can be found in Appendix D.

5.1 Privacy-Preserving Polynomial Interpolation

The problem of privacy-preserving polynomial interpolation (3PI) was introduced in [16] and is defined as follows. We have n clients in a set $\mathcal{P} = \{P_1,\ldots,P_n\}$ where each client P_i holds a point (x_i,y_i) in some finite field $\mathbb{F}$. All clients want to agree on a polynomial $c(X)$ (of minimum degree) such that $y_i = c(x_i)$, for all $i = 1,\ldots,n$, without disclosing the actual points. We are interested here in a slightly different context where an additional party S is added to the set $\mathcal{P}$, but it provides no input. Loosely speaking, an ideal implementation for 3PI realizes a functionality $((x_1,y_1),\ldots,(x_n,y_n),-) \mapsto (-,\ldots,-,c(X))$; we denote this functionality with $\mathcal{I}_{\mathsf{3PI}}$.

Solutions via linear secret sharing or homomorphic encryption. A natural approach to the problem of privacy-preserving polynomial interpolation, is to interpret the inputs of each client as one equation in a system of n equations in n unknowns, the unknowns being the coefficient of the polynomial $c(X)$. In fact, $c(X)$ interpolates all (x_i,y_i) if and only if $\mathbf{A}\cdot\mathbf{c} = \mathbf{b}$ for

$$\mathbf{A} = \begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ & & \vdots & & \\ 1 & x_n & x_n^2 & \ldots & x_n^{n-1} \end{pmatrix} \qquad \mathbf{c} = \begin{pmatrix} c_0 \\ \vdots \\ c_{n-1} \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \tag{1}$$

where $\mathbf{A}$ is a Vandermonde matrix.

We briefly review already known solutions for the above problem. Such solutions can be based on any instantiation of the following two primitives:

- *Threshold additively homomorphic encryption (e.g., Paillier's cryptosystem [35, 18]).* Such a scheme has the following properties: (i) To share a value a party can encrypt it using the public key of the cryptosystem and broadcast the ciphertext; (ii) An encrypted value can be opened using threshold decryption; (iii) Given ciphertexts $\mathsf{Enc}_{pk}(\mu_1)$, $\mathsf{Enc}_{pk}(\mu_2)$ and plaintext μ_3, parties can compute $\mathsf{Enc}_{pk}(\mu_1+\mu_2)$ and $\mathsf{Enc}_{pk}(\mu_3\cdot\mu_1)$ non-interactively; (iv) Given ciphertexts $\mathsf{Enc}_{pk}(\mu_1)$ and $\mathsf{Enc}_{pk}(\mu_2)$, parties can compute $\mathsf{Enc}_{pk}(\mu_1\cdot\mu_2)$ in a constant number of rounds.

- *Linear secret sharing (eg., [40, 20]).* Such a scheme has the following properties: (i) Parties can share a value in a constant number of rounds; (ii) Parties can open a value in a constant number of rounds; (iii) Given shares of values μ_1, μ_2 and value μ_3, parties can compute shares of $\mu_1 + \mu_2$ and $\mu_3 \cdot \mu_1$ non-interactively; (iv) Given shares of values μ_1 and μ_2, parties can compute shares of $\mu_1 \cdot \mu_2$ in a constant number of rounds.

In what follows we say that a value is *shared* if it is distributed according to one of the above two methods; similarly a matrix or a polynomial are shared if all the elements of the matrix or the coefficients of the polynomial are shared.

A solution to 3PI can be obtained using the method of [4] for inverting a matrix in constant round. Let $\mathbf{A} \cdot \mathbf{c} = \mathbf{b}$ be the above system of equations. Note that if the x_i's are distinct, $\mathbf{A}$ is non-singular and can be inverted yielding the desired vector $\mathbf{c} = \mathbf{A}^{-1} \cdot \mathbf{b}$. Denote with $\mathbf{A} = (\mathbf{A}[1], \ldots, \mathbf{A}[n])$ the rows of $\mathbf{A}$ and with $\mathbf{b} = (\mathbf{b}[1], \ldots, \mathbf{b}[n])$ the elements of $\mathbf{b}$; party P_i shares $\mathbf{A}[i]$ and $\mathbf{b}[i]$. Then, the parties share a random non-zero invertible matrix $\mathbf{R}$ (this can be done in constant rounds [4]), compute the shares of $\mathbf{R} \cdot \mathbf{A}$, and reveal the result. At this point, parties can compute the shares of $(\mathbf{R} \cdot \mathbf{A})^{-1} = \mathbf{A}^{-1} \cdot \mathbf{R}^{-1}$ and thus $\mathbf{A}^{-1} \cdot \mathbf{R}^{-1} \cdot \mathbf{R} = \mathbf{A}^{-1}$ non-interactively. Finally, the shares of $\mathbf{A}^{-1} \cdot \mathbf{b}$ can be computed non-interactively.

The method in [4] requires a constant number of rounds and $O(n^3)$ multiplications of shared values. (Recall that in turn each multiplication of shared values requires interaction.) An improvement can be found in [30]. The parties share polynomials $\xi(X) = \prod_{i=1}^{n}(X - x_i)$ and $\xi_i(X) = \xi(X)/(X - x_i)$ (for all $i = 1, \ldots, n$) and values $\xi_i(x_i)$, $\xi_i^{-1}(x_i)$ (for all $i = 1, \ldots, n$). Hence, parties can compute the shares of $\xi_i'(X) = \xi_i(X) \cdot \xi_i(x_i)^{-1}$ (for all $i = 1, \ldots, n$) and obtain the shares of $c(X) = \sum_{i=1}^{n} \xi_i'(X) \cdot \mathbf{b}[i]$. This protocol has a constant number of rounds and $O(n^2)$ multiplications of shared values.

We remark that all the solutions sketched above can be made secure against active adversaries by relying on verifiable secret sharing or specific zero-knowledge proofs.

Solution using OT. A different approach to 3PI can be based on a solution to a related problem, called *privacy preserving cooperative linear system of equation* (PPC-LSE). Every client P_i holds a matrix $\mathbf{A}_i \in \mathbb{F}^{n \times n}$ and a vector $\mathbf{b}_i \in \mathbb{F}^n$. The clients execute a protocol to privately compute a solution $\mathbf{c} \in \mathbb{F}^n$ such that

$$(\mathbf{A}_1 + \cdots + \mathbf{A}_n) \cdot \mathbf{c} = (\mathbf{b}_1 + \cdots + \mathbf{b}_n),$$

where, at the end of the protocol, each client learns $\mathbf{c}[i]$, i.e. a single element of $\mathbf{c}$. Notice that a solution to this problem implies a multi-party protocol Π_{3PI} for 3PI, where each client P_i holds

$$\mathbf{A}_i = \begin{pmatrix} & & \mathbf{0} & & \\ 1 & x_i & x_i^2 & \ldots & x_i^{n-1} \\ & & \mathbf{0} & & \end{pmatrix} \qquad \mathbf{b}_i = \begin{pmatrix} \mathbf{0} \\ y_i \\ \mathbf{0} \end{pmatrix}, \tag{2}$$

and the vector $\mathbf{c}$ contains the coefficients of the polynomial $c(X)$. Indeed, the sum of $\mathbf{A}_i$ results in a Vandermonde matrix.

A solution to this problem exists in [16, 46] but only for the 2-party scenario (in the semi-honest model). In Appendix A, we extend this solution to the multi-party case, assuming one of the players is not willing to collude.

5.2 Secure Polynomial Evaluation

Let $\mathbb{F}$ be a finite field and $c(X) \in \mathbb{F}[X]$ a polynomial stored on a remote server S. Let $\mathbf{c} = (c_1, \ldots, c_n)$ be the coefficients of $c(\cdot)$. A secure polynomial evaluation (SPE) is an interactive protocol Π_{SPE} which allows a client P_i to evaluate $c(X)$ on a chosen input $x \in \mathbb{F}$ without disclosing to S any information about x and $c(X)$.

More precisely, Π_{SPE} is a two-party protocol which computes a functionality $(x, c(\cdot)) \mapsto ((c(x), \nu(\mathbf{c})), -)$, where ν is some arbitrary function $\nu : \mathbb{F}^n \to \{0,1\}^*$. We denote with $\mathcal{I}_{\mathsf{SPE}}$ the ideal functionality associated to SPE.

Possible constructions. A related notion to SPE is oblivious polynomial evaluation (OPE). The difference is that, in OPE, the client does not learn anything about $c(X)$, apart from the value $c(x)$ (i.e., $\nu(\mathbf{c})$ is the empty string).

Clearly any protocol for OPE would work also for SPE. The most efficient instantiation of OPE (due to Hazay and Lindell [24]) is based on Paillier's encryption and runs in a constant number of rounds; the computational complexity consists of $1297n+4160$ exponentiations (where n is the degree of the polynomial held by the server) and the communication complexity is $O(s \cdot n \cdot |N|)$, where s is a security parameter (that should be set to 160) and N is the RSA modulus.

Since SPE is *easier* than OPE, we expect to devise more efficient solutions. We sketch some options below:

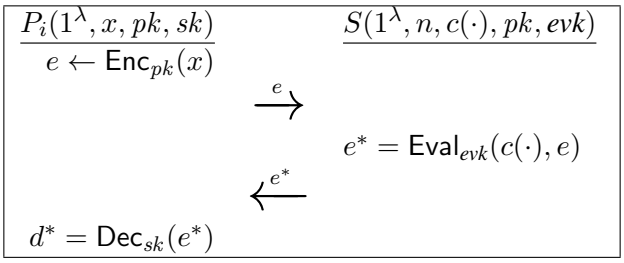

Figure 2: A construction of SPE based on SHE.

- *Naïve approach.* The trivial solution is to simply "leak" the entire polynomial $c(\cdot)$ to the client. This solution requires the transmission of n field elements and the client to evaluate a polynomial over a large field.

- *Homomorphic encryption.* If we use an additive homomorphic encryption scheme, e.g. Paillier [35], then the client could send the powers $\{x^i\}_{i=1}^{n-1}$ encrypted and the server could evaluate $c(x)$ in encrypted form (under P_i's public key). Communication complexity is similar as in the naïve approach but now P_i does not have to compute $c(x)$.

- *Somewhat homomorphic encryption.* The most effective approach is to rely on a (somewhat) homomorphic encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ that is able to perform up to n multiplications and an arbitrarily large number of additions (cf. Appendix B). Here, client P_i sends the point x encrypted and the server simply evaluates $c(x)$ in encrypted form (under P_i's public key) using the Eval procedure. The complete protocol is shown in Figure 2.

We have the following theorem, whose proof can be found in Appendix B.

