Entangled Cloud Storage

Giuseppe Ateniese[1], Özgür Dagdelen[2], Ivan Damgård[3], and Daniele Venturi[4]

[1]*Sapienza University of Rome and Johns Hopkins University*
[2]*Technische Universität Darmstadt*
[3]*Aarhus University*
[4]*Sapienza University of Rome*

Abstract

Entangled cloud storage enables a set of clients to "entangle" their files into a single *clew* to be stored by a (potentially malicious) cloud provider. The entanglement makes it impossible to modify or delete significant part of the clew without affecting *all* files encoded in the clew. A clew keeps the files in it private but still lets each client recover his own data by interacting with the cloud provider; no cooperation from other clients is needed. At the same time, the cloud provider is discouraged from altering or overwriting any significant part of the clew 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 modify or delete their data illegitimately. Current solutions, e.g., based on Provable Data Possession and Proof of Retrievability, require the server to be challenged regularly to provide evidence that the clients' files are stored *at a given time*. Entangled cloud storage provides an alternative approach where any single client operates implicitly on behalf of all others, i.e., as long as one client's files are intact, the entire remote database continues to be safe and unblemished.

Contents

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 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 rarely-accessed files are altered? What if the storage service provider experiences Byzantine failures and tries to hide data errors from the clients? Can we detect these changes and catch a misbehaving provider?

Existing 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 [17] (building on a prior work by Naor and Rothblum [21]). They were later extended in several ways in [24, 11, 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.

A novel approach. The main drawback of proof-of-storage schemes is that a successful run of an audit provides evidence about the integrity of a remote file *only at a given time*. As a consequence, all users must challenge the storage server regularly to make sure their files are still intact. In this paper, we propose a novel solution to the problem of cloud storage. The main idea is to make altering or deleting files extremely inconvenient for the cloud provider. To achieve this feature, we consider a setting where many 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. 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). The entanglement procedure is distributed across the clients and a single client can retrieve its own file by interacting with the server alone, without the need of any trusted entity. We refer to our framework as *entangled storage*.

We believe that our approach is useful for a number of reasons:

1. As in PDP/POR, the cloud provider is strongly discouraged from misbehaving. In addition, any single client implicitly operates on behalf of all clients in the sense that the client, while inspecting the soundness of his own files, implicitly checks for the integrity of the files of all other users. Thus, the disincentive to misbehave in entangled storage is stronger than in PDP/POR since a dishonest cloud provider will likely be prosecuted by all users rather than only by the affected ones.

2. From a practical perspective, users within our framework do not have to keep constantly querying the cloud provider with proof-of-storage challenges as in PDP/POR schemes. No user has to explicitly request a file to check for its integrity. As long as other clients are able to retrieve their own files, everybody else in the system will be ensured that their files are intact.

3. Whenever a client fails to recover a file, it could be because the server deleted or modified it or is simply refusing to hand it over. A dishonest client could in principle frame the cloud provider by falsely claiming his files are unrecoverable. Fortunately, though, any other client can establish the truth and expose the villain by successfully retrieving any of his own files. This property cannot be realized within existing PDP/POR schemes where the cloud provider is always susceptible to blackmail.

The advantages described above come at a price: Users must coordinate and run an expensive procedure to build the entanglement. That said, the entangled storage framework can also be used by a *single* user to entangle his own files and then outsource the related clew to the cloud. This way, no coordination step is required, leading to a significantly more efficient scheme. Quite remarkably, this allows the user to verify that *all* files are still in place by recovering just one of them. For instance, as long as the user downloads regularly accessed files (e.g., family pictures), he can be sure any other files are still intact, even those rarely retrieved (e.g., tax returns).

A note on the terminology. *Entanglement* usually refers to a physical interaction between two particles at the quantum level. Even if separated, the particles are in a quantum superposition until a measurement is made, in which case both particles assume definitive and correlated states. Analogously, in our more modest context, two entangled files are somehow linked together: A file that is intact implies the other must also be intact. Any single change to one file, destroys the other.

1.1 Our Contributions

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

Entangled encodings. 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 consists of an algorithm Encode that 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 the file f_i together with a verification value ξ. 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". The verification value is a fixed function of the encoded string and the randomness.

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 from 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.

We provide a concrete instantiation of an entangled encoding scheme based on polynomials over a finite field $\mathbf{F}$. Here, the encoding of a string f_i is generated by choosing a random pair of elements $(s_i, x_i) \in \mathbf{F}^2$ and defining a point $(x_i, y_i = f_i + s_i)$. The entanglement of $(f_1, \ldots, f_n)$ consists of 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 $\mathbf{F}$ is large enough, this encoding satisfies the all-or-nothing integrity property for a proper choice of the parameters. The latter holds even in case the adversary is computationally unbounded.

Simulation-based security. Next, we propose a simulation-based definition of security for entangled storage in the cloud setting, in the model of universal composability [8] (UC). In the UC 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.

Roughly, the ideal functionality $\mathcal{I}_{\text{ESS}}$ 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. Intuitively, the latter property says that the probability that a cloud provider answers correctly a recovery query for some file is roughly the same for all files which are part of the entanglement: such probability is either one (in case the server did not modify the clew), or negligibly close to zero (in case the entanglement was modified).

We choose to prove security in the UC model as this gives strong composition guarantees. However, some technical difficulties arise as a result. First, we face the problem that if the server is corrupt it may choose to overwrite the encoding we give it with something else, and so we may enter a state where the server's uncertainty about the encoding is so large that no request can be answered. Now, in any simulation based definition, the simulator must clearly know whether we are in a such a state. But since the server is corrupt we do not know how it stores data and therefore it is not clear how the simulator could efficiently compute the server's uncertainty about the encoding. In the UC model it is even impossible because the data could be stored in the state of the environment which is not accessible to the simulator.

We solve this problem by introducing a "memory module" in the form of an ideal functionality, and we store the encoded data only inside this functionality. This means that the encoding can only be accessed via commands we define. In particular, data cannot be taken out of the functionality

and can only be overwritten by issuing an explicit command. This solves the simulator's problem we just mentioned. A corrupt server is allowed, however, to specify how it wants to handle retrieval requests by giving a (possibly adversarial) machine to the functionality, who then will let it execute the retrieval on behalf of the server.

We emphasise that this memory functionality is not something we hope to implement using simpler tools, it should be thought of as a model of an adversary that stores the encoded data only in one particular location and will explicitly overwrite that location if it wants to use it for something else.

A protocol realizing $\mathcal{I}_{\text{ESS}}$. Finally we design a protocol implementing our ideal functionality for entangled storage. The scheme relies on the entangled encoding scheme based on polynomials in a finite field $\mathbf{F}$ described above, and on a somewhat homomorphic encryption scheme with message space equal to $\mathbf{F}$. Each client has a file f_i (represented as a field element in $\mathbf{F}$), samples $(s_i, x_i) \leftarrow \mathbf{F}^2$, defines $(x_i, y_i = f_i + s_i)$, and keeps a hash θ_i of the original file. During the "entanglement phase", the clients run a secure protocol for computing the coefficients of the polynomial $c(\cdot)$ of minimum degree interpolating all of (x_i, y_i). This can be done by using standard techniques relying on linear secret sharing (see Appendix A). The polynomial $c(\cdot)$ is stored in the ideal functionality for the memory module, which can be accessed by the server.

Whenever a client wants to recover its own file, it forwards to the server a ciphertext e corresponding to an encryption of x_i. The server returns an encryption of $c(x_i)$, computed trough the ciphertext e and using the homomorphic properties of the encryption scheme, together with a proof that the computation was performed correctly.[1] The client can verify the proof, decrypt the received ciphertext in order to obtain y_i and thus $f_i = y_i - s_i$, and check that the hash value θ_i matches.

Our final protocol is a bit more involved than the above, as clients are not allowed to store the entire (x_i, s_i) (otherwise they could just store the file f_i in the first place); however this can be easily solved by having the client store only the seed σ_i of a pseudo-random generator $G(\cdot)$, and recover (x_i, s_i) as the output of $G(\sigma_i)$. We refer to Section 5 for the details.

1.2 Efficiency Trade-off

In this work, we do not focus on performance optimization and the proposed scheme should be interpreted more as a feasibility result and as the first practical implementation of an entangled storage scheme. In order to improve the performance of our construction, one must make sure that the polynomial, which represents the entanglement, has a low degree. One natural way to achieve this is by limiting the number of users who take part to the entanglement and create several smaller clews. This would offer a clear tradeoff between security and efficiency.

1.3 Alternative Solutions

Entangled storage was implemented through our notion of entangled encoding but, of course, it should be considered whether it can be realized in other ways.

