Towards a Theory of Data Entanglement*
(Extended Abstract)

James Aspnes**, Joan Feigenbaum***,
Aleksandr Yampolskiy†, and Sheng Zhong‡

Department of Computer Science, Yale University, New Haven CT 06520-8285, USA
aspnes@cs.yale.edu,
{joan.feigenbaum,aleksandr.yampolskiy,sheng.zhong}@yale.edu

Abstract. We give a formal model for systems that store data in entangled form. We propose a new notion of entanglement, called **all-or-nothing integrity** (AONI) that binds the users' data in a way that makes it hard to corrupt the data of any one user without corrupting the data of all users. AONI can be a useful defense against negligent or dishonest storage providers who might otherwise be tempted to discard documents belonging to users without much clout. We show that, if all users use the standard recovery algorithm, we can implement AONI using a MAC, but, if some of the users adopt the adversary's non-standard recovery algorithm, AONI can no longer be achieved. However, even for the latter scenario, we describe a simple entangling mechanism that provides AONI for a restricted class of destructive adversaries.

1 Introduction

Suppose that I provide you with remote storage for your most valuable information. I may advertise various desirable properties of my service: underground disk farms protected from nuclear attack, daily backups to chiseled granite monuments, replication to thousands of sites scattered across the globe. But what assurance do you have that I will not maliciously delete your data as soon as your subscription check clears?

If I consider deleting the data of a rich or powerful customer, I may be deterred by economic, social, or legal repercussions. The small secret joy I might experience from the thought of the loss will not compensate me for losing a posted bond, destroying my reputation, or being imprisoned. But if you are an ordinary customer who does not have much clout, and I see a lucrative opportunity in altering – or simply neglecting to keep – your data, then deterrence loses

* This work was supported by the DoD University Research Initiative (URI) administered by the Office of Naval Research under Grant N00014-01-1-0795.
** Supported in part by NSF grants CCR-0098078 and CNS-0305258.
*** Supported in part by ONR grant N00014-01-1-0795 and NSF grants ITR-0219018 and ITR-0331548.
† Supported by NSF grants CCR-0098078 and ANI-0207399.
‡ Supported by NSF grant CCR-0208972.

P. Samarati et al. (Eds.): ESORICS 2004, LNCS 3193, pp. 177–192, 2004.
© Springer-Verlag Berlin Heidelberg 2004

its effectiveness. Consequently, data of powerful customers end up being more protected than data of average customers. To convince an average customer that she will not lose her data at my random whim, I might offer stronger technical guarantees that I cannot destroy her data without serious costs. One way to do this would be to link the fate of her documents to the documents of enough other users that I cannot hope to offend them all with impunity. We shall call such documents **entangled**.

Data entanglement was initially suggested as a mechanism for increasing censorship-resistance in document-storage systems, *e.g.*, Dagster [21] and Tangler [22]. These systems split data into blocks in such a way that a single block becomes part of several documents. New documents are represented using some number of existing blocks, chosen randomly from the pool, combined with new blocks created using exclusive-or (Dagster) or 3-out-of-4 secret sharing [19] (Tangler). Dagster and Tangler use entanglement as one of many mechanisms to prevent a censor from tampering with unpopular data; others involve disguising the ownership and contents of documents and (in Tangler) storing documents redundantly.

It is not clear that data entanglement is actually useful for censorship resistance. Instead of having to specifically attack a target document, a censor only needs to damage any document entangled with the target to achieve his goal. Instead, we consider data entanglement for a different purpose: protecting the data from an untrusted storage provider that might be tempted to damage or destroy the data through negligence or malice. Entanglement provides an incentive for the storage provider to take extra care in protecting average users' documents by increasing the cost of errors.

We begin in Section 2 by analyzing the intuitive notion of entanglement provided by Dagster and Tangler. We show that entanglement as provided by Dagster and Tangler is not by itself sufficiently strong to deter a dishonest storage provider from tampering with data, because not enough documents get deleted on average when destroying a block of a typical document. This motivates our efforts to obtain a stronger form of an entanglement than the ones provided by these systems.