Theorem 5.1 *Assume the somewhat homomorphic encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *is CPA-secure. Then, the protocol* Π_{SPE} *of Figure 2 securely computes* $\mathcal{I}_{\mathsf{SPE}}$ *in the presence of malicious adversaries.*

Efficiency improvements & trade-offs. The efficiency of the protocol of Figure 2, in reality, depends on the SHE scheme that is employed. For instance, if we consider the schemes in [7, 6], we observe that the ciphertext e^* will be larger as we increase the number of multiplications allowed. Thus, given the current state of efficiency of SHE schemes, the protocol of Figure 2 is less efficient than the solution based on Paillier [35]. (Indeed, with [35], the server would return always an element of $\mathbb{Z}_{N^2}^*$, independently of the number of homomorphic operations performed.)

The following simple observation about the homomorphic encryption approach allows to reduce the communication complexity, while keeping the same computational complexity for P_i. Let $n = (n_1, \ldots, n_\ell)_2$ be the binary representation of the exponent n, for $\ell = \lceil \log_2 n \rceil$, so that $n = \sum_{i=0}^{\ell} 2^i n_i$. It is easy to verify that it is sufficient for the client to transmit $\{\mathsf{Enc}_{pk}(x^{2^i})\}_{i=0}^{\ell}$ to allow S to compute (homomorphically) $\{\mathsf{Enc}_{pk}(x^j)\}_{j=1}^{n}$ and thus $\mathsf{Enc}_{pk}(c(x))$. This reduces the communication from $O(n)$ to $O(\log n)$.

If we allow the client to work a bit more, we can reduce communication further. In Appendix C we present a method to encode a polynomial $c(X)$, which allows the client to evaluate $\mathsf{Enc}_{pk}(c(x))$ by uploading/downloading only $\lceil \sqrt{n} \rceil$ ciphertexts. When combined with the previous trick, this drops down the communication complexity from $O(n)$ to $O(\log \sqrt{n})$.

Yet another trade-off is possible if we assume that P_i and S share a factorization of the polynomial $c(X)$, say $c(X) = \prod_j \gamma_j(X)$ for polynomials $\gamma_j(\cdot)$ of degree δ_j such that $\sum_j \delta_j = n - 1$.[3] In this case, the client *works* more since it has to: (i) compute and send the ciphertexts $\{\mathsf{Enc}_{pk}(x^i)\}_{i=1}^{\delta}$, for $\delta = max(\delta_j)$; (ii) download $\{\mathsf{Enc}_{pk}(\gamma_j(x))\}_j$; (iii) decrypt and multiply the resulting plaintexts.

5.3 Final Protocol

We construct an entangled storage scheme using our entangled encoding scheme $(\mathsf{Setup}, \mathsf{Encode}, \mathsf{Decode})$ based on polynomials over a finite field $\mathbb{F}$ (see Section 3); we do this by combining Π_{3PI} for 3PI (see Section 5.1) and Π_{SPE} for SPE (see

[3] It is well-known that a random polynomial of degree n over a field of prime order is irreducible with probability close to $1/n$. Clients must agree on the factorization of $c(\cdot)$ at the end of the entanglement phase.

Section 5.2). The scheme assumes implicitly that there exists an efficient mapping to encode binary string of length ℓ as elements in a finite field $\mathbb{F}$. Our scheme $\mathcal{ESS} = (\mathsf{Gen}, \Pi_{\mathsf{ETG}}, \Pi_{\mathsf{RCV}})$, as in Definition 4.1, is described below.

Parameters Generation. Upon input a security parameter $k \in \mathbb{N}$, the number of clients n and the length parameter ℓ, the Gen algorithm provides each client P_i with secret parameters $sp_i = \sigma_i$. The secret σ_i is the seed for a (publicly available) pseudo-random generator $G(\cdot)$. In addition, Gen outputs the description of a collision resistant hash function $H(\cdot)$ and $(\mathcal{F} = \mathbb{F}, \mathcal{R} = \mathbb{F}^2, \mathcal{C} = \mathbb{F}^n) \leftarrow \mathsf{Setup}(1^k, n, \ell)$.

Entanglement. Protocol Π_{ETG} is run between clients P_i, holding a point (x_i, y_i) where $y_i = f_i + s_i$ for $G(\sigma_i) = (s_i, x_i)$, and server S (holding no input). The protocol consist of a run of protocol Π_{3PI} from Section 5.1. At the end of the execution the coefficients $c = (c_0, \ldots, c_{n-1})$ of the polynomial $c(X)$ (of minimum degree) interpolating all points (x_i, y_i) are stored in a memory module $\mathcal{M}$ and S is given black-box access to $\mathcal{M}$.

Note that $c = \mathsf{Encode}(f_1, \ldots, f_n, r_1, \ldots, r_n)$ where $r_i = (s_i, x_i) = G(\sigma_i)$. The clients need to store only the seed σ_i, and a hash value $\theta_i = H(y_i)$.

Recovery. Protocol Π_{RCV} is run by client P_i, holding input (σ_i, θ_i), and server S, holding input $c(\cdot)$. To retrieve f_i, client P_i first computes $(s_i, x_i) = G(\sigma_i)$ and then runs protocol Π_{SPE} from Section 5.2 with the server S. In the end, P_i recovers $c(x_i) = y_i$ and thus $f_i = \mathsf{Decode}(c, (s_i, x_i), i) = c(x_i) - s_i$.

In addition S gives a proof of "correct computation" with respect to the operations involved in the SPE protocol. Consider the version of Π_{SPE} based on somewhat homomorphic encryption. Let (P, V) be an interactive argument of knowledge for the following $\mathcal{NP}$-language:[4]

$$L_{\mathrm{SPE}} = \{(e, e^*) : \exists (\mathit{evk}, c(\cdot)) \text{ s.t. } e^* = \mathsf{Eval}_{\mathit{evk}}(e, c(\cdot))\}.$$

The server S plays the role of the prover and client P_i plays the role of the verifier. The client outputs *accept* if and only if $H(y_i) = \theta_i$ and $\mathsf{P}(\mathit{evk}, c(\cdot)), \mathsf{V})(e, e^*) = 1$.

Security analysis. To argue about the security of our construction, we will rely on the security guarantees provided by the underlying building blocks.

Theorem 5.2 (Security of main construction) *Define $\mathbb{F} = GF(2^{max(\ell, 3k + \log n + \log\log n)})$, where ℓ is the length of the files, n is the number of parties and k is the security parameter. Assume protocol Π_{3PI} securely implements $\mathcal{I}_{\mathsf{3PI}}$ in the presence of an active, static adversary with adversary structure Δ_{3PI} and that Π_{SPE} securely implements $\mathcal{I}_{\mathsf{SPE}}$ in the presence of an active adversary. Moreover, let $G : \{0,1\}^k \to \{0,1\}^{2max(\ell, 3k + \log n + \log\log n)}$ be a secure pseudo-random generator. Then, the entangled storage scheme $\mathcal{ESS}$ from above securely implements $\mathcal{I}_{\mathsf{ESS}}$ in the presence of a static, active $(\mathcal{M}, k)$-limited adversary with adversary structure $\Delta^* = \{\emptyset, \Delta_{\mathsf{3PI}}, \{S\}\}$.*

Proof. In general to prove such a statement one should deal with the case of a corrupted server and a honest server separately. However, we can work here with a single simulator which is able to simulate both views of the corrupted server and of the corrupted client. In fact, the only point in the proof where we need to rely on the assumption that the clients and the server are not corrupted at the same time is when we rely on the all-or-nothing integrity of the underlying entangled encoding scheme. Since $\mathcal{A}$ is static, the set of corrupted clients in Δ_{3PI} is fixed once and for all; we denote this set as $\Gamma_{\mathsf{3PI}} \in \Delta_{\mathsf{3PI}}$. We need to show that for any static $(\mathcal{M}, k)$-limited adversary $\mathcal{A}$, there exists a simulator $\mathcal{Z}$ in the ideal functionality $\mathcal{I}_{\mathsf{ESS}}$, such that for all inputs $f_1, \ldots, f_n \in \mathbb{F}$ and auxiliary input z,

$$\{\mathbf{IDEAL}_{\mathcal{I}_{\mathsf{ESS}}, \mathcal{Z}(z)}(k, (f_1, \ldots, f_n, -))\}_{k \in \mathbb{N}} \approx \{\mathbf{REAL}_{\mathcal{ESS}, \mathcal{A}(z)}(k, (f_1, \ldots, f_n, -)\}_{k \in \mathbb{N}}.$$

The simulator $\mathcal{Z}$, with access to $\mathcal{A}$, is described below.

1. Upon input security parameter k, values $\{(f_j, \sigma_j)\}$ (for all indexes j such that $P_j \in \Gamma_{\mathsf{3PI}}$), and auxiliary input z, the simulator invokes $\mathcal{A}$ on these inputs.

[4] A similar language can be defined for the version of Π_{SPE} based on homomorphic encryption. On the other hand, in the naïve solution where the server simply transmits the entire polynomial the argument of knowledge is not needed.

2. Run the simulator $\mathcal{Z}_{\mathsf{3PI}}$ underlying protocol Π_{3PI} on input $\{(f_j, \sigma_j)\}$ (for all indexes j such that $P_j \in \Gamma_{\mathsf{3PI}}$). The simulator $\mathcal{Z}_{\mathsf{3PI}}$ invokes $\mathcal{A}$ who controls all clients $P_j \in \Gamma_{\mathsf{3PI}}$ and S and will choose the values $(x'_i, y'_i = f'_i + s'_i)$ at random for all the honest players $P_i \notin \Gamma_{\mathsf{3PI}}$. Receive $\mathbf{c} = (c_0, \ldots, c_{n-1})$ from $\mathcal{Z}_{\mathsf{3PI}}$. and forward the result to $\mathcal{A}$. Send $(\texttt{Entangle}, P_j, f_j)$ to the ideal functionality $\mathcal{I}_{\mathsf{ESS}}$; receive back message $\texttt{Entangled}$.

3. Upon input $(\texttt{Overwrite}, *, *)$ from $\mathcal{A}$ send $(\texttt{Overwrite})$ to $\mathcal{I}_{\mathsf{ESS}}$. Upon input $(\texttt{Reset})$ from $\mathcal{A}$ send $(\texttt{Reset})$ to $\mathcal{I}_{\mathsf{ESS}}$.

4. Upon input $(\texttt{Recover}, P_i)$ for client P_i, wait for message $(\texttt{Modify}, T(\cdot))$ from $\mathcal{A}$. If such a message is not sent, abort the execution; otherwise send $(\texttt{Recover}, S, i)$ to the ideal functionality. Let $c'(\cdot) = T(c(\cdot))$, i.e. $c'(\cdot)$ is the polynomial obtained by running machine $T(\cdot)$ on input (the coefficients of) $c(\cdot)$.