A first natural idea is to upload each file in encrypted form to the server. Whenever a file is retrieved, a proof of retrievability (PoR) for the *entire* set of (encrypted) files is also executed

[1] Note that the latter requires some interaction between the server and the memory functionality, as the polynomial is needed in order to compute the answer.

between the client and the server. We believe such a solution would satisfy our definition of entangled storage. However, there are two impeding drawbacks to consider. First, a PoR scheme requires a redundant encoding of the data, hence the server needs more storage than strictly necessary. Second, the local computation performed by the client in a PoR scheme typically depends on the total size of the remote data. In our scenario, this is not acceptable since it makes the work of the client depend on the total number of clients. In contrast, our entangled encoding has size exactly equal to the encoded data (when files are large enough) and the work performed by a client is independent of the number of clients.

A different idea would be to encrypt each file and then upload them to the server in a randomly permuted order, such that each client knows the position of his own file. A client may use private information retrieval (PIR) to retrieve files. This way the server remains oblivious of the relative position of any file, even after several retrievals. At first, this solution may seem good enough to deter the server from erasing files. But note that the server could correctly estimate any file positions with non-negligible probability, possibly with the help of malicious clients. Most importantly, this proposal based on PIR does not actually satisfy our definition. Indeed, the server may end up excluding *some* clients while allowing others to still retrieve their files. Entangled storage mandates that *no* client can retrieve data whenever a significant part of it is erased.

1.4 Other Related Work

The approach of *data entanglement* was originally proposed by Aspnes *et al.* [1]. 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.[2] Data entanglement also appears in the context of censorship-resistant publishing systems; see, e.g., Dagster [26] and Tangler [27].

The notion of all-or-nothing integrity is inspired by the all-or-nothing transform introduced by Rivest et al. [23], and later generalized in [10]. 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

2.1 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

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

function ϵ such that $|\mathbb{P}(\mathcal{A}(X)=1) - \mathbb{P}(\mathcal{A}(Y)=1)| \leq \epsilon(k)$. 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 as $\mathbb{SD}(X,Y) = \sum_a |\mathbb{P}(X=a) - \mathbb{P}(Y=a)|$. The min-entropy of a random variable X is $\mathbb{H}_\infty(X) = -\log \max_x \mathbb{P}(X=x)$.

2.2 The UC Framework

We briefly review the framework of universal composability (UC) [8]. 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 $(x_1,\ldots,x_n) \mapsto (y_1,\ldots,y_n)$.

Consider a protocol Π run by a set of parties $P_1,\ldots,P_n$ (where each party P_i holds input x_i), for computing $\phi(x_1,\ldots,x_n)$. In order to define security of Π, we introduce an ideal process involving an incorruptible "trusted party" that is programmed to capture the desired requirements from the task at hand. Roughly, we say that a protocol for ϕ is secure if it "emulates" the ideal process. Details follow.

The real execution. We represent a protocol as a system of interactive Turing machines (ITMs), where each ITM represents the program to be run within a different party. Adversarial entities are also modeled as ITMs; we concentrate on a non-uniform complexity model where the adversaries have an arbitrary additional input, or an "advice". We consider the computational environment where a protocol is run as asynchronous, without guaranteed delivery of messages. The communication is public (i.e., all messages can be seen by the adversary) but ideally authenticated (i.e., messages sent by honest parties cannot be modified by the adversary).

The process of executing protocol Π (run by parties $P_1,\ldots,P_n$) with some adversary $\mathcal{A}$ and an environment machine $\mathcal{Z}$ with input z is defined as follows. All parties have a security parameter $k \in \mathbb{N}$ and are polynomial in k. The execution consists of a sequence of activations, where in each activation a single participant (either $\mathcal{Z}$, $\mathcal{A}$ or P_i) is activated. The activated participant reads information from its input and incoming communication tapes, executes its code, and possibly writes information on its outgoing communication tapes and output tapes. In addition, the environment can write information on the input tapes of the parties, and read their output tapes. The adversary can read messages on the outgoing message tapes of the parties and deliver them by copying them to the incoming message tapes of the recipient parties. The adversary can also corrupt parties, with the usual consequences that it learns the internal information known to the corrupt party and that, from now on, it controls that party.

Let $\mathbf{REAL}_{\Pi,\mathcal{A}(z),\mathcal{Z}}(k,(x_1,\ldots,x_n))$ denote the random variable corresponding to the output of environment $\mathcal{Z}$ when interacting with adversary $\mathcal{A}$ and parties running protocol Π (holding inputs $x_1,\ldots,x_n$), on input security parameter k, advice z and uniformly chosen random coins for all entities.

The ideal execution. The ideal process for the computation of ϕ involves a set of dummy parties $P_1,\ldots,P_n$, an ideal adversary $\mathcal{SIM}$ (a.k.a. the simulator), an environment machine $\mathcal{Z}$ with input z, and an ideal functionality $\mathcal{I}$ (also modeled as an ITM). The ideal functionality simply receives all inputs by $P_1,\ldots,P_n$ and returns to the parties their respective outputs $\phi_i(x_1,\ldots,x_n)$. The ideal adversary $\mathcal{SIM}$ proceeds as in the real execution, except that it has no access to the contents of the messages sent between $\mathcal{I}$ and the parties. In particular, $\mathcal{SIM}$ is responsible for delivering

messages from $\mathcal{I}$ to the parties. It can also corrupt parties, learn the information they know, and control their future activities.

Let $\mathbf{IDEAL}_{\mathcal{I},\mathcal{SIM}(z),\mathcal{Z}}(k,(x_1,\ldots,x_n))$ denote the random variable corresponding to the output of environment $\mathcal{Z}$ when interacting with adversary $\mathcal{SIM}$, dummy parties $P_1,\ldots,P_n$ (holding inputs $x_1,\ldots,x_n$), and ideal functionality $\mathcal{I}$, on input security parameter k, advice z and uniformly chosen random coins for all entities.

Securely realizing an ideal functionality. We can now define universally composable (UC) security, following [8].

Definition 1 *Let $n \in \mathbb{N}$. Let $\mathcal{I}$ be an ideal functionality for $\phi : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ and let Π be an n-party protocol. We say that Π securely realizes $\mathcal{I}$ if for any adversary $\mathcal{A}$ there exists an ideal adversary $\mathcal{SIM}$ such that for any environment $\mathcal{Z}$, any tuple of inputs $(x_1,\ldots,x_n)$, we have*

$$\left\{\mathbf{IDEAL}_{\mathcal{I},\mathcal{SIM}(z),\mathcal{Z}}(k,(x_1,\ldots,x_n))\right\}_{k\in\mathbb{N},z\in\{0,1\}^*} \approx \left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),\mathcal{Z}}(k,(x_1,\ldots,x_n))\right\}_{k\in\mathbb{N},z\in\{0,1\}^*}.$$

In this paper we only allow static corruptions, that is, adversaries determine the parties to corrupt at the beginning of the protocol execution. The adversary is called *passive* if it follows faithfully the protocol specifications but can save intermediate computations; on the other hand an *active* adversary can behave arbitrarily during a protocol execution. Security of a protocol is sometimes defined with respect to an *adversary structure* Δ, i.e., a monotone[3] set of subsets of the players, where the adversary may corrupt the players of one set in Δ. When this is the case, we say that Π Δ-securely realizes a given functionality.

The composition theorem. The above notion of security allows a modular design of protocols, where security of each protocol is preserved regardless of the environment where that protocol is executed. In order to state the composition theorem, we sketch the so-called $\mathcal{I}$-hybrid model, where a real-life protocol is augmented with an ideal functionality. This model is identical to the above real execution, with the following additions. On top of sending messages to each other, the parties may send messages to and receive messages from an unbounded number of copies of $\mathcal{I}$. (Each copy is identified via a unique session identifier, chosen by the protocol run by the parties.)

The communication between the parties and each one of the copies of $\mathcal{I}$ mimics the ideal process. That is, once a party sends a message to some copy of $\mathcal{I}$, that copy is immediately activated and reads that message of the party's tape. Furthermore, although the adversary in the hybrid model is responsible for delivering the messages from the copies of $\mathcal{I}$ to the parties, it does not have access to the contents of these messages. It is stressed that the environment does not have direct access to the copies of $\mathcal{I}$.

Let Π be a protocol in the $\mathcal{I}'$-hybrid model and let Π' be a protocol UC-realizing $\mathcal{I}'$. Consider the composed protocol $\Pi^{\Pi'}$, where each call to the ideal functionality $\mathcal{I}'$ is replaced with an execution of protocol Π'.