   Run the simulator $\mathcal{Z}_{\mathsf{SPE}}$ underlying protocol Π_{SPE} on input $c'(\cdot)$. If $\mathcal{Z}_{\mathsf{SPE}}$ sends $\perp$ to its own ideal functionality, $\mathcal{Z}$ sends also $\perp$ leading to an abort of the execution.

   Otherwise, in case P_i is honest play the role of the verifier in (P, V) upon input x'_i (for the previously chosen values (x'_i, y'_i)), with $\mathcal{A}$ being the prover. In case P_i is corrupted, receive $(\texttt{Cheat}, P_i)$ from the trusted party, send back $(\texttt{Cheat}, P_i, y''_i)$ for $y''_i = c'(x''_i)$ and randomly chosen x''_i and play the role of the prover in (P, V), with $\mathcal{A}$ being the verifier.

5. Output whatever $\mathcal{A}$ does.

We consider a series of intermediate hybrid experiments. In the first experiment, we modify the simulator by letting it play directly the role of the trusted party. In particular, the actual modified input $c'(\cdot)$ is used to answer recovery queries by honest clients. To argue indistinguishability, we rely on the the security property of the underlying entangled encoding scheme, the argument of knowledge (P, V) and the collision resistant hash function $H(\cdot)$. In the second experiment, we assume that in contrast to the simulation above, the real values $(x_i, y_i = f_i + s_i)$, for $G(\sigma_i) = (s_i, x_i)$, are used. We argue that this modification is not distinguishable by the adversary due to the pseudo-randomness property of $G(\cdot)$. In the last experiments, we remove the simulators $\mathcal{Z}_{\mathsf{3PI}}$ and $\mathcal{Z}_{\mathsf{SPE}}$ and let the parties execute protocols Π_{3PI} and Π_{SPE} as it would be done in a real execution. Details follow.

Hybrid $\mathcal{H}^1(k, (f_1, \ldots, f_n, -))$. We replace $\mathcal{Z}$ by $\mathcal{Z}^1$ which does not interact with a trusted party. Rather, it receives the real inputs f_i of the honest clients. Simulator $\mathcal{Z}^1$ works essentially as $\mathcal{Z}$, except that it defines the values y'_i of the honest players as $y'_i = f_i + s'_i$ (for random s'_i as before). Moreover, upon input $(\texttt{Overwrite}, \{i_1, \ldots, i_{k'}\}, *)$ from $\mathcal{A}$ (for some $k' \geq k$) modifies $c(\cdot)$ by overwriting the corresponding k' bits chosen by the adversary.

Denote with $c'(\cdot)$ the new polynomial and consider the following events

- FOOL_1: The event becomes true whenever a honest client accepts the output y'_i of a recovery query, and such output corresponds to the overwritten polynomial $c'(\cdot) \neq c(\cdot)$.
- FOOL_2: The event becomes true whenever a client accepts the output of a recovery query, and such output corresponds to the polynomial $T(c(\cdot)) \neq c(\cdot)$.

Let $\widetilde{\mathcal{H}^1}(k, (f_1, \ldots, f_n, -))$ be the probability distribution $\mathcal{H}^1(k, (f_1, \ldots, f_n, -))$ conditioned on $\text{FOOL}_1 \vee \text{FOOL}_2$ not happening. Clearly,

$$\{\mathbf{IDEAL}_{\mathcal{I}_{\mathsf{ESS}}, \mathcal{Z}(z)}(k, (f_1, \ldots, f_n, -))\}_{k \in \mathbb{N}} \equiv \{\widetilde{\mathcal{H}^1}(k, (f_1, \ldots, f_n, -))\}_{k \in \mathbb{N}}.$$

In fact, due to the privacy property of the underlying entangled encoding, replacing the f'_i's files with the real f_i's induces an identical distribution. In particular, provided that FOOL_1 does not happen, a honest client will always reject at the end of the recovery procedure if the output is generated using the overwritten polynomial $c'(\cdot)$. This is exactly what happens in the ideal execution when the value bad is set to *true*. On the other hand, when the adversary does not overwrite $\mathcal{M}$ the value bad is never set to *true* and a honest client will always reject the output of a wrong polynomial $T(c(\cdot)) \neq c(\cdot)$ as long as FOOL_2 does not happen.

The next two claims show that both events FOOL_1 and FOOL_2 happen only with a negligible probability.

Claim 5.3 *$Pr(\text{FOOL}_1)$ is negligible in k.*

Proof (of claim). We rely here on the fact that, when $\mathbb{F} = GF(2^{max(\ell,3k+\log n+\log\log n)})$, our entangled encoding scheme of Section 3 has (α, β) all-or-nothing integrity for $\beta = max(2^{-k+2}, 2^{-(\alpha-3)/2})$. In particular, any (computationally bounded) adversarial strategy provoking event FOOL_1 with probability $\geq \beta$ starting from a polynomial where at least α bits have been overwritten, can be used to break the all-or-nothing integrity property of the encoding scheme.

In the reduction, an adversary attacking the all-or-nothing integrity property of $(\mathsf{Setup}, \mathsf{Encode}, \mathsf{Decode})$ would simply simulate the environment for $\mathcal{A}$ and choose the function $g(\cdot)$ in such a way that $g(c(\cdot)) = c'(\cdot)$, so that he will win if $c'(x_i) = c(x_i)$. Now, since $\mathcal{A}$ is $(\mathcal{M}, k)$-limited, the min-entropy left in the entanglement given $c'(\cdot)$ is at least k bits and thus, by Theorem 3.5, we can conclude $Pr(\text{FOOL}_1) \leq \sqrt{8} \cdot 2^{-k/2}$, which is exponentially small in k.

There is a small caveat, though. The proof of the all-or-nothing integrity property crucially relies on the fact that the function $g(\cdot)$ is chosen independently on x_i. In the above reduction, instead, the adversary sees an encryption of x_i so that it is not immediately clear if the argument goes trough. However, we will argue that if the choice of the function $g(\cdot)$ could depend on x_i, then one could violate semantic security of the encryption scheme underlying protocol Π_{SPE}.

To prove the latter claim, we rely on the extractor $\mathsf{E}_{\mathsf{P}^*}$ of the argument of knowledge P. Consider an adversary $\mathcal{B}$ in the "real-or-random" version of the semantic security game against $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$: Given the public key, $\mathcal{B}$ chooses some message x_i, receives a ciphertext e and has to decide whether this corresponds to an encryption of x_i or it is an encryption of a random message. At this point, $\mathcal{B}$ can simulate the environment for $\mathcal{A}$ by replacing the encryption of x_i with the challenge ciphertext e. Given an accepting answer as output, $\mathcal{B}$ runs the extractor $\mathsf{E}_{\mathsf{P}^*}$ to obtain a witness $(evk^*, c^*(\cdot))$. Hence, if $\mathsf{Eval}_{evk^*}(e, c^*(\cdot)) = c'(x_i)$ the adversary can conclude that e must be an encryption of x_i (i.e., output "real") and otherwise the challenge ciphertext must be completely independent of x_i (i.e., output "random"). We see $\mathcal{B}$ breaks semantic security, and thus the choice of $g(\cdot)$ must be independent of x_i as desired. □

Claim 5.4 *$Pr(\text{FOOL}_2)$ is negligible in k.*

Proof (of claim). One can show that from an adversary $\mathcal{A}$ provoking event FOOL_2, it is possible to build a computationally bounded machine violating both the soundness of (P, V) and the collision resistance of $H(\cdot)$. The reduction is straightforward and is therefore omitted.

Hybrid $\mathcal{H}^2(k, (f_1, \ldots, f_n, -))$. We replace $\mathcal{Z}^1$ by $\mathcal{Z}^2$ which chooses the points (x_i, y_i) of the honest players as in the real protocol, i.e. it defines $y_i = f_i + s_i$ for $G(\sigma_i) = (s_i, x_i)$. We claim that any probabilistic polynomial-time machine distinguishing between the two hybrids can be turned into a probabilistic polynomial-time distinguisher breaking pseudo-randomness of $G(\cdot)$.

The distinguisher is given access to an oracle returning strings $v \in \{0,1\}^{2max(\ell,3k+\log n+\log\log n)}$ with the promise that they are either uniformly distributed or computed through $G(\cdot)$. Hence, the distinguisher interprets v_i as an element in $\mathbb{F}^2$, parses v_i as $v_i = (s_i, x_i)$ and uses these values together with files f_i to define the input of $\mathcal{Z}_{\mathsf{3PI}}$. Now, when the v_i's are uniform, the distribution is the same as in hybrid $\widetilde{\mathcal{H}^1}(k, (f_1, \ldots, f_n, -))$, whereas when $v_i = G(\sigma_i)$ the distribution is the same as in hybrid $\mathcal{H}^2(k, (f_1, \ldots, f_n, -))$. Thus, given a distinguisher between the two hybrids we can break the pseudo-randomness of $G(\cdot)$.

Hybrid $\mathcal{H}^3(k, (f_1, \ldots, f_n, -))$. We replace $\mathcal{Z}^2$ by $\mathcal{Z}^3$ which executes protocols Π_{3PI} and Π_{SPE} as it would be done in a real execution, i.e. without relying on simulators $\mathcal{Z}_{\mathsf{3PI}}$ and $\mathcal{Z}_{\mathsf{SPE}}$. Indistinguishability of $\mathcal{H}^3(k, (f_1, \ldots, f_n, -))$ and $\mathcal{H}^2(k, (f_1, \ldots, f_n, -))$ immediately follows from the security properties of the underlying protocols.

It is easy to see that the output distribution of the last hybrid experiment is identical to the distribution resulting from a real execution of the protocol, thus showing that $\mathbf{IDEAL}_{\mathcal{I}_{\mathsf{ESS}}, \mathcal{Z}(z)}(k, (f_1, \ldots, f_n, -))$ and $\mathbf{REAL}_{\mathcal{ESS}, \mathcal{A}(z)}(k, (f_1, \ldots, f_n, -)$ are computationally close. This concludes the proof. □

References

[1] J. Aspnes, J. Feigenbaum, A. Yampolskiy, and S. Zhong. Towards a theory of data entanglement. *Theor. Comput. Sci.*, 389(1-2):26–43, 2007.

[2] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. X. Song. Provable data possession at untrusted stores. In *ACM CCS*, pages 598–609, 2007.

[3] G. Ateniese, S. Kamara, and J. Katz. Proofs of storage from homomorphic identification protocols. In *ASIACRYPT*, pages 319–333, 2009.