Theorem 1 ([8]) *Let $\mathcal{I}$, $\mathcal{I}'$ be ideal functionalities. Let Π be an n-party protocol that securely realizes $\mathcal{I}$ in the $\mathcal{I}'$-hybrid model and let Π' be an n-party protocol that securely realizes $\mathcal{I}'$. Then protocol $\Pi^{\Pi'}$ securely realizes $\mathcal{I}$.*

[3] An adversary structure is monotone in the sense of being closed with respect to taking subsets.

2.3 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 $(\mathcal{P}, \mathcal{V})$ for $L \in NP$ 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 $(\mathcal{P}, \mathcal{V})$ is only polylogarithmic in the instance and witness sizes; (ii) (completeness) for any $x \in L$ we have that $(\mathcal{P}(w), \mathcal{V})(x)$ outputs 1 with overwhelming probability; (iii) (argument of knowledge) for any $x \notin L$ and any computationally bounded prover $\mathcal{P}^*$ such that $(\mathcal{P}^*, \mathcal{V})(x)$ outputs 1 there exists a polynomial time extractor $\mathcal{EXT}_{\mathcal{P}^*}$ outputting a witness w that satisfies $(x,w) \in R$ with overwhelming probability. See for instance [28, 5].

Succinct interactive argument systems for NP exists in 4 rounds based on the PCP theorem, under the assumption that collision-resistant function ensembles exists [18, 28]. Succinct non-interactive argument systems, also called SNARGs, are impossible under any falsifiable cryptographic assumption [15] but are known to exists in the random-oracle model [19] or under non-falsifiable cryptographic assumptions [5].

2.4 Somewhat Homomorphic Encryption

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 2 (CPA security) *A homomorphic scheme $\mathcal{HE}$ is CPA-secure if for any probabilistic polynomial time algorithm $\mathcal{A}$ it holds that*

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

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

Sometimes we also refer to the "real" or "random" variant of CPA-security, where $\mathcal{A}$ has to distinguish the encryption of a known message μ from the encryption of a random unrelated message μ'. The two notions are equivalent up-to a constant factor in security.

Definition 3 ($\mathcal{C}$-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 $(pk, evk, sk) \leftarrow \mathsf{Gen}(1^k)$ 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 $\mathbf{F}$ (thus both addition and multiplication over $\mathbf{F}$).

A *somewhat* homomorphic encryption (SHE) scheme allows to compute functions $c(\cdot)$ of "low degree" and it is used as a subroutine of fully homomorphic encryption [14] (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.

2.5 Collision Resistant Hashing

We recall what it means for a family of hash functions to be collision resistant. Let $\ell, \ell' : \mathbb{N} \to \mathbb{N}$ be such that $\ell(k) > \ell'(k)$, and let $I \subseteq \{0,1\}^*$. A function family $\{H_\iota\}_{\iota \in I}$ is called a collision-resistant hash family if the following holds.

- There exists a probabilistic polynomial time algorithm IGen that on input 1^k outputs $\iota \in I$, indexing a function H_ι mapping from $\ell(k)$ bits to $\ell'(k)$ bits.
- There exists a deterministic polynomial time algorithm that on input $x \in \{0,1\}^\ell$ and $\iota \in I$, outputs $H_\iota(x)$.
- For all probabilistic polynomial time adversaries $\mathcal{B}$ we have that

$$\mathbb{P}\left(H_\iota(x) = H_\iota(x') : (x, x') \leftarrow \mathcal{B}(1^k, \iota); \iota \leftarrow \mathsf{IGen}(1^k)\right) \leq negl(k),$$

  where the probability is taken over the coin tosses of IGen and of $\mathcal{B}$.

2.6 Pseudorandom Generators

We recall the definition of a pseudorandom generator. Let $G : \{0,1\}^k \to \{0,1\}^{\ell(k)}$ be a deterministic function, where $\ell(k) > k$. We say that G is a secure PRG if there exists a polynomial time algorithm that given $\sigma \in \{0,1\}^k$ outputs $G(\sigma)$, and moreover for all probabilistic polynomial time adversaries $\mathcal{B}$ we have:

$$\mathbb{P}\left(\mathcal{B}(G(\sigma)) = 1 : \sigma \leftarrow \{0,1\}^k\right) - \mathbb{P}\left(\mathcal{B}(U_{\ell(k)}) = 1\right) \leq negl(k),$$

where $U_{\ell(k)}$ is uniform over $\{0,1\}^{\ell(k)}$.

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.[4] The following

[4] In case files have different lengths, they can be simply padded to some pre-fixed value ℓ (which is a parameter of the scheme).

definition captures an entangled encoding scheme formally.

Definition 4 (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* $f_i \in \mathcal{F}$ *and a* verification value ξ*. This value must be a fixed function* $\xi(f_i, r_i)$ *of the file and the randomness.*

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 $(f_i, \xi(f_i, r_i)) = \mathsf{Decode}(\mathsf{Encode}(f_1, \ldots, f_n, r_1, \ldots, r_n), r_i, i)$ for all $i \in [n]$.

3.1 Security Properties

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, but note that of course the distribution of R_i will be fixed by the encoding scheme. 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 C be the encoding corresponding to all elements, i.e., $C = \mathsf{Encode}(F_1, \ldots, F_n, R_1, \ldots, R_n)$. 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 all $u \in U$ and all $c \in C$, the distribution $\mathcal{D}_{V|U}$ of the random variable V when given $U = u$ is statistically close to the distribution $\mathcal{D}_{V|UC}$ of the random variable V when given $(U = u, C = c)$, i.e., $\mathbb{SD}(\mathcal{D}_{V|U}, \mathcal{D'}_{V|UC}) \leq negl(k)$.

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 $\mathbb{H}_\infty(C|C'_U = c') \geq \alpha$ implies that for each i, the adversary wins at position i with probability at most β. In particular, in order to win, the adversary's function must output both the correct file and verification value.

Definition 5 ((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 for all (possibly unbounded) adversaries* $\mathcal{A}$*, for all subsets* $U \subset \{(F_j, R_j)\}_{j=1\ldots n}$*, for all (possibly unbounded) functions* $g(\cdot)$ *and for all* $i \in [n] \setminus \{j : (F_j, R_j) \in U\}$*, we have that*

$$\mathbb{P}\left(\mathsf{Decode}_{Adv}(R_i, i) = \mathsf{Decode}(C, R_i, i) : \begin{array}{l}(\mathcal{F}, \mathcal{R}, \mathcal{C}) \leftarrow \mathsf{Setup}(1^k, n, \ell), \\ C = \mathsf{Encode}(F_1, \ldots, F_n; R_1, \ldots, R_n), \\ C'_U = g(C, U), \mathsf{Decode}_{Adv} \leftarrow \mathcal{A}(C'_U)\end{array}\right) \leq \beta,$$

whenever $\mathbb{H}_\infty(C|C'_U = c') \geq \alpha$ *(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.

3.2 A Code based on Polynomials

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 $\mathbf{F}$, 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 $\mathbf{F}$ 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 $\mathbf{F}$. 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 difficult since candidates can only agree pairwise 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 1 *Given a non-constant polynomial* c *of degree at most* n*, the distribution of* $c(R)$*, where* R *is uniform in* $\mathbf{F}$*, has min-entropy at least* $\log|\mathbf{F}| - \log(n)$*.*

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 $\mathbb{P}(c(R) = y) \leq n/|\mathbf{F}|$, 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 the distribution of $c(R)$ is not uniform as we would like, but only close (if $n \ll |\mathbf{F}|$). 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 $\mathcal{F} = \mathbf{F} = GF(2^{max(\ell, 3k+\log n+\log\log n)})$, $\mathcal{R} = \mathbf{F}^2$ and $\mathcal{C} = \mathbf{F}^n$.

Encoding: Given $f_1, \ldots, f_n$ to encode, choose $x_i, s_i \in \mathbf{F}$ uniformly (and independently) at random, and set $r_i = (x_i, s_i)$; in case $x_i = x_j$ for some index $i \neq j$ output a special symbol $\perp$ and abort. Otherwise, define $\mathsf{Encode}(f_1, \ldots, f_n, r_1, \ldots, r_n) = c$ to be 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 $\mathsf{Decode}(c, r_i, i) = \mathsf{Decode}(c, (x_i, s_i), i) = (c(x_i) - s_i,\ c(x_i))$.