[4] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *PODC*, pages 201–209, 1989.

[5] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, pages 326–349, 2012.

[6] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, page to appear, 2012.

[7] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *FOCS*, pages 97–106, 2011.

[8] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *CRYPTO*, pages 505–524, 2011.

[9] G. Brassard, C. Crépeau, and J.-M. Robert. All-or-nothing disclosure of secrets. In *CRYPTO*, pages 234–238, 1986.

[10] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[11] Y.-C. Chang and C.-J. Lu. Oblivious polynomial evaluation and oblivious neural learning. *Theor. Comput. Sci.*, 341(1-3):39–54, 2005.

[12] J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *CRYPTO*, pages 487–504, 2011.

[13] I. Damgrd, J. B. Nielsen, and C. Orlandi. Essentially optimal universally composable oblivious transfer. Cryptology ePrint Archive, Report 2008/220, 2008.

[14] F. Davì, S. Dziembowski, and D. Venturi. Leakage-resilient storage. In *SCN*, pages 121–137, 2010.

[15] Y. Dodis, S. P. Vadhan, and D. Wichs. Proofs of retrievability via hardness amplification. In *TCC*, pages 109–127, 2009.

[16] W. Du and M. J. Atallah. Privacy-preserving cooperative scientific computations. In *CSFW*, pages 273–294, 2001.

[17] M. Dubovitskaya, A. Scafuro, and I. Visconti. On efficient non-interactive oblivious transfer with tamper-proof hardware. Cryptology ePrint Archive, Report 2010/509, 2010.

[18] P.-A. Fouque and D. Pointcheval. Threshold cryptosystems secure against chosen-ciphertext attacks. In *ASIACRYPT*, pages 351–368, 2001.

[19] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, pages 303–324, 2005.

[20] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In *PODC*, pages 101–111, 1998.

[21] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[22] C. Gentry and S. Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *EUROCRYPT*, pages 129–148, 2011.

[23] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108, 2011.

[24] C. Hazay and Y. Lindell. Efficient oblivious polynomial evaluation with simulation-based security. *IACR Cryptology ePrint Archive*, 2009:459, 2009.

[25] A. Juels and B. S. K. Jr. PoRs: proofs of retrievability for large files. In *ACM CCS*, pages 584–597, 2007.

[26] S. Kamara, P. Mohassel, and M. Raykova. Outsourcing multi-party computation. *IACR Cryptology ePrint Archive*, 2011:272, 2011.

[27] J.-S. Kang and D. Hong. A practical privacy-preserving cooperative computation protocol without oblivious transfer for linear systems of equations. *JIPS*, 3(1):21–25, 2007.

[28] J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992.

[29] S. Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[30] P. Mohassel and M. K. Franklin. Efficient polynomial operations in the shared-coefficients setting. In *Public Key Cryptography*, pages 44–57, 2006.

[31] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *CCSW*, pages 113–124, 2011.

[32] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *STOC*, pages 245–254, 1999.

[33] M. Naor and B. Pinkas. Oblivious polynomial evaluation. *SIAM J. Comput.*, 35(5):1254–1281, 2006.

[34] M. Naor and G. N. Rothblum. The complexity of online memory checking. In *FOCS*, pages 573–584, 2005.

[35] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[36] C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, volume 5157 of *LNCS*, pages 554–571. 2008.

[37] M. O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005.

[38] R. L. Rivest. All-or-nothing encryption and the package transform. In *FSE*, pages 210–218, 1997.

[39] H. Shacham and B. Waters. Compact proofs of retrievability. In *ASIACRYPT*, pages 90–107, 2008.

[40] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[41] N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *PKC*, pages 420–443, 2010.

[42] A. Stubblefield and D. Wallach. Dagster: Censorship-resistant publishing without replication. Technical Report TR01-380, Rice University, 2001.

[43] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010.

[44] M. Waldman and D. Mazières. Tangler: A censorship-resistant publishing system based on document entanglements. In *ACM CCS*, pages 126–135, 2001.

[45] H. Wee. On round-efficient argument systems. In *ICALP*, pages 140–152, 2005.

[46] X. Yang, Z. Yu, and B. Kang. Privacy-preserving cooperative linear system of equations protocol and its application. In *WiCOM*, pages 1–4, 2008.

[47] H. Zhu and F. Bao. Augmented oblivious polynomial evaluation protocol and its applications. In *ESORICS*, pages 222–230, 2005.

A A Solution to 3PI based on OT

Before describing the protocol, we sketch how to handle non-colluding parties in secure computation. We start establishing some notation. Let $\phi : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a functionality, where $\phi_i(x_1,\ldots,x_n)$ denotes the i-th element of $\phi(x_1,\ldots,x_n)$ for $i \in [n]$. The input-output behavior of ϕ is denoted $(x_1,\ldots,x_n) \mapsto (y_1,\ldots,y_n)$. Further, let Π be a multi-party protocol for computing ϕ. We analyze the security of our multi-party computation protocols in the simulation-based model. Roughly, in this model one shows that the view (or resp. output) of any adversary $\mathcal{A}$ involved in the protocol Π can be simulated by ideal-world adversary $\mathcal{Z}$, sometimes called simulator, who interacts directly with an ideal functionality $\mathcal{I}_\phi$. More precisely, in the *real world*, clients $P_1,\ldots,P_n$ execute protocol Π in order to compute the functionality $\phi(x_1,\ldots,x_n)$ in the presence of an (efficient) adversary $\mathcal{A}$. Whereas in the *ideal world*, the computation of ϕ is performed by an ideal functionality $\mathcal{I}_\phi$ which receives all inputs by clients $P_1,\ldots,P_n$ and returns to the clients their respective outputs $\phi_i(x_1,\ldots,x_n)$. Clearly, the clients do not learn any information other than their input/output given by $\mathcal{I}_\phi$ in an execution in the ideal world.

We only allow static corruptions, that is, adversaries determine the clients to corrupt at the beginning of the protocol execution. The security of a multi-party computation protocol is usually defined with respect to an *adversary structure* Δ, i.e. a monotone set of subsets of the players, where the adversary may corrupt the players of one set in Δ. An adversary structure is monotone in the sense of being closed with respect to taking subsets. Let $\mathbf{REAL}_{\Pi,\mathcal{A}(z)}(k,(x_1,\ldots,x_n))$ denote the joint output of adversary $\mathcal{A}$ (holding auxiliary input z) and clients $P_1,\ldots,P_n$ in an execution of protocol Π on inputs $(x_1,\ldots,x_n)$ and security parameter k. Similarly, let $\mathbf{IDEAL}_{\mathcal{I}_\phi,\mathcal{Z}(z)}(k,(x_1,\ldots,x_n))$ denote the joint output of ideal-world adversary $\mathcal{Z}$ and clients $P_1,\ldots,P_n$ in an execution with ideal functionality $\mathcal{I}_\phi$ and inputs $(x_1,\ldots,x_n)$ with security parameter k. Then, protocol Π securely (resp. privately) computes $\mathcal{I}_\phi$ if for every (efficient) malicious (resp. semi-honest) real-world adversary $\mathcal{A}$, there exists an (efficient) ideal-world adversary $\mathcal{Z}$ such that

$$\mathbf{IDEAL}_{\mathcal{I}_\phi,\mathcal{Z}(z)}(k,(x_1,\ldots,x_n)) \approx \mathbf{REAL}_{\Pi,\mathcal{A}(z)}(k,(x_1,\ldots,x_n)).$$

A semi-honest party follows faithfully the protocol specifications but can save intermediate computations. A protocol is secure in the semi-honest model if a semi-honest party cannot obtain more information during an execution of the protocol than what can be inferred from its input and output. In the standard definition of secure computation the adversary $\mathcal{A}$ is considered as *monolithic*. This automatically models collusion between clients and gives strong security guarantees. For some protocols one may be able to prove security in a more restricted setting where some of the parties are not wishing to collude. This needs to be defined explicitly; here we adopt the formalism of [26].

Instead of considering a single adversary which gets to see the state and all the messages exchanged by the corrupted clients, we consider a set of non-monolithic adversaries, each corrupting at most one (non-colluding) party and having access only to the view of that party. A different (monolithic) adversary controls the set of colluding clients. Security is defined by requiring that indistinguishability between the real and ideal world distributions hold with respect to the honest clients' outputs and a single adversary's view. In other words, for each independent adversary $\mathcal{A}_i$, the joint distribution composed of the honest clients' outputs and $\mathcal{A}_i$'s view in the real world—denoted $\mathbf{REAL}^{(i)}_{\Pi,\mathcal{A}(z)}(k,(x_1,\ldots,x_n))$—should be indistinguishable from the joint distribution composed of the honest clients' outputs and the simulator $\mathcal{Z}_i$'s output in the ideal world—denoted $\mathbf{IDEAL}^{(i)}_{\mathcal{I}_\phi,\mathcal{Z}(z)}(k,(x_1,\ldots,x_n))$. We refer the reader to [26, Definition 4.1] for the details.

The following two sub-protocols Π_1 and Π_2 will serve as building blocks for the final construction.

- The first sub-protocol Π_1 allows client P_1, called initiator, to privately retrieve $\widehat{\mathbf{A}} := \mathbf{P}(\mathbf{A}_1 + \ldots + \mathbf{A}_n)\mathbf{Q}$ where $\mathbf{P},\mathbf{Q}$ are randomly chosen matrices.

Protocol Π_1

Steps performed by clients $P_1, \ldots, P_n$. Each P_i holds $\mathbf{A}_i \in \mathbb{F}^{n \times n}$.
Parameters p, m are chosen such that $\log(p)m = O(k)$.

1. Clients P_1 and P_2 decompose $\mathbf{A}_1$ (resp. $\mathbf{A}_2$) by sampling random matrices $\mathbf{X}_1^{(1)}, \ldots, \mathbf{X}_m^{(1)}$ satisfying $\mathbf{A}_1 = \mathbf{X}_1^{(1)} + \ldots + \mathbf{X}_m^{(1)}$ (resp. $\mathbf{A}_2 = \mathbf{X}_1^{(2)} + \ldots + \mathbf{X}_m^{(2)}$).
2. For each $j = 1, \ldots, m$, client P_1 and P_2 perform the following sub-steps:
   (a) P_1 sends $(\mathbf{H}_1, \ldots, \mathbf{H}_p)$ to client P_2 where $\mathbf{H}_l = \mathbf{X}_j^{(1)}$ for a secretly chosen index $l \in [p]$ and $\mathbf{H}_i$ (with $i \neq l$) sampled uniformly.
   (b) P_2 chooses random matrices $\mathbf{P}, \mathbf{Q}$ and computes $\mathbf{P}(\mathbf{H}_i + \mathbf{X}_j^{(2)})\mathbf{Q} + \mathbf{R}_j$ for each $i = 1, \ldots, p$, where $\mathbf{R}_j$ is a random matrix.
   (c) P_1 executes 1-out-of-p OT and learns $\mathbf{P}(\mathbf{H}_l + \mathbf{X}_j^{(2)})\mathbf{Q} + \mathbf{R}_j = \mathbf{P}(\mathbf{X}_j^{(1)} + \mathbf{X}_j^{(2)})\mathbf{Q} + \mathbf{R}_j$.
3. Client P_2 sends to clients P_i (with $i = 3, \ldots, n$) the matrices $\mathbf{P}$ and $\mathbf{Q}$, and $\sum_{j=1}^{m} \mathbf{R}_j$ to P_1.
4. Clients P_i with $i = 3, \ldots, n$ send $\mathbf{P}\mathbf{A}_i\mathbf{Q}$ to client P_1.
5. Client P_1, after receiving all values, computes:

$$\begin{aligned}\widehat{\mathbf{A}} &= \sum_{j=1}^{m}(\mathbf{P}(\mathbf{X}_j^{(1)} + \mathbf{X}_j^{(2)})\mathbf{Q} + \mathbf{R}_j) - \sum_{j=1}^{m}\mathbf{R}_j + \mathbf{P}\mathbf{A}_3\mathbf{Q} + \ldots + \mathbf{P}\mathbf{A}_n\mathbf{Q} \\ &= \mathbf{P}(\mathbf{A}_1 + \mathbf{A}_2)\mathbf{Q} + \mathbf{P}\mathbf{A}_3\mathbf{Q} + \ldots + \mathbf{P}\mathbf{A}_n\mathbf{Q} \\ &= \mathbf{P}(\mathbf{A}_1 + \mathbf{A}_2 + \mathbf{A}_3 + \ldots + \mathbf{A}_n)\mathbf{Q}.\end{aligned}$$

Figure 3: Description of protocol Π_1

- The second sub-protocol Π_2 allows the initiator P_1 to obtain privately $\widehat{\mathbf{b}} := \mathbf{P}(\mathbf{b}_1 + \ldots + \mathbf{b}_n)$.

The initiator P_1 solves the linear equation $\widehat{\mathbf{A}} \cdot \widehat{\mathbf{c}} = \widehat{\mathbf{b}}$ and sends $\widehat{\mathbf{c}}$ to client P_2 (called the assembler). P_2 derives $\mathbf{c} = \mathbf{Q} \cdot \widehat{\mathbf{c}}$ as the final solution to the equation $(\mathbf{A}_1 + \ldots + \mathbf{A}_n) \cdot \mathbf{c} = (\mathbf{b}_1 + \ldots + \mathbf{b}_n)$. Recall that privacy and security hold with respect to semi-honest clients $P_1, \ldots, P_n$. In all (sub)protocols, we merely assume client P_1 does not collude to ensure privacy of clients's inputs. The collusion of any (subset of) client(s) excluding P_1 does not harm the privacy.

First sub-protocol. Π_1 is described in Figure 3. We prove that protocol Π_1 privately computes $\mathcal{I}_{\mathsf{LSE}_1}$ in the semi-honest model. The ideal functionality $\mathcal{I}_{\mathsf{LSE}_1}$ receives as input $\mathbf{A}_i$ of client P_i and outputs to client P_1 the value $\mathbf{P}(\mathbf{A}_1 + \ldots + \mathbf{A}_n)\mathbf{Q}$, where $\mathbf{P}, \mathbf{Q} \xleftarrow{\$} \mathbb{F}^{n \times n}$. All other clients $P_2, \ldots, P_n$ receive the empty string.

Theorem A.1 (Privacy of protocol Π_1) *Assuming the clients $P_1, \ldots, P_n$ are semi-honest, and P_1 does not collude, the protocol Π_1 enables $P_1, \ldots, P_n$ to privately compute functionality $\mathcal{I}_{\mathsf{LSE}_1}$.*

Proof. Since client P_1 is non-colluding, we need to consider a non-monolithic adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as described in Section 2. At the beginning of the execution, $\mathcal{A}_1$ is given the index corresponding to the non-colluding (corrupted) client P_1, whereas $\mathcal{A}_2$ is given the indexes corresponding to the (corrupted) colluding clients. We denote the latter set with J and we let H be the set of (indexes corresponding to) honest clients; as we only consider static adversaries these sets are fixed once the protocol starts. Moreover, since P_1 and P_2 have a special role in the protocol, we will assume (without loss of generality) that these clients are always corrupted; the case when P_1 and P_2 are honest can be easily derived as a special case.

In order to prove privacy, we build simulator $\mathcal{Z} = (\mathcal{Z}_1, \mathcal{Z}_2)$ whose output in a joint execution with clients $P_1, \ldots, P_n$ interacting with ideal functionality $\mathcal{I}_{\mathsf{LSE}_1}$ is computationally indistinguishable from a real execution with an adversary $\mathcal{A}$. In particular, for each independent adversary $\mathcal{A}_i$, we need to show that the joint distribution composed of the honest clients' outputs and $\mathcal{A}_i$'s view in the real world is indistinguishable from the joint distribution composed of the honest clients' outputs and the simulator $\mathcal{Z}_i$'s output in the ideal world.

We introduce the following notation. The view of client P_i during an execution of Π on $(x_1, \ldots, x_n)$, denoted by $\mathbf{VIEW}_i^{\Pi}(x_1, \ldots, x_n)$, is $(x_i, \omega_i, \mu_1^i, \ldots, \mu_t^i)$, where x_i is P_i's input, ω_i is P_i's internal coin tosses, and μ_j^i is the j-th received message by client P_i. The output of client P_i is denoted by $\mathbf{REAL}_i^{\Pi}(x_1, \ldots, x_n)$ which is included by definition in its view.

As for the non-colluding client, $\mathbf{REAL}^{(1)}_{\Pi_1, \mathcal{A}(z)}(k, (x_1, \ldots, x_n))$ is defined by

$$\{\mathbf{REAL}_i^{\Pi_1}(\mathbf{A}_1, \ldots, \mathbf{A}_n) : \ i \in H\} \cup \mathbf{VIEW}_1^{\Pi_1}(\mathbf{A}_1, \ldots, \mathbf{A}_n).$$

In the ideal world, let $\mathbf{REAL}_i^{\Pi_1}$ be the output returned to $\{P_i\}_{i \in H}$ by the trusted party. Then, $\mathbf{IDEAL}^{(1)}_{\mathcal{I}_{\mathsf{LSE}_1}, \mathcal{Z}(z)}(k, (x_1, \ldots, x_n))$ is defined by

$$\{\mathbf{REAL}_i^{\Pi_1} = - : \ i \in H\} \cup \mathcal{Z}_1(\mathbf{A}_1, \widehat{\mathbf{A}}).$$

In a similar fashion, as for the colluding clients, $\mathbf{REAL}^{(2)}_{\Pi_1, \mathcal{A}(z)}(k, (x_1, \ldots, x_n))$ is defined by

$$\{\mathbf{REAL}_i^{\Pi_1}(\mathbf{A}_1, \ldots, \mathbf{A}_n) : \ i \in H\} \cup \{\mathbf{VIEW}_j^{\Pi_1}(\mathbf{A}_1, \ldots, \mathbf{A}_n)\}_{j \in J}$$

and $\mathbf{IDEAL}^{(2)}_{\mathcal{I}_{\mathsf{LSE}_1}, \mathcal{Z}(z)}(k, (x_1, \ldots, x_n))$ is defined by

$$\{\mathbf{REAL}_i^{\Pi_1} = - : \ i \in H\} \cup \{\mathcal{Z}_2(\mathbf{A}_j) : \ j \in J\}.$$

Description of Simulator. We start with the description of $\mathcal{Z}_1$.

- Upon input $(\mathbf{A}_1, \widehat{\mathbf{A}})$, $\mathcal{Z}_1$ samples uniformly matrices $\mathbf{P}'$ and $\mathbf{Q}'$ which will simulate the matrices $\mathbf{P}, \mathbf{Q}$.
- Afterwards, $\mathcal{Z}_1$ samples uniformly $\mathbf{A}'_i \in_R \mathbb{F}^{n \times n}$ for $i = 3, \ldots, n$ and computes $\mathbf{P}'\mathbf{A}'_i\mathbf{Q}'$ which simulates the matrices $\mathbf{P}\mathbf{A}_i\mathbf{Q}$ sent by client P_i. In our case, the matrices $\mathbf{A}_i$ have a special structure (they are sparse and only one row is non-zero). In this case, $\mathcal{Z}_1$ picks randomly $x_i \in_R \mathbb{F}$ (for $i = 3, \ldots, n$) and generates matrices $\mathbf{A}'_i$ according to the special structure.
- $\mathcal{Z}_1$ finds $\mathbf{A}'_2$ (to simulate $\mathbf{A}_2$) by solving

$$\mathbf{P}'(\mathbf{A}_1 + \mathbf{A}'_2)\mathbf{Q}' = \widehat{\mathbf{A}} - (\mathbf{P}'\mathbf{A}'_3\mathbf{Q}' + \ldots + \mathbf{P}'\mathbf{A}'_n\mathbf{Q}').$$

- $\mathcal{Z}_1$ generates m random matrices $\mathbf{Y}_1^{(2)}, \ldots, \mathbf{Y}_m^{(2)}$ s.t. $\mathbf{A}'_2 = \mathbf{Y}_1^{(2)} + \ldots + \mathbf{Y}_m^{(2)}$.
- $\mathcal{Z}_1$ generates random matrices $\mathbf{X}_j^{(1)}$ for $j = 1, \ldots, m$ using the same coin tosses ω that P_1 uses in generating these matrices.
- $\mathcal{Z}_1$ generates random matrices $\mathbf{R}_j$ for $j = 1, \ldots, m$.