It is trivial to see that Decoding outputs the correct file. The verification value is $c(x_i) = f_i + s_i$ thus it is indeed a function of the file and the randomness, as required by the definition. The encoding is also easily seen to be private: In fact, by the uniformly random choice of s_i, given any subset U of $(F_j, R_j)_{j=1\ldots n}$ the encoding C does not reveal any additional information on $V = (F_{-U}, R_{-U})$. 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 constant). However, the latter strategy succeeds with probability exponentially small in $|\mathbf{F}|$. The theorem says that, up to constant factor losses in the exponent, these are the only options open to the adversary.

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

3.3 Proof of Theorem 2

Before coming to the theorem, we need the following lemma.

Lemma 2 *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* $\mathbb{H}_\infty(X_i |\ C'_U = c') \geq \log|\mathbf{F}| - \log n$.

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 $|\mathbf{F}|/n$ 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 $n/|\mathbf{F}|$, 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 $\log|\mathbf{F}| - \log n$.

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 $\log |\mathbf{F}| - \log n$ and hence so does any weighted average. $\square$

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 $(f_i, c(x_i)) = \mathsf{Decode}_{Adv}(x_i, s_i)$ for an arbitrary but fixed $i \in [n]$. We want to bound the adversary's advantage.

In particular, the adversary's function must output the correct value of $c(x_i)$, so we may as well bound the probability ϵ that $g(x_i) = c(x_i)$ for a function g chosen by the adversary, where c is chosen according to $\mathcal{D}$ and x_i has large min-entropy as shown in Lemma 2 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 $\mathbb{P}(\epsilon_c \geq \epsilon/2) \geq \epsilon/2$ 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 $\epsilon 2^{\alpha-1} \leq \sqrt[3]{|\mathbf{F}|/n}$, 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 $\sqrt[3]{|\mathbf{F}|/n}$ 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 $\mathcal{X}_{ij} = \{x \in \mathbf{F} \mid c_i(x) = c_j(x)\}$. 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:

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

where the last inequality follows from Lemma 2. Since we already know that there is a way for the adversary to win with probability $\epsilon/2$, we have $\epsilon/2 \leq \frac{|\mathcal{C}'|^2 n \log n}{|\mathbf{F}|} + \frac{1}{|\mathcal{C}'|}$. 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

In this section we present our model for entangled storage in the cloud setting. At a very intuitive level consider the following natural way to specify an ideal functionality $\mathcal{I}_{\text{ESS}}$ capturing the security properties we want: it will receive data from all players, and will return data to honest players on request. If the server is corrupt it will ask the adversary (the simulator in the ideal process) if a request should be answered (since in real life a corrupt server could just refuse to play). However, if $\mathcal{I}_{\text{ESS}}$ ever gets an "overwrite" command it will refuse to answer any requests. The hope would then be to implement such a functionality using an entangled encoding scheme, as the AONI property ensures that whenever there is enough uncertainty (in the information theoretic sense) about the encoding, a corrupt server cannot design a function that will answer decoding queries correctly.

Functionality $\mathcal{I}_{\text{mem}}$:

The functionality $\mathcal{I}_{\text{mem}}$ is parameterized by the security parameter k, entanglement size n and a sharing scheme $(\mathsf{Share}, \mathsf{Reconstruct})$. The interaction with an ordered set of (possibly corrupt) clients $P_1, \ldots, P_n$, a (possibly corrupt) server S, an (incorruptible) Observer $\mathcal{OBS}$, and ideal adversary $\mathcal{SIM}$ is enabled via the following queries:

- On input $(\texttt{Store}, i, s_i)$ from P_i (where $s_i \in \{0,1\}^*$), record (i, s_i). Ignore any subsequent query $(\texttt{Store}, i, *, *)$ from P_i. If there are already n recorded tuples, send `Done` to all clients, to S and to $\mathcal{SIM}$. Mark session as `Active`; define $c = \mathsf{Reconstruct}(s_1, \ldots, s_n)$, and $\mathcal{K} = \emptyset$.
- On input $(\texttt{Overwrite}, \{i_j\}_{j\in[t]})$ from $\mathcal{SIM}$ (where $t \leq \log|\mathcal{C}|$), check that the session is `Active` (if not ignore the input). Set $c[i_j] = 0$ for all $j \in [t]$ and $\mathcal{K} \leftarrow \mathcal{K} \cup \{i_j\}_{j\in[t]}$. If $|\mathcal{K}| \geq k$, send $(\texttt{Overwrite})$ to $\mathcal{OBS}$.
- On input $(\texttt{Read}, \mathcal{M}, i)$ from S or $\mathcal{SIM}$ (where $\mathcal{M}$ is a read-only Turing machine and $i \in [n]$), check that the session is `Active` and either P_i or S are honest (if not ignore the input). Send $\mathcal{M}(c)$ to P_i.

Figure 1: The basic memory functionality $\mathcal{I}_{\text{mem}}$

However, technical difficulties arise due to the fact that the simulator should know when the uncertainty about the encoding is high enough. This requires the simulator to estimate the adversary's uncertainty about the encoding, which is not necessarily easy to compute (e.g., the adversary could store the encoding in some unintelligible format). To deal with this problem, we introduce another functionality (which we call the memory functionality, see Section 4.1) modeling how data is stored in the cloud, and how the server can access the stored data.

A second difficulty is that simply specifying the functionality $\mathcal{I}_{\text{ESS}}$ as sketched above is not sufficient to capture the security we want. The problem is that even a "bad" protocol where data from different players are stored separately (no entanglement) can be shown to implement $\mathcal{I}_{\text{ESS}}$. The issue is that if the adversary overwrites data from just one player, say P_1, the simulator can "cheat" and not send an overwrite command to $\mathcal{I}_{\text{ESS}}$. Later, if P_1 requests data, the simulator can instruct $\mathcal{I}_{\text{ESS}}$ to not answer the request. Now the request it fails in both the real and in the ideal process, and everything seems fine to the environment.

We therefore need to add something that will force a simulator to send overwrite as soon as too much data is overwritten. We do this by introducing an additional incorruptible player called the *Observer*. In the real process, when the memory functionality has been asked to overwrite too much data, it will send "overwrite" to the Observer, who outputs this (to the environment). We also augment $\mathcal{I}_{\text{ESS}}$ such that when it receives an overwrite command, it will send "overwrite" to the Observer. Now note that in the real process, when too much data is overwritten, the environment will always receive "overwrite" from the Observer. Hence whenever the ideal process gets into a similar state, the simulator must send an overwrite command to $\mathcal{I}_{\text{ESS}}$: this is the only way to make the Observer output "overwrite" and if he does not, the environment can trivially distinguish.

The functionality for data entanglement in the cloud is presented in Section 4.2. We emphasize that a real application of our protocol does not need to include an Observer (as he takes no active

Functionality $\mathcal{I}^*_{\text{mem}}$:

The functionality $\mathcal{I}_{\text{mem}}$ is parameterized by the security parameter k, entanglement size n and an entangled encoding scheme $(\mathsf{Encode}, \mathsf{Decode})$ with file space $\mathcal{F}$, randomness space $\mathcal{R}$, and entanglement space $\mathcal{C}$. The interaction with an ordered set of (possibly corrupt) clients $P_1, \ldots, P_n$, a (possibly corrupt) server S, an (incorruptible) Observer $\mathcal{OBS}$, and ideal adversary $\mathcal{SIM}$ is enabled via the following queries:

- On input $(\texttt{Store}, i, f_i, r_i)$ from P_i (where $f_i \in \mathcal{F}$ and $r_i \in \mathcal{R}$), record (i, f_i, r_i). Ignore any subsequent query $(\texttt{Store}, i, *, *)$ from P_i. If there are already n recorded tuples, send `Done` to all clients, to S and to $\mathcal{SIM}$. Mark session as `Active`; define $c \leftarrow \mathsf{Encode}(f_1, \ldots, f_n, r_1, \ldots, r_n)$, and $\mathcal{K} = \emptyset$.
- On input $(\texttt{Overwrite}, \{i_j\}_{j\in[t]})$ from $\mathcal{SIM}$ (where $t \leq \log|\mathcal{C}|$), check that the session is `Active` (if not ignore the input). Set $c[i_j] = 0$ for all $j \in [t]$ and $\mathcal{K} \leftarrow \mathcal{K} \cup \{i_j\}_{j\in[t]}$. If $|\mathcal{K}| \geq k$, send $(\texttt{Overwrite})$ to $\mathcal{OBS}$.
- On input $(\texttt{Read}, \mathcal{M}, i)$ from S or $\mathcal{SIM}$ (where $\mathcal{M}$ is a read-only Turing machine and $i \in [n]$), check that the session is `Active` and either P_i or S are honest (if not ignore the input). Send $\mathcal{M}(c)$ to P_i.

Figure 2: The augmented memory functionality $\mathcal{I}^*_{\text{mem}}$

part in the protocol). He is only there to make our security definition capture exactly what we want.

4.1 The Memory Functionality

The memory functionality $\mathcal{I}_{\text{mem}}$ is given in Figure 1 and specifies how data is stored in the cloud and how a (possibly corrupt) server can access this data. As explained above, we cannot give the server direct access to the data since then he might store it elsewhere encoded in some form we cannot recognize, and then he may have full information on the data even if he overwrites the original memory.

Roughly $\mathcal{I}_{\text{mem}}$ allows a set of parties to store a piece of information in the cloud. For technical reasons this information is interpreted in the form of "shares" that are then combined inside the functionality to define the actual storage $c \in \mathcal{C}$.[5] We use the term "share" informally here, and refer the reader to Appendix A for a formal definition.

The simulator can access the data stored inside the memory functionality in two ways: (i) by computing any function of the data and forwarding the output to some party; (ii) by explicitly forgetting (part of) the data stored inside the functionality. Looking ahead, the first type of interaction will allow the server to answer recovery queries from the clients.[6] The second type of interaction corresponds to the fact that overwriting data is an explicit action by the server. This

[5]This is because the real protocol allows clients to securely compute "shares" of the entanglement, but they are not allowed to recover the entanglement itself from the shares as otherwise malicious clients would learn the encoding (and so would do a colluding malicious server, making the memory functionality useless).

[6]The function above is specified via a Turing machine; this Turing machine has to be read-only, as otherwise the server could overwrite data without explicitly calling "overwrite".

way, the adversarial behavior in the retrieval protocol is decided based only on what $\mathcal{I}_{\text{mem}}$ stores, and data can only be forgotten by explicit command from the adversary.

As explained at the beginning of this section, we also need to introduce an additional incorruptible player called the Observer. He takes no input and does not take part in the real protocol. But when $\mathcal{I}_{\text{mem}}$ overwrites the data, it sends "overwrite" to the Observer, who then outputs "overwrite" to the environment.

The augmented memory functionality. We also define an "augmented" memory functionality $\mathcal{I}^*_{\text{mem}}$, which will allow for a more modular description of our main construction (see Section 5). The functionality $\mathcal{I}^*_{\text{mem}}$ is conceptually very similar to $\mathcal{I}_{\text{mem}}$, but instead of being parametrized by a sharing scheme is parametrized by an entangled encoding scheme. The main difference is that now clients are allowed to send the actual files and the randomness, and the functionality defines $c \in \mathcal{C}$ to be an encoding of all files using the given randomness.

The augmented memory functionality is presented in Figure 2. In Appendix A, we show that $\mathcal{I}^*_{\text{mem}}$ can be securely realized (for the entangled encoding based on polynomials, see Section 3) from the more basic functionality $\mathcal{I}_{\text{mem}}$, and a suitable sharing scheme over a finite field.

4.2 Ideal Implementation of Data Entanglement

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 into a single one or by allowing to put in as many files as desired.

The functionality $\mathcal{I}_{\text{ESS}}$ is shown in Figure 3. Below, we give a high level overview of the security properties captured by $\mathcal{I}_{\text{ESS}}$.

The functionality runs with a set of clients $\{P_1, \ldots, P_n\}$ (willing to entangle their files), a server S, ideal adversary $\mathcal{SIM}$ and Observer $\mathcal{OBS}$. The entanglement process consists simply in the clients handling their file f_i to the functionality: at the end of this process the server learns nothing, and each of the clients does not learn anything about the other clients' files. In other words, each party only learns that the session is "entangled", but nothing beyond that (in this sense the entanglement process is private).

At any point in time the adversary can decide to cheat and "forget" or alter part of the clients' data; this is captured by the (`Overwrite`) command. Whenever this happens, the functionality outputs (`Overwrite`) to the Observer that then writes it on its own output tape.

Furthermore, client P_i can ask the functionality to recover f_i. In case the adversary allows this, the functionality first checks whether the (`Overwrite`) command was never issued: If this is the case, it gives file f'_i (where $f'_i = f_i$ if the server is not corrupt) to P_i and outputs nothing to S (in this sense the recovery process is private); otherwise it outputs $\perp$ to P_i (this captures the all-or-nothing integrity property).

5 A Protocol for Data Entanglement

Next, we present our main protocol securely realizing the ideal functionality $\mathcal{I}_{\text{ESS}}$ in the $\mathcal{I}_{\text{mem}}$-hybrid model. We do this in two steps. In the first step, we show a protocol Π securely realizing $\mathcal{I}_{\text{ESS}}$ in the $\mathcal{I}^*_{\text{mem}}$-hybrid model. Then, in Appendix A, we build a protocol Π' securely realizing

Functionality $\mathcal{I}_{\text{ESS}}$:

The functionality $\mathcal{I}_{\text{ESS}}$ 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 corrupt) clients $P_1, \ldots, P_n$, a (possibly corrupt) server S, an (incorruptible) Observer $\mathcal{OBS}$, and ideal adversary $\mathcal{SIM}$ is enabled via the following queries:

- On input $(\texttt{Entangle}, i, f_i)$ from P_i (where $f_i \in \mathcal{F}$), record (P_i, f_i). Ignore any subsequent query $(\texttt{Entangle}, i, *)$ from P_i. If there are already n recorded tuples, send $\texttt{Entangled}$ to all clients, to S, and to $\mathcal{SIM}$. Mark session as $\texttt{Entangled}$.
- On input $(\texttt{Overwrite})$ from $\mathcal{SIM}$, set bad to *true* and forward $(\texttt{Overwrite})$ to $\mathcal{OBS}$.
- On input $(\texttt{Recover}, i)$ from P_i, check if session is $\texttt{Entangled}$. If not ignore the input. Otherwise, record $(\texttt{Pending}, i)$ and send $(\texttt{Recover}, i)$ to S and $\mathcal{SIM}$.

  On input $(\texttt{Recover}, S, i)$ from S or $\mathcal{SIM}$, check if session is $\texttt{Entangled}$ and record $(\texttt{Pending}, i)$ exists. If not, ignore the input. Otherwise:

  - If S and P_i are both corrupt ignore the input.
  - If P_i is corrupt and S is honest, hand $(\texttt{Cheat}, i)$ to $\mathcal{SIM}$. Upon input $(\texttt{Cheat}, i, f_i')$ from $\mathcal{SIM}$, output f_i' to P_i.
  - If S is corrupt and P_i is honest, in case bad is *true* output $\perp$ to P_i. Otherwise hand $(\texttt{Cheat}, S)$ to $\mathcal{SIM}$. Upon input $(\texttt{Cheat}, S, \mathsf{deliver} \in \{\mathsf{yes}, \mathsf{no}\})$ from $\mathcal{SIM}$, if $\mathsf{deliver} = \mathsf{yes}$ output f_i to P_i and if $\mathsf{deliver} = \mathsf{no}$ output $\perp$ to P_i.
  - If S and P_i are both honest, output f_i to P_i.

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

Figure 3: Ideal functionality $\mathcal{I}_{\text{ESS}}$ for entangled storage

$\mathcal{I}_{\text{mem}}^*$ in the $\mathcal{I}_{\text{mem}}$-hybrid model. It follows by the composition theorem (cf. Theorem 1) that $\Pi^{\Pi'}$ securely realizes $\mathcal{I}_{\text{ESS}}$ in the $\mathcal{I}_{\text{mem}}$-hybrid model.

The main protocol. Our protocol for entangled storage relies on the following building blocks:

- The entangled encoding scheme $(\mathsf{Setup}, \mathsf{Encode}, \mathsf{Decode})$ based on polynomials over a finite field $\mathbf{F} = GF(2^{max(\ell, 3k+\log n+\log\log n)})$ (see Section 3). The functionality $\mathcal{I}_{\text{mem}}^*$ will be parametrized by this encoding.
- A somewhat homomorphic encryption $\mathcal{HE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ scheme with message space $\mathbf{F}$, that is able to perform up to n multiplications and an arbitrarily large number of additions.
- An interactive argument of knowledge $(\mathcal{P}, \mathcal{V})$ for the following NP-language:

$$L = \{(evk, e, e^*) : \ \exists c^*(\cdot) \text{ s.t. } e^* = \mathsf{Eval}_{evk}(c^*(\cdot), e)\}\,, \tag{1}$$

  where the function $c(\cdot)$ is a polynomial of degree $n \in \mathbb{N}$ with coefficients in $\mathbf{F}$.

We implicitly assume that there exists an efficient mapping to encode binary strings of length ℓ as elements in $\mathbf{F}$.