We define $\mathcal{Z}_1(\mathbf{A}_1, \widehat{\mathbf{A}})$ as

$$\{\mathbf{A}_1, \omega, \mathbf{P}'(\mathbf{X}_1^{(1)} + \mathbf{Y}_1^{(2)})\mathbf{Q}' + \mathbf{R}_1, \ldots, \mathbf{P}'(\mathbf{X}_m^{(1)} + \mathbf{Y}_m^{(2)})\mathbf{Q}' + \mathbf{R}_m, \sum_{j=1}^{m} \mathbf{R}_j, \mathbf{P}'\mathbf{A}'_3\mathbf{Q}', \ldots, \mathbf{P}'\mathbf{A}'_n\mathbf{Q}'\}.$$

Next, we describe simulator $\mathcal{Z}_2$ for simulating $\{\mathbf{VIEW}_j^{\Pi_1}(\mathbf{A}_1, \ldots, \mathbf{A}_n)\}_{j \in J}$. The simulator takes as input $\mathbf{A}_j$ for $j \in J$. Note that protocol Π_1 provides no output to client P_j $(j \in J)$. Thus, $\mathcal{Z}_2$ simply generates $m \cdot p$ uniformly chosen matrices $(\mathbf{H}'_{1,1}, \ldots, \mathbf{H}'_{1,p}), \ldots, (\mathbf{H}'_{m,1}, \ldots, \mathbf{H}'_{m,p})$, uses the randomness ω to sample $\mathbf{X}_1^{(2)}, \ldots, \mathbf{X}_m^{(2)}$ and defines the view of client P_2 as $\{\mathbf{A}_2, \omega, \mathbf{X}_1^{(2)}, \ldots, \mathbf{X}_m^{(2)}, (\mathbf{H}'_{1,1}, \ldots, \mathbf{H}'_{1,p}), \ldots, (\mathbf{H}'_{m,1}, \ldots, \mathbf{H}'_{m,p})\}$.

For the view of the remaining clients in J, the simulator $\mathcal{Z}_2$, upon input $\{\mathbf{A}_j\}_{j \in J}$ of clients P_j, samples uniformly matrices $\mathbf{P}'$ and $\mathbf{Q}'$ which will simulate the matrices $\mathbf{P}, \mathbf{Q}$. Therefore, we define the view of P_j as $\{\mathbf{A}_j, \omega, (\mathbf{P}', \mathbf{Q}')\}$ for $j \in J$ and $j \neq 2$.

Analysis. Protocol Π_1 privately computes $\mathbf{P}(\mathbf{A}_1 + \ldots + \mathbf{A}_n)\mathbf{Q}$ if the following statements hold:

1. $\mathbf{REAL}^{(1)}_{\Pi_1,\mathcal{A}(z)}(k,(x_1,\ldots,x_n)) \equiv_c \mathbf{IDEAL}^{(1)}_{\mathcal{I}_{\mathsf{LSE}_1},\mathcal{Z}(z)}(k,(x_1,\ldots,x_n))$;
2. $\mathbf{REAL}^{(2)}_{\Pi_1,\mathcal{A}(z)}(k,(x_1,\ldots,x_n)) \equiv_c \mathbf{IDEAL}^{(2)}_{\mathcal{I}_{\mathsf{LSE}_1},\mathcal{Z}(z)}(k,(x_1,\ldots,x_n))$.

Statement (1). Recall that the view of client P_1 in the protocol is defined by $\mathbf{VIEW}_1^{\Pi_1}(\mathbf{A}_1,\ldots,\mathbf{A}_n) = \{\mathbf{A}_1, \omega, \mathbf{P}(\mathbf{X}_1^{(1)} + \mathbf{X}_1^{(2)})\mathbf{Q} + \mathbf{R}_1, \ldots, \mathbf{P}(\mathbf{X}_m^{(1)} + \mathbf{X}_m^{(2)})\mathbf{Q} + \mathbf{R}_m, \sum_{j=1}^m \mathbf{R}_j, \mathbf{P}\mathbf{A}_3\mathbf{Q}, \ldots, \mathbf{P}\mathbf{A}_n\mathbf{Q}\}$. Since $\mathbf{P}', \mathbf{Q}'$ are sampled uniformly from all matrices of size $n \times n$, the simulation of matrices $\mathbf{P}, \mathbf{Q}$ is statistically close. The same argument holds for matrices $\mathbf{Y}_1^{(2)}, \ldots, \mathbf{Y}_m^{(2)}$ simulating $\mathbf{X}_1^{(2)}, \ldots, \mathbf{X}_m^{(2)}$ perfectly. All matrices $\mathbf{A}'_i$ are built according to the predefined structure and thus, are sampled from the same distribution as $\mathbf{A}_i$. Hence, both random variables $\mathcal{Z}_1(\mathbf{A}_1, \widehat{\mathbf{A}})$ and $\mathbf{VIEW}_1^{\Pi_1}(\mathbf{A}_1,\ldots,\mathbf{A}_n)$ are (computationally) indistinguishable.[5]

Statement (2). The view of client P_2 in the protocol is $\mathbf{VIEW}_2^{\Pi_1}(\mathbf{A}_1,\ldots,\mathbf{A}_n) = \{\mathbf{A}_2, \omega, \mathbf{X}_1^{(2)}, \ldots, \mathbf{X}_m^{(2)}, (\mathbf{H}_{1,1},\ldots,\mathbf{H}_{1,p}), \ldots, (\mathbf{H}_{m,1},\ldots,\mathbf{H}_{m,p})\}$. However, since matrices $\mathbf{H}$'s are randomly chosen, we cannot conclude immediately that $\mathcal{Z}_2$ simulates $\mathbf{VIEW}_2^{\Pi_1}(\mathbf{A}_1,\ldots,\mathbf{A}_n)$ correctly. In fact, the matrices $\mathbf{H}$'s in the real protocol are not all random, but satisfy the following property: There exists a vector $\mathbf{l} = (l_1,\ldots,l_m)$ (with $1 \le l_i \le p$) such that $\sum_{i=1}^m \mathbf{H}_{i,l_i} = \mathbf{A}_1$. However, it is not possible to identify such a vector with probability better than $1/p^m$ which is negligible since $\log(p)m = O(k)$. We can thus conclude that the output produced by $\mathcal{Z}_2$ is computationally indistinguishable from the view of party P_2.

Similarly, the view of P_j (for $j > 2$ and $j \in J$) is defined by $\mathbf{VIEW}_j^{\Pi_1}(\mathbf{A}_1,\ldots,\mathbf{A}_n) = \{\mathbf{A}_j, \omega, (\mathbf{P},\mathbf{Q})\}$ which is indistinguishable from the output $\{\mathbf{A}_j, \omega, (\mathbf{P}',\mathbf{Q}')\}$ computed by $\mathcal{Z}_2$ because $\mathbf{P}', \mathbf{Q}'$ are sampled uniformly from all matrices of size $n \times n$ as matrices $\mathbf{P}, \mathbf{Q}$ are. Thus, (2) holds, as well.

□

Second sub-protocol. Π_2 is quite similar to Π_1 and is described in Figure 4. Basically, protocol Π_2 privately computes $\mathcal{I}_{\mathsf{LSE}_2}$ in the semi-honest model, where ideal functionality $\mathcal{I}_{\mathsf{LSE}_2}$, upon input $\mathbf{b}_i$ of client P_i, outputs to client P_1 the value $\widehat{\mathbf{b}} := \mathbf{P}(\mathbf{b}_1 + \ldots + \mathbf{b}_n)$, with $\mathbf{P} \xleftarrow{\$} \mathbb{F}^{n\times n}$. All other clients $P_2,\ldots,P_n$ receive the empty string.

Theorem A.2 (Privacy of protocol Π_2) *Assuming the clients $P_1,\ldots,P_n$ are semi-honest, and P_1 does not collude, the protocol Π_2 enables $P_1,\ldots,P_n$ to privately compute functionality $\mathcal{I}_{\mathsf{LSE}_2}$.*

The proof goes along the lines of the proof of Theorem A.1 and is therefore omitted.

The final protocol. Given both sub-protocols Π_1 and Π_2, we build protocol Π_{3PI} for 3PI as follows.

1. For $i \in [n]$, client P_i computes $\mathbf{A}_i$ and $\mathbf{b}_i$ as

$$\mathbf{A}_i = \begin{pmatrix} & & \mathbf{0} & & \\ 1 & x_i & x_i^2 & \ldots & x_i^{n-1} \\ & & \mathbf{0} & & \end{pmatrix} \qquad \mathbf{b}_i = \begin{pmatrix} \mathbf{0} \\ y_i \\ \mathbf{0} \end{pmatrix}. \tag{3}$$

2. Clients $P_1,\ldots,P_n$ execute protocol Π_1 such that only P_1 privately computes $\widehat{\mathbf{A}} = \mathbf{P}(\mathbf{A}_1 + \ldots + \mathbf{A}_n)\mathbf{Q}$. Clients $P_2,\ldots,P_n$ have no knowledge about $\widehat{\mathbf{A}}$.
3. Clients $P_1,\ldots,P_n$ execute protocol Π_2 such that only P_1 privately computes $\widehat{\mathbf{b}} = \mathbf{P}(\mathbf{b}_1 + \ldots + \mathbf{b}_n)$. Clients $P_2,\ldots,P_n$ have no knowledge about $\widehat{\mathbf{b}}$. Here, P_2 chooses the same matrix $\mathbf{P}$ as in the previous step.
4. P_1 solves the linear equation $\widehat{\mathbf{A}} \cdot \widehat{\mathbf{c}} = \widehat{\mathbf{b}}$. If a solution $\widehat{\mathbf{c}}$ exists, hands it over to client P_2. Otherwise, the protocol is aborted.

[5] We are treating here the underlying OT functionality as ideal; more formally one could transform any distinguisher into an adversary breaking OT.

Protocol Π_2

Steps performed by clients $P_1, \ldots, P_n$. Each P_i holds a vector $\mathbf{b}_i \in \mathbb{F}^n$.
Parameters p, m are chosen such that $\log(p)m = O(k)$.