We start with an informal description of the protocol. In the first step each client stores in $\mathcal{I}^*_{\text{mem}}$ its own file f_i together with the randomness $r_i = (s_i, x_i)$ needed to generate the entanglement. To recover f_i, each client sends an encryption of x_i to the server using a somewhat homomorphic encryption scheme (see Section 2.4); the server can thus compute an encryption of $c(x_i)$ homomorphically and forward it to the client, together with the a proof that the computation was done correctly.[7] The client verifies the proof and, after decryption, recovers $f_i = c(x_i) - s_i$.

The actual protocol is slightly different than this, in that, in order to have each client only keep a short state, the randomness r_i is computed using a PRG such that each client can just store the seed and recover r_i at any time. Moreover each client has to also store a hash value of the file, in order to verify that the retrieved file is the correct one. (Note that a successful verification of the proof is not sufficient for this, as it only ensures that the answer from the server was computed correctly with respect to *some* polynomial.) A detailed description of protocol Π follows:

Parameters Generation. Let $\mathbf{F}$ be a finite field. Upon input a security parameter $k \in \mathbb{N}$, the number of clients n and the length parameter ℓ, output the description of a collision resistant hash function $H(\cdot)$ with input space $\{0,1\}^\ell$ and $(\mathcal{F} = \mathbf{F}, \mathcal{R} = \mathbf{F}^2, \mathcal{C} = \mathbf{F}^n) \leftarrow \mathsf{Setup}(1^k, n, \ell)$. Furthermore, provide each client P_i with secret parameters (σ_i, sk_i). Here, σ_i is the seed for a (publicly available) pseudo-random generator $G : \{0,1\}^k \to \{0,1\}^{2max(\ell, 3k+\log n+\log\log n)}$ and $(pk_i, sk_i, evk_i) \leftarrow \mathsf{Gen}(1^k)$ are the public/secret/evaluation key of a somewhat homomorphic encryption scheme $\mathcal{HE} = (\mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ with message space $\mathbf{F}$.

Entanglement. Each client P_i defines $G(\sigma_i) := (s_i, x_i) \in \mathbf{F}^2$ and sends $(\mathtt{Store}, i, f_i, (s_i, x_i))$ to $\mathcal{I}^*_{\text{mem}}$. Note that, as a consequence, $\mathcal{I}^*_{\text{mem}}$ now holds $c = \mathsf{Encode}(f_1, \ldots, f_n, r_1, \ldots, r_n)$ where $r_i = (s_i, x_i)$. Recall that the entanglement corresponds to the coefficients $c = (c_0, \ldots, c_{n-1})$ of the polynomial $c(X)$ (of minimum degree) interpolating all points $(x_i, f_i + s_i)$. The clients store the seed σ_i, and a hash value $\theta_i = H(f_i)$.

Recovery. To retrieve f_i, client P_i first computes $(s_i, x_i) = G(\sigma_i)$ and then interacts with the server S as follows:

1. P_i computes $e \leftarrow \mathsf{Enc}_{pk_i}(x_i)$ and sends it to S.
2. S sends $(\mathtt{Read}, \mathcal{M}, i)$ to $\mathcal{I}^*_{\text{mem}}$, where the Turing machine $\mathcal{M}$ runs $e^* = \mathsf{Eval}_{evk_i}(c(\cdot), e)$.
3. Let $(\mathcal{P}, \mathcal{V})$ be an interactive argument of knowledge for the language of Eq. (1). The server S plays the role of the prover and client P_i the role of the verifier; in case $(\mathcal{P}(c(\cdot)), \mathcal{V})(evk_i, e, e^*) = 0$ the client outputs $\perp$.[8]
4. P_i computes $c(x_i) = \mathsf{Dec}_{sk_i}(e^*)$ and outputs $f_i = c(x_i) - s_i$ if and only if $H(f_i) = \theta_i$.

In Appendix B we discuss several variant of the above protocol Π, leading to different efficiency trade-offs. We prove the following result.

Theorem 3 *Assuming the PRG is secure, the hash function is collision resistant, and $\mathcal{HE}$ is CPA-secure, the above protocol Π securely realizes $\mathcal{I}_{\text{ESS}}$ in the $\mathcal{I}^*_{\text{mem}}$-hybrid model.*

[7] Recall that the server does not have direct access to the data, so the above computation is performed by issuing commands to $\mathcal{I}^*_{\text{mem}}$.

[8] Note that the above requires one suitable $(\mathtt{Read}, \mathcal{M}, i)$ command from S to $\mathcal{I}^*_{\text{mem}}$ for each message from the prover to the verifier in $(\mathcal{P}(c(\cdot)), \mathcal{V})(evk_i, e, e^*)$ (this is because the witness $c(\cdot)$ is stored inside $\mathcal{I}^*_{\text{mem}}$).

Proof. Since the adversary is static, the set of corrupt parties is fixed once and for all at the beginning of the execution; we denote this set by Δ. Our goal is to show that for all adversaries $\mathcal{A}$ corrupting parties in a real execution of Π, there exists a simulator $\mathcal{SIM}$ interacting with the ideal functionality $\mathcal{I}_{\text{ESS}}$, such that for all environments $\mathcal{Z}$ and all inputs $f_1, \ldots, f_n \in \mathbf{F}$,

$$\{\mathbf{IDEAL}_{\mathcal{I}_{\text{ESS}},\mathcal{SIM}(z),\mathcal{Z}}(k,(f_1,\ldots,f_n))\}_{k\in\mathbb{N},z\in\{0,1\}^*} \approx \{\mathbf{REAL}_{\Pi,\mathcal{A}(z),\mathcal{Z}}(k,(f_1,\ldots,f_n)\}_{k\in\mathbb{N},z\in\{0,1\}^*}.$$

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

1. Upon input security parameter k, secret values (f_i, σ_i, sk_i) (for all $i \in [n]$ such that $P_i \in \Delta$), public values $(pk_i, evk_i)_{i\in[n]}$, and auxiliary input z, the simulator invokes $\mathcal{A}$ on these inputs.

2. Every input value that $\mathcal{SIM}$ receives from $\mathcal{Z}$ externally is written into the adversary $\mathcal{A}$'s input tape (as if coming from $\mathcal{A}$'s environment). Every output value written by $\mathcal{SIM}$ on its output tape is copied to $\mathcal{A}$'s own output tape (to be read by the external $\mathcal{Z}$).

3. Upon receiving $(\texttt{Store}, i, f'_i, r'_i)$ from $P_i \in \Delta$ (where r'_i is a pair $(s'_i, x'_i) \in \mathbf{F}^2$), issue $(\texttt{Entangle}, i, f'_i)$. After receiving message `Entangled` from $\mathcal{I}_{\text{ESS}}$, return `Done` to P_i.

4. Sample $(f'_i, s'_i, x'_i) \leftarrow \mathbf{F}^3$ and define $y'_i = f'_i + s'_i$ for all $i \in [n]$ such that $P_i \notin \Delta$. Emulate the ideal functionality $\mathcal{I}^*_{\text{mem}}$ by computing the polynomial $c \in \mathbf{F}[X]$ of minimal degree interpolating $(x'_i, y'_i)_{i\in[n]}$; let $\mathcal{K} = \emptyset$.

5. Upon receiving $(\texttt{Overwrite}, \{i_j\}_{j\in[t]})$ from $\mathcal{A}$, set $c[i_j] = 0$ and update $\mathcal{K} \leftarrow \mathcal{K} \cup \{i_j\}_{j\in[t]}$. In case $|\mathcal{K}| \geq k$, send $(\texttt{Overwrite})$ to $\mathcal{I}_{\text{ESS}}$.

6. Whenever the adversary forwards a ciphertext e on behalf of a corrupt player $P_i \in \Delta$, send $(\texttt{Recover}, i)$ to $\mathcal{I}_{\text{ESS}}$ and receive back $(\texttt{Recover}, i)$. Then, in case $S \notin \Delta$, act as follows:

   (a) Send $(\texttt{Recover}, S, i)$ to $\mathcal{I}_{\text{ESS}}$ and receive back message $(\texttt{Cheat}, i)$. Run $x'_i = \mathsf{Dec}_{sk_i}(e)$, define $f'_i = c(x'_i) - s'_i$ and send $(\texttt{Cheat}, i, f'_i)$ to $\mathcal{I}_{\text{ESS}}$.

   (b) Simulate the ciphertext $e^* = \mathsf{Eval}_{evk_i}(c(\cdot), e)$ and play the role of the prover in $(\mathcal{P}(c(\cdot)), \mathcal{V})(evk_i, e, e^*)$ (with P_i being the verifier).