1. Clients P_1 and P_2 decompose $\mathbf{b}_1$ (resp. $\mathbf{b}_2$) by sampling random vectors $\mathbf{x}_1^{(1)}, \ldots, \mathbf{x}_m^{(1)}$ satisfying $\mathbf{b}_1 = \mathbf{x}_1^{(1)} + \ldots + \mathbf{x}_m^{(1)}$ (resp. $\mathbf{b}_2 = \mathbf{x}_1^{(2)} + \ldots + \mathbf{x}_m^{(2)}$).
2. For each $j = 1, \ldots, m$ client P_1 and P_2 perform the following sub-steps:
   (a) P_1 sends $(\mathbf{h}_1, \ldots, \mathbf{h}_p)$ to client P_2 where $\mathbf{h}_l = \mathbf{x}_j^{(1)}$ for a secretly chosen index $l \in [p]$ and $\mathbf{h}_i$ (with $i \neq l$) sampled uniformly.
   (b) P_2 chooses random matrices $\mathbf{P}$ and computes $\mathbf{P}(\mathbf{h}_i + \mathbf{x}_j^{(2)}) + \mathbf{r}_j$ for each $i = 1, \ldots, p$, where $\mathbf{r}_j$ is a random vector.
   (c) P_1 executes 1-out-of-p OT and learns $\mathbf{P}(\mathbf{h}_l + \mathbf{x}_j^{(2)}) + \mathbf{r}_j = \mathbf{P}(\mathbf{x}_j^{(1)} + \mathbf{x}_j^{(2)}) + \mathbf{r}_j$.
3. Client P_2 sends to clients P_i (with $i = 3, \ldots, n$) the matrix $\mathbf{P}$, and $\sum_{j=1}^{m} \mathbf{r}_j$ to P_1.
4. Clients P_i with $i = 3, \ldots, n$ send $\mathbf{P}\mathbf{b}_i$ to client P_1.
5. Client P_1, after receiving all values, computes:

$$\begin{aligned} \widehat{\mathbf{b}} &= \sum_{j=1}^{m} (\mathbf{P}(\mathbf{x}_j^{(1)} + \mathbf{x}_j^{(2)}) + \mathbf{r}_j) - \sum_{j=1}^{m} \mathbf{r}_j + \mathbf{P}\mathbf{b}_3 + \ldots + \mathbf{P}\mathbf{b}_n \\ &= \mathbf{P}(\mathbf{b}_1 + \mathbf{b}_2) + \mathbf{P}\mathbf{b}_3 + \ldots + \mathbf{P}\mathbf{b}_n \\ &= \mathbf{P}(\mathbf{b}_1 + \mathbf{b}_2 + \mathbf{b}_3 + \ldots + \mathbf{b}_n). \end{aligned}$$

Figure 4: Description of protocol Π_2

5. P_2 computes $\mathbf{c} = \mathbf{Q} \cdot \widehat{\mathbf{c}}$ and broadcast vector $\mathbf{c}$ to all clients $P_1, P_3, \ldots, P_n$.
6. The vector $\mathbf{c}$ contains the coefficients of the sought-after polynomial $c(\cdot)$.

Theorem A.3 (Privacy of protocol Π_{3PI}) *Assuming clients $P_1, \ldots, P_n$ are semi-honest, and P_1 does not collude, the protocol Π_{3PI} enables $P_1, \ldots, P_n$ to privately compute functionality $\mathcal{I}_{\mathsf{3PI}}$, i.e., it allows clients P_i with input $(x_i, y_i) \in \mathbb{F}$ to privately compute polynomial $c(\cdot)$ such that $c(x_i) = y_i$ for all $i \in [n]$.*

We need to provide simulators $\mathcal{Z}_1, \mathcal{Z}_2$ whose output is computationally indistinguishable from the view of clients P_1 and $P_2, \ldots, P_n$ in a real execution of protocol Π_{3PI}. This is straightforward through the simulators of the sub-protocols Π_1 and Π_2. Hence, we omit the formal proof here.

Efficiency. When estimating computational and communication costs, we must take into consideration that some clients assume specific roles (e.g., P_1 as initiator and P_2 as assembler). Table 1 provides an overview for sub-protocol Π_1. Protocol Π_2 has similar complexity with the exception that computations are performed on vectors of size n rather than on matrices of size n^2. Regardless, all operations are performed in the finite field $\mathbb{F}$.

Note also that clients P_1 and P_2 execute two instances of 1-out-of-p OT [9]. Choosing $p = 2$ and $m = O(k)$, where k denotes the security parameter, allows to instantiate our protocol with the usage of the well-investigated 1-out-of-2 OT [37]. Efficient constructions for OT can be found, for instance, in [13, 36, 17]. Note that our construction's security holds against semi-honest adversaries. For this reason we merely need an OT scheme secure against such adversaries.

Client P_1 solves also one linear equation, and P_2 executes one multiplication before the solution vector is broadcasted to every client. The overall complexity for P_1 is $O(n + k)$, and $O(k)$ for P_2 while being constant for clients $P_3, \ldots, P_n$. Thus, we have $O(n^3 + n^2 k)$ in terms of computational complexity since operations are performed on either matrices of size n^2 or vectors of size n. Efficiency can be improved by exploiting the fact that matrices $\mathbf{A}_i$'s are sparse. We do not elaborate on this further.

Clients	Download	Sending	Additions	Multiplications
P_1	$n+1$	$m \cdot p$	$m+n-1$	0
P_2	$m \cdot p$	$1+2$ (broadcast)	$(2p+1)+m$	$2m \cdot p$
$P_3, \ldots, P_n$	2	1	0	2

Table 1: Efficiency estimates for Π_1 w.r.t. matrices of size n^2

We compare the efficiency of our protocol with previous solutions for PPC-LSE. Given the two-party protocols of [16, 46], one can obtain a scheme for the case of n parties by running the underlying two-party solution $\binom{n}{2}$ times; this approach results in a high communication complexity due to the high number of OT executions. A related work [27] addresses PPC-LSE in the multi-party setting in the so-called "commodity-server model". Here, parties derive the solutions to a PPC-LSE instance with the help of a (possibly untrusted) commodity-server. The solution of [27] has a lower communication complexity of $O(t \cdot n)$, where $1 \leq t \leq n-1$ is a bound on the number of parties allowed to collude. (We stress that, in any case, the commodity-server is never allowed to collude so that there must be at least 2 non-colluding parties.)

In contrast, our scheme yields communication complexity $O(n+k)$ and needs to assume a single non-colluding party. Moreover, due to the asymmetric structure of our construction, resources-limited clients may be in favor of playing the roles of $P_3, \ldots, P_n$.

B Proof of Theorem 5.1

Before proceeding to the proof, we recall the syntax and security definitions for fully-homomorphic encryption (FHE), pioneered by Gentry [21] and studied intensively in the last few years [22, 31, 43, 41, 8, 12, 7, 6]. Informally, FHE is an additively and multiplicatively homomorphic encryption scheme that makes it possible to add and multiply encrypted messages without resorting to decryption.

A homomorphic (public key) encryption scheme is a collection of the following algorithms $\mathcal{HE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, defined below.

Key Generation. Upon input a security parameter 1^k, algorithm Gen outputs a secret and public key (sk, pk) and an evaluation key *evk*.

Encryption. Upon input a public key pk and a message μ, algorithm Enc outputs a ciphertext e.

Decryption. Upon input a secret key sk and a ciphertext e, algorithm Dec outputs a message μ.

Evaluation. Upon input an evaluation key *evk*, a function $c : \{0,1\}^* \to \{0,1\}^*$ and a set of n ciphertexts $e_1, \ldots, e_n$, algorithm Eval outputs a ciphertext e_c.

Definition B.1 (CPA security) *A homomorphic scheme $\mathcal{HE}$ is CPA-(t, ϵ)-secure if for any algorithm $\mathcal{A}$ that runs in time t, for $t = t(k)$, it holds that*

$$|Pr(\mathcal{A}(pk, evk, \mathsf{Enc}_{pk}(\mu_0)) = 1) - Pr(\mathcal{A}(pk, evk, \mathsf{Enc}_{pk}(\mu_1)) = 1)| \leq \epsilon,$$

where $(pk, evk, sk) \leftarrow \mathsf{Gen}(1^k)$, and $(\mu_0, \mu_1) \leftarrow \mathcal{A}(1^\lambda, pk)$ is such that $|\mu_0| = |\mu_1|$.

Definition B.2 ($\mathcal{C}$-homomorphism) *Let $\mathcal{C} = \{\mathcal{C}_k\}_{k \in \mathbb{N}}$ be a class of functions (together with their respective representations). A scheme $\mathcal{HE}$ is $\mathcal{C}$-homomorphic if for any sequence of functions $c_k \in \mathcal{C}_k$ and respective inputs $\mu_1, \ldots, \mu_n$, where $n = n(k)$, it holds that*

$$Pr(\mathsf{Dec}_{sk}(\mathsf{Eval}_{evk}(c, e_1, \ldots, e_n)) \neq c(\mu_1, \ldots, \mu_n)) = negl(k),$$

where the probability is taken over the random choice of $(pk, evk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ and $e_i \leftarrow \mathsf{Enc}_{pk}(\mu_i)$.

Note that the standard properties of additive or multiplicative homomorphism, satisfied for instance by RSA, Paillier, or ElGamal, are captured when the class $\mathcal{C}$ contains only addition or multiplication, respectively.

An homomorphic encryption scheme is said to be *compact* if the output length of $\mathsf{Eval}_{evk}(\cdot)$ is bounded by a polynomial in k (regardless of the function c and of the number of inputs). An encryption scheme is fully-homomorphic when it is both compact and homomorphic with respect to the class $\mathcal{C}$ of all arithmetic circuits over a finite field $\mathbb{F}$ (thus both addition and multiplication over $\mathbb{F}$).

A *somewhat* homomorphic encryption (SHE) scheme allows to compute functions $c(\cdot)$ of "low degree" and it is used as a subroutine of FHE (applying a "bootstrapping" or re-linearization technique of [7, 6] to perform an unbounded number of operations). We use SHE in our schemes since it is significantly faster than FHE.

Proof (Proof of Theorem 5.1). Since we only require security for clients' input and output, we only need to deal with corruptions of server S. Intuitively, an adversary $\mathcal{A}$ corrupting S does not learn anything on the value retrieved by P_i because it is encrypted. Consider the following simulator $\mathcal{Z}$, controlling the server S in the ideal world.

1. $\mathcal{Z}$ is given as input $c(\cdot)$, the number of clients n, the security parameter 1^λ and a pair (pk, evk) with $(pk, evk, sk) \leftarrow \mathsf{Gen}(1^k)$. Then, $\mathcal{Z}$ invokes $\mathcal{A}$ on these inputs.

2. $\mathcal{Z}$ chooses a random point $x' \xleftarrow{\$} \mathbb{F}$, computes $e' \leftarrow \mathsf{Enc}_{pk}(x')$, and sends the result to $\mathcal{A}$.

3. $\mathcal{Z}$ receives e^* from $\mathcal{A}$. The simulator checks whether $e^* = \mathsf{Eval}_{evk}(c(\cdot), e')$;[6] if this is the case, it sends $\hat{c}(\cdot) = c(\cdot)$ as input to the trusted party, otherwise it sends $\perp$.