7. Upon receiving $(\texttt{Recover}, i)$ from $\mathcal{I}_{\text{ESS}}$ (for $i \in [n]$ such that $P_i \notin \Delta$), in case $S \in \Delta$ act as follows:

   (a) Simulate the ciphertext $e \leftarrow \mathsf{Enc}_{pk_i}(x'_i)$ for the previously chosen value x'_i.

   (b) Wait for the next $(\texttt{Read}, \mathcal{M}^*, i)$ command from $\mathcal{A}$ (if any) and forward $(\texttt{Recover}, S, i)$ to $\mathcal{I}_{\text{ESS}}$. Upon input $(\texttt{Cheat}, S)$ from $\mathcal{I}_{\text{ESS}}$, play the role of the verifier in $(\mathcal{P}(c(\cdot)), \mathcal{V})(evk_i, e, \mathcal{M}^*(c))$ (with S being the prover).

   (c) In case the above check passes and the proof is verified correctly, issue $(\texttt{Cheat}, S, \texttt{yes})$, and otherwise issue $(\texttt{Cheat}, S, \texttt{no})$.

8. Upon receiving $(\texttt{Read}, \mathcal{M}, i)$ from the adversary, in case $P_i \in \Delta$ forward $\mathcal{M}(c)$ to P_i.

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

We consider a series of intermediate hybrid experiments, to show that the ideal output and the real output are computationally close. A description of the hybrids follow.

Hybrid $\mathbf{HYB}^1(k,(f_1,\ldots,f_n))$. We replace $\mathcal{SIM}$ by $\mathcal{SIM}^1$ which knows the real inputs f_i of the honest clients and uses these values to define the polynomial c in step 4. From the privacy property of the entangled encoding scheme, we get that $\mathbf{IDEAL}_{\mathcal{I}_{\text{ESS}},\mathcal{SIM}(z),\mathcal{Z}}(k,(f_1,\ldots,f_n))$ and $\mathbf{HYB}^1(k,(f_1,\ldots,f_n))$ are statistically close.

Hybrid $\mathbf{HYB}^2(k,(f_1,\ldots,f_n))$. We replace $\mathcal{SIM}^1$ by $\mathcal{SIM}^2$ which instead of sending an encryption of x_i' in step 7a, defines $e \leftarrow \mathsf{Enc}_{pk_i}(x_i'')$ for a random $x_i'' \leftarrow \mathbf{F}$.

We argue that if one could distinguish between the distribution of $\mathbf{HYB}^1(k,(f_1,\ldots,f_n))$ and $\mathbf{HYB}^2(k,(f_1,\ldots,f_n))$, then we could define an adversary $\mathcal{B}$ breaking semantic security of $\mathcal{HE}$. Adversary $\mathcal{B}$ receives the target public key (pk^*, evk^*) for $\mathcal{HE}$ and behaves exactly as $\mathcal{SIM}^1$ with the difference that it sets $(pk_i, evk_i) = (pk^*, evk^*)$. The challenge is set to the value x_i' chosen by $\mathcal{SIM}$ in step 4; denote with e the corresponding ciphertext, which is either an encryption of x_i' or an encryption of a randomly chosen x_i''. Then $\mathcal{B}$ uses e in step 7a to simulate the ciphertext sent from P_i to S.

Now, if the distinguisher guesses to be in $\mathbf{HYB}^1(k,(f_1,\ldots,f_n))$, the adversary guesses that e must be an encryption of x_i' (i.e., output "real"), and otherwise if the distinguisher guesses to be in $\mathbf{HYB}^2(k,(f_1,\ldots,f_n))$, the adversary guesses that the challenge ciphertext must encrypt an independent value (i.e., output "random"). Thus, semantic security of $\mathcal{HE}$ implies that the two hybrids are computationally indistinguishable.

Hybrid $\mathbf{HYB}^3(k,(f_1,\ldots,f_n))$. We replace $\mathcal{SIM}^2$ by $\mathcal{SIM}^3$ which on step 5 does not send (`Overwrite`) to $\mathcal{I}_{\text{ESS}}$, but instead continues to answer recovery queries from $P_i \in \Delta$ as done by $\mathcal{SIM}$ in step 7. Notice that in $\mathbf{HYB}^2$, once the flag `bad` is set, the ideal functionality would answer all such queries with $\perp$.

Let BAD be the following event: BAD becomes true whenever an honest client P_i accepts the output of a recovery query (produced via Turing machine $\mathcal{M}$) as valid, and $|\mathcal{K}| \geq k$ (i.e., the flag `bad` was already set in the previous hybrid). Denote with $\widetilde{\mathbf{HYB}}^3(k,(f_1,\ldots,f_n))$ the distribution of $\mathbf{HYB}^3$ conditioned on BAD not happening. Clearly $\mathbf{HYB}^2(k,(f_1,\ldots,f_n))$ and $\widetilde{\mathbf{HYB}}^3(k,(f_1,\ldots,f_n))$ are identically distributed; next we argue that BAD happens with probability exponentially small in k, which implies that $\mathbf{HYB}^2(k,(f_1,\ldots,f_n))$ and $\mathbf{HYB}^3(k,(f_1,\ldots,f_n))$ are close.

We rely here on the fact that, for $\mathbf{F} = GF(2^{max(\ell,3k+\log n+\log\log n)})$, our entangled encoding scheme of Section 3 has $(k, \sqrt{8}\cdot 2^{-k/2})$ all-or-nothing integrity. In particular, any adversarial strategy provoking event BAD with probability $\geq \sqrt{8}\cdot 2^{-k/2}$ starting from a polynomial where at least k 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 behave as $\mathcal{SIM}^3$ and, after P_i is done with verifying the proof and recovering its file, run the extractor $\mathcal{EXT}_{\mathcal{P}}$ to obtain a witness $c^*(\cdot)$. Then $\mathcal{SIM}^3$ sets $\mathsf{Decode}_{Adv} := (c^*(\cdot) - s_i', c^*(\cdot))$.

Clearly, in case BAD happens, we obtain that the reduction above breaks the all-or-nothing integrity of the encoding scheme provided that the extractor does not fail to extract a valid witness (which will happen with negligible probability by simulation extractability). Thus, by Theorem 2, we get that $\mathbb{P}(\text{BAD}) \leq \sqrt{8}\cdot 2^{-k/2}$.[9]

[9]Note that in order to apply Theorem 2, we need the property that the view in the reduction is independent of x_i';

Hybrid $\mathbf{HYB}^4(k,(f_1,\ldots,f_n))$. We replace $\mathcal{SIM}^3$ by $\mathcal{SIM}^4$, which answers recovery queries differently in case S is corrupt. Namely, on step 7 $\mathcal{SIM}^4$ does not send $(\mathtt{Cheat}, S, \mathsf{deliver})$ to $\mathcal{I}_{\mathrm{ESS}}$, but instead computes the answer to a recovery query from honest P_i as $f_i' = c(x_i') - s_i'$ where $c(\cdot)$ is the emulated polynomial used by $\mathcal{SIM}$. The only difference between $\mathbf{HYB}^3(k,(f_1,\ldots,f_n))$ and $\mathbf{HYB}^4(k,(f_1,\ldots,f_n))$ is that in the former the ideal functionality would always answer queries where $(\mathtt{Cheat}, S, \mathsf{yes})$ was sent with the correct value f_i.

Let BAD′ be the event that P_i accepts the output of a recovery query in $\mathbf{HYB}^4(k,(f_1,\ldots,f_n))$ and $f_i' \neq f_i$; clearly the distribution of $\mathbf{HYB}^3(k,(f_1,\ldots,f_n))$ and the distribution of $\mathbf{HYB}^4(k,(f_1,\ldots,f_n))$ conditioned on BAD′ not happening are identical. It is easy to verify that the probability of BAD′ is negligible, otherwise one could break collision resistance of $H(\cdot)$. The reduction is straightforward and is therefore omitted.

Hybrid $\mathbf{HYB}^5(k,(f_1,\ldots,f_n))$. We replace $\mathcal{SIM}^4$ with $\mathcal{SIM}^5$, that computes again the ciphertext in step 7a by encrypting the right value x_i'. Semantic security of $\mathcal{HE}$ implies that $\mathbf{HYB}^4(k,(f_1,\ldots,f_n))$ and $\mathbf{HYB}^5(k,(f_1,\ldots,f_n))$ are computationally close. The proof is analogous to an above argument, and is therefore omitted.

Hybrid $\mathbf{HYB}^6(k,(f_1,\ldots,f_n))$. We replace $\mathcal{SIM}^5$ by $\mathcal{SIM}^6$ 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 distinguisher between the two hybrids can be turned into another 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 $\mathbf{F}^2$, parses v_i as $v_i = (s_i, x_i)$ and uses these values together with files f_i to define the emulated polynomial c in step 4. Now, when the v_i's are uniform, the distribution is the same as in $\mathbf{HYB}^5(k,(f_1,\ldots,f_n))$, whereas when $v_i = G(\sigma_i)$ the distribution is the same as in hybrid $\mathbf{HYB}^6(k,(f_1,\ldots,f_n))$. Thus, given a distinguisher between the two hybrids we can break the pseudo-randomness of $G(\cdot)$.

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. We have thus showed that $\mathbf{IDEAL}_{\mathcal{I}_{\mathrm{ESS}},\mathcal{SIM}(z)}(k,(f_1,\ldots,f_n))$ and $\mathbf{REAL}_{\mathcal{ESS},\mathcal{A}(z)}(k,(f_1,\ldots,f_n))$ are computationally close, as desired.

Acknowledgments

We thank the anonymous reviewers of Crypto 2013 and TCC 2014 for the useful feedback provided on earlier versions of this paper.

References

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

this is clearly the case, as in $\mathbf{HYB}^3(k,(f_1,\ldots,f_n))$ the ciphertext e has been replaced by an encryption of a random value.

[2] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Xiaodong Song. Provable data possession at untrusted stores. In *ACM CCS*, pages 598–609, 2007.

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

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

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

[6] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption without Bootstrapping. In *ITCS*, page to appear, 2012.

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

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

[9] Özgür Dagdelen, Payman Mohassel, and Daniele Venturi. Rate-limited secure function evaluation: Definitions and constructions. In *Public Key Cryptography*, pages 461–478, 2013.

[10] Francesco Davì, Stefan Dziembowski, and Daniele Venturi. Leakage-resilient storage. In *SCN*, pages 121–137, 2010.

[11] Yevgeniy Dodis, Salil P. Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *TCC*, pages 109–127, 2009.

[12] Pierre-Alain Fouque and David Pointcheval. Threshold cryptosystems secure against chosen-ciphertext attacks. In *ASIACRYPT*, pages 351–368, 2001.

[13] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In *PODC*, pages 101–111, 1998.

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

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

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

[17] Ari Juels and Burton S. Kaliski Jr. PoRs: proofs of retrievability for large files. In *ACM CCS*, pages 584–597, 2007.

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

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

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

[21] Moni Naor and Guy N. Rothblum. The complexity of online memory checking. In *FOCS*, pages 573–584, 2005.

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

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

[24] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *ASIACRYPT*, pages 90–107, 2008.

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

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

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

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

A A Protocol for Realizing $\mathcal{I}^*_{\text{mem}}$

We describe a protocol Π' that securely realizes $\mathcal{I}^*_{\text{mem}}$ in the $\mathcal{I}_{\text{mem}}$-hybrid model (see Section 4.1), whenever $\mathcal{I}^*_{\text{mem}}$ is parametrized by our encoding scheme based on polynomials (see Section 3.2). Recall that $\mathcal{I}_{\text{mem}}$ is parametrized by a sharing scheme $(\mathsf{Share}, \mathsf{Reconstruct})$. We propose two concrete instantiations below:

- *Threshold additively homomorphic encryption (e.g., Paillier's cryptosystem [22, 12]).* 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., [25, 13]).* 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.

Let $\mathbf{F}$ be a finite field. Consider the following linear system $\mathbf{A} \cdot \mathbf{c} = \mathbf{b}$, where

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

and $\mathbf{A}$ is a Vandermonde matrix. Note that if the x_i's are distinct, $\mathbf{A}$ is non-singular and can thus be inverted yielding the vector $\mathbf{c} = \mathbf{A}^{-1} \cdot \mathbf{b}$ containing the coefficients of the polynomial $c(X)$ of minimal degree interpolating all (x_i, y_i).

Denote with $\mathbf{A} = (\mathbf{A}[1], \dots, \mathbf{A}[n])$ the rows of $\mathbf{A}$ and with $\mathbf{b} = (\mathbf{b}[1], \dots, \mathbf{b}[n])$ the elements of $\mathbf{b}$. The following protocol Π' runs with clients P_i holding an input $(x_i, y_i) \in \mathbf{F}^2$, and is based on [4].

1. Each client P_i shares $\mathbf{A}[i]$ and $\mathbf{b}[i]$.
2. Clients 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.
3. Each client computes 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.
4. Each client computes the shares of $\mathbf{A}^{-1} \cdot \mathbf{b}$ non-interactively.
5. For all $j \in [0, n-1]$, let $s_{i,j}$ be the share of $\mathbf{c}[j]$ held by P_i. Client P_i issues $(\texttt{Store}, i, s_{i,j})$.

The above protocol 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 [20] with only $O(n^2)$ multiplications. The parties share polynomials $\xi(X) = \prod_{i=1}^{n}(X - x_i)$ and $\xi_i(X) = \xi(X)/(X - x_i)$ (for all $i = 1, \dots, n$) and values $\xi_i(x_i)$, $\xi_i^{-1}(x_i)$ (for all $i = 1, \dots, n$). Hence, parties can compute the shares of $\xi_i'(X) = \xi_i(X) \cdot \xi_i(x_i)^{-1}$ (for all $i = 1, \dots, n$) and obtain the shares of $c(X) = \sum_{i=1}^{n} \xi_i'(X) \cdot \mathbf{b}[i]$.

The type of security we achieve depends on the particular sharing scheme we employ. In case of passive adversaries, the protocols above are secure for adversary structure $\Delta = \mathbf{Q}_2$ (i.e., no two sets in Δ cover the entire set of clients). In case of active adversaries, we can tolerate $\Delta = \mathbf{Q}_3$ by using verifiable secret sharing or zero-knowledge proofs ($\Delta = \mathbf{Q}_2$ assuming a broadcast channel). In case protocol Π' above is instantiated using verifiable secret sharing (with no broadcast channel available), and setting $y_i = f_i + s_i$ for $(s_i, x_i) = G(\sigma_i)$, we obtain, e.g., the following statement:

Theorem 4 *Protocol Π' above $\mathbf{Q}_3$-securely realizes $\mathcal{I}_{\text{mem}}^*$ in the $\mathcal{I}_{\text{mem}}$-hybrid model, with active corruptions.*

The proof follows directly by the result in [4]. The intuition is that the simulator can extract, by relying on the properties of the verifiable secret sharing scheme, input and randomness for the corrupt players to be forwarded to the ideal functionality $\mathcal{I}_{\text{mem}}^*$.

B Secure Polynomial Evaluation

In this appendix we discuss a few variants of our main protocol Π (see Section 5). Recall that in protocol Π, whenever a client P_i wants to retrieve its own file it runs a sub-protocol Π'' for evaluating the polynomial $c(\cdot)$ at point $x_i \in \mathbf{F}$. Intuitively Π'' guarantees that P_i learns nothing more than $c(x_i)$, whereas the server does not learn anything on the client's input. A related problem is the one of oblivious polynomial evaluation (OPE) [16] (see also [9]), where the server holds the actual polynomial and we want that additionally the client does not learn anything about $c(\cdot)$, apart from the value $c(x)$ itself. Note that any protocol for OPE could be used as a sub-protocol for file recovery in Π, but given the efficiency of OPE our solution is more efficient.

An alternative approach is to replace the somewhat homomorphic encryption scheme with an additively homomorphic encryption scheme, e.g. Paillier [22]. In this case the client would send the powers $\{x^i\}_{i=1}^{n-1}$ encrypted, and the server would evaluate $c(x)$ homomorphically 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 $c(x)$. This solution requires the transmission of n field elements from the client to the server and one field element from the server to the client.

Efficiency considerations. We observe that the efficiency of sub-protocol Π'', 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, this sub-protocol is less efficient than the solution based on additively homomorphic encryption. (Indeed, with [22], 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 us 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 B 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 the communication complexity from $O(n)$ down 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 $\sum_j \delta_j = n - 1$.[10] 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.

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 $\mathbf{F}$. 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 4, upon input coefficients $c_0, \ldots, c_{n-1}$, outputs polynomials $\zeta_0(\cdot), \ldots, \zeta_m(\cdot)$ each

[10] 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.

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 4: 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 4 can be easily verified. We need to show $c_{n-1}X^{n-1} + \ldots + c_1X + c_0 = \zeta_m(X) \cdot X^{m \cdot m} + \ldots + \zeta_1(X) \cdot X^m + \zeta_0(X)$. 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 provided.