4. Finally, $\mathcal{Z}$ outputs whatever $\mathcal{A}$ outputs.

We now show, via a hybrid argument, that for any $c(\cdot) \in \mathbb{F}[X]$ and any $x \in \mathbb{F}$, and for any auxiliary input z

$$\left\{\mathbf{IDEAL}_{\mathcal{I}_{\mathsf{SPE}},\mathcal{Z}(z)}(k,(c(\cdot),x))\right\}_{k\in\mathbb{N}} \equiv_c \left\{\mathbf{REAL}_{\Pi_{\mathsf{SPE}},\mathcal{A}(z)}(k,(c(\cdot),x))\right\}_{k\in\mathbb{N}}.$$

Hybrid $\mathcal{H}^1_{\mathcal{A}(z)}(k,(c(\cdot),x))$: In the first hybrid experiment, we remove the trusted party and let $\mathcal{Z}^1$ interact directly with P_i. Essentially $\mathcal{Z}^1$ is identical to $\mathcal{Z}$, but it receives the correct input x from P_i. Hence, $\mathcal{Z}^1$ extracts $\hat{c}$ as $\mathcal{Z}$ would do and uses $\hat{c}(x)$ as output of the trusted party. Then $\mathcal{Z}^1$ outputs $\mathcal{Z}$'s output together with $\hat{c}(x)$. Note that this experiment is distributed exactly as an execution in the ideal world, because $\mathcal{Z}^1$ just plays all the roles by itself. Thus,

$$\left\{\mathbf{IDEAL}_{\mathcal{I}_{\mathsf{SPE}},\mathcal{Z}(z)}(k,(c(\cdot),x))\right\}_{k\in\mathbb{N}} \equiv \left\{\mathcal{H}^1_{\mathcal{A}(z)}(k,(c(\cdot),x))\right\}_{k\in\mathbb{N}}.$$

Hybrid $\mathcal{H}^2_{\mathcal{A}(z)}(k,(c(\cdot),x))$: In the second hybrid experiment, we replace $\mathcal{Z}^1$ with $\mathcal{Z}^2$ which computes the ciphertext e using the value $x \in \mathbb{F}$ it receives from P_i rather than a random x'. We now show that any distinguisher able to tell apart the two experiments can be turned into an adversary $\mathcal{B}$ that breaks the semantic security of the encryption scheme. The adversary $\mathcal{B}$ receives as input a pair (pk, evk) and chooses messages $\mu_0 = x$ and $\mu_1 = x'$; let e_b denote the encryption corresponding to μ_b. From this point on, $\mathcal{B}$ acts exactly like $\mathcal{Z}^2$, but it forwards e_b instead of computing the ciphertext e. Clearly, if e_b is an encryption of x, the output generated by $\mathcal{B}$ is distributed exactly as in $\mathcal{H}^2_{\mathcal{A}(z)}$; on the other hand, if e_b is an encryption of x', the output generated by $\mathcal{B}$ is distributed exactly as in $\mathcal{H}^1_{\mathcal{A}(z)}$. Since the encryption scheme is semantically secure, we must conclude that the two experiments are computationally close.

Hybrid $\mathcal{H}^3_{\mathcal{A}(z)}(k,(c(\cdot),x))$: In the third experiment, we define a simulator $\mathcal{Z}^3$ that is identical to $\mathcal{Z}^2$, but computes the output of P_i as P_i would do in a real execution of the protocol. Hence, instead of computing $\hat{c}(x)$ (provided that $e^* = \mathsf{Eval}_{evk}(c(\cdot), e')$), it decrypts e^* (obtaining $d^* = \mathsf{Dec}_{sk}(e^*)$) and outputs d^*.

To conclude the proof, it suffices to observe that the output distribution produced by $\mathcal{Z}^3$ is identical to the one in a real execution of Π_{SPE}. Hence, $\mathbf{REAL}$ and $\mathbf{IDEAL}$ must be computationally close.

It is easy to show that Π_{SPE} hides the polynomial and its input even to a man-in-the-middle adversary since only encrypted values are transmitted. We omit a formal treatment of this case.

□

[6] We stress that the simulator can perform this check only because the function $c(\cdot)$ is not randomized. See also Remark B.3.

Remark B.3 (On $\mathcal{Z}$) *In the above proof the simulator is asked to check whether a given pair of ciphertexts (e', e^*) satisfies $\mathsf{Eval}_{evk}(c(\cdot), e') = e^*$. In order to do so, $\mathcal{Z}$ needs first to compute $e_i = \mathsf{Enc}_{pk}(c_i \cdot x^{i-1})$ (for all $i = 1, \ldots, n$) and finally evaluate $\sum_i e_i = \mathsf{Enc}_{pk}(c(x))$. However, this strategy will work only provided that the result of the homomorphic computation is deterministic, in the sense that it should be possible, for any constant $c_i \in \mathbb{F}$, to compute ciphertext $\mathsf{Enc}_{pk}(c_i \cdot x^{i-1})$ from $\mathsf{Enc}_{pk}(x)$* without encrypting c_i. *(This is the case for instance in Paillier's scheme.) When such a property is not satisfied, the randomness used to encrypt the coefficients needs to be published as a common reference string.*

C Communication-Efficient Encoding of Polynomials

Let $c(X) = c_{n-1}X^{n-1} + \ldots + c_1 X + c_0$ be a polynomial of degree $n-1$ with coefficients $c_0, \ldots, c_{n-1}$ from a field $\mathbb{F}$. For simplicity, assume there exists an element $m \in \mathbb{N}$ such that $m^2 = n-1$ (i.e., $m = \sqrt{n-1}$). Then, the algorithm described in Figure 5, upon input coefficients $c_0, \ldots, c_{n-1}$, outputs polynomials $\zeta_0(\cdot), \ldots, \zeta_m(\cdot)$ each of maximum degree m such that $c(X) = \zeta_m(X) \cdot X^{m \cdot m} + \zeta_{m-1}(X) \cdot X^{m(m-1)} + \ldots + \zeta_1(X) \cdot X^m + \zeta_0(X)$.

Input: Coefficients $c_{n-1}, \ldots, c_0$

1. Compute $m = \sqrt{n-1}$
2. For $i = 0$ to $m-1$ define
$$\zeta_i(X) := c_{im} + c_{im+1} \cdot X + \ldots + c_{(i+1)m-1} \cdot X^{m-1}$$
3. Define $\zeta_m(X) := c_{n-1}$

Output: Polynomials $\zeta_m(X), \ldots, \zeta_0(X)$

Figure 5: Advantageous encoding of polynomial $c(X)$. The encoding algorithm can be easily modified to handle values $n-1$ which do not have a root in $\mathbb{N}$.

The correctness of the encoding algorithm of Figure 5 can be easily verified. We need to show $c_{n-1}X^{n-1} + \ldots + c_1 X + c_0 = \zeta_m(X) \cdot X^{m \cdot m} + \ldots + \zeta_1(X) \cdot X^m + \zeta_0(X)$. We see that $\zeta_i(X) \cdot X^{im} = c_{im}X^{im} + \ldots + c_{(i+1)m-1}X^{(i+1)m-1}$ for all $i = 1, \ldots, m-1$. That is,

$$\begin{aligned} \zeta_i(X) \cdot X^{im} &= (c_{im} + c_{im+1} \cdot X + \ldots + c_{(i+1)m-1} \cdot X^{m-1}) \cdot X^{im} \\ &= c_{im}X^{im} + c_{im+1} \cdot X^{im+1} + \ldots + c_{(i+1)m-1} \cdot X^{(i+1)m-1}. \end{aligned}$$

Now, by adding all (sub)terms we have $c(X) = \sum_{i=0}^{m} \zeta_i(X) X^{im}$. Thus, correctness is given.

Back on our construction from Section 5, after handing the clew c (which is a polynomial of degree $n-1$) over to the server, the server encodes c as described above to enhance the recovery process. Whenever a client ask for its file, say mapped by pre-image x, the server simply evaluates all polynomials $\zeta_m, \ldots, \zeta_0$ given the ciphertext of all powers of the input up to degree $m = \lceil \sqrt{n-1} \rceil$. Note that it is not necessary to apply multiplications over encrypted values, so that any additive homomorphic encryptions scheme will work. All encrypted evaluated polynomials are then transmitted to the client who decrypts m ciphertexts and evaluate $c(x)$ accordingly, given the values $\zeta_0(x), \ldots, \zeta_m(x)$.

D Extension to the UC-Model

The protocols presented in the previous sections are analyzed in the standalone setting where only sequential composition is allowed. In the context of universally composable (UC) security, protocols are analyzed in a more general setting where

sub-protocols are run concurrently with other secure and insecure protocols. The composition theorem of [10] ensures that a protocol which is UC maintains its security properties even in such a general scenario.

We briefly explain how to analyze our constructions to the UC-model. The definition of universal composability is formalized by considering an additional adversarial entity called the environment $\mathcal{E}$. This environment generates the inputs to all parties, reads all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. A protocol Π securely computes an ideal functionality $\mathcal{I}$ in this framework if, for any adversary $\mathcal{A}$ that interacts with the parties running the protocol, there exists an ideal adversary $\mathcal{Z}$ that interacts with the trusted party, so that no environment $\mathcal{E}$ can distinguish the case that it is interacting with $\mathcal{A}$ and the parties running Π, or with $\mathcal{Z}$ in the ideal world with a trusted party. In this case, we say that Π UC-realizes $\mathcal{I}$. We remark that secure two-party computation of most interesting functionalities in this model requires an additional trusted setup assumption such as a common reference string.

We can easily extend Theorem 5.2 to prove that our main construction from Section 5 UC-realizes $\mathcal{I}_{\mathsf{ESS}}$ in the presence of active, static, $(\mathcal{M}, k)$-limited adversaries. In the proof, the communication with the environment can be handled as follows: Every input value that $\mathcal{Z}$ receives from $\mathcal{E}$ is written on $\mathcal{A}$'s input tape (as if coming from $\mathcal{A}$'s environment). Similarly, every output value written by $\mathcal{A}$ on its output tape is copied to $\mathcal{Z}$'s own output tape (to be read by $\mathcal{Z}$'s environment $\mathcal{E}$). Apart from this, the same analysis as in the current proof will work, because all the interactions between $\mathcal{Z}$ and $\mathcal{Z}_{\mathsf{3PI}}$ and $\mathcal{Z}_{\mathsf{SPE}}$ are straight-line (i.e., $\mathcal{Z}$ does not use rewinding). One exception is the rewinding extractor $\mathsf{E}_{\mathsf{P}^*}$ which is invoked in the proof of Claim 5.3. However, the extractor is called only internally during the reduction to semantic security and is not directly invoked by the simulator $\mathcal{Z}$; so, this rewinding procedure does not affect the proof in the UC-model. Note also that in protocol Π_{SPE} we use a PKI to validate P_i's public key (see Section 5.2). In case we do not do this, P_i needs to transmit the value pk and prove in zero-knowledge it knows the corresponding secret key sk. Then, $\mathcal{Z}_{\mathsf{SPE}}$ needs to use rewinding in order to extract the secret key and maintain the simulation. In such a case, to prove UC-security, we would need a UC zero-knowledge argument of knowledge for the key generation procedure in question.