In Section 3, we define our general model of entanglement in an untrusted storage system. Our goal here is to model the entanglement operation itself, and we do not address the question of where in the system entanglement occurs. However, we do assume that the storage provider does *not* carry out the entangling operation itself, as giving it the users' raw data would allow it to store copies that it could selectively return later, even if the entangled store were lost or damaged. Instead, some trusted third party is assumed to carry out the entangling operation, and a negligent or malicious storage provider is modeled separately as a "tamperer" that has access only to the entangled store.

Section 4 contains our definitions of **document dependency**, where a document cannot be recovered if any document it depends on is lost, and **all-or-nothing integrity**, where no document can be recovered if any document is lost. These definitions allow a system-independent description of the binding between entangled documents. We then consider how different levels of attacks on

the common data store do or do not prevent enforcement of document dependency or all-or-nothing integrity.

In particular, we show that, if all clients use a standard algorithm to recover their data, then all-or-nothing integrity requires only the ability to detect tampering using a MAC (Section 5.1); in this model, the standard recovery algorithm is too polite to return any user's data if any other user's data has been lost. Relying on such fastidiousness provides only a weak guarantee; what we really want is that all data becomes irretrievable even by non-standard algorithms if any is lost. We show that this goal is impossible if an adversary is allowed to both tamper with the common store arbitrarily and provide a replacement recovery algorithm (Section 5.2). Despite such **upgrade attacks**, it is still possible to provide a weaker guarantee that we call **symmetric recovery**, in which each document is equally likely to be destroyed (Section 5.3). Furthermore, if we restrict the adversary to **destructive tampering**, which reduces the amount of information in the common store, all-or-nothing guarantees are possible even with upgrade attacks (Section 5.4).

These results provide a first step toward understanding document dependency. Suggestions for further work are given in Section 6.

Because of space limitations, many proofs are omitted from this extended abstract. Complete proofs can be found in the full version, available as a Yale CS technical report [2].

1.1 Related Work

Entanglement is motivated by the goal of deterring data tampering by untrusted servers, a more general problem that has been studied extensively. Entanglement has been used specifically in Dagster [21] and Tangler [15], as we describe in Section 2. Other approaches to preventing or deterring tampering include replication across global networks of tamper-resistant servers [1, 4, 5, 9, 17, 23] or tamper detection [6–8, 12–14, 20] using digital signatures and Merkle hash trees [16]. Replication protects against data loss if a small number of servers are compromised; tamper detection prevents data loss from going unnoticed. Both techniques complement the entanglement approach considered here.

All-or-nothing integrity as defined in the present work is related to the guarantee provided by the **all-or-nothing transform** proposed by Rivest [18]. An all-or-nothing transform is an invertible transform that guarantees that no bits of the preimage can be recovered if ℓ bits of the image are lost. All-or-nothing transforms are not directly applicable to our problem, because we consider the more general case in which the image may be corrupted in other ways, such as by superencryption or alteration of part or all of the common store.

2 Dagster and Tangler

We now review how Dagster [21] and Tangler [22] work, concentrating on their operations at a block level and omitting details of how documents are divided

into blocks. We then analyze the resulting entangling effects and show their shortcomings for protecting data from a negligent storage provider, motivating our stronger notions of entanglement in Section 4.

2.1 Dagster

The Dagster storage system may run on a single server or on a P2P overlay network. Each document in Dagster consists of $c + 1$ server blocks: c blocks of older documents and one new block, an exclusive-or of previous blocks with the document. The $c + 1$ blocks that must be XORed to recover the document are listed in a **Dagster Resource Locator**.

2.2 Tangler

The Tangler storage system uses (3, 4) Shamir secret sharing [19] to entangle the data: Each document is represented by four server blocks, any three of which are sufficient to reconstruct the original document. The blocks get replicated across a subset of Tangler servers. Hashes of blocks are recorded in a data structure, similar to Dagster Resource Locator, called an **inode**.

2.3 Analysis of Entanglement

At a given point in time, a Dagster or Tangler server contains a set of blocks $\{C_1, \ldots, C_m\}$ comprising documents $\{d_1, \ldots, d_n\}$ of a group of users. (Here $m, n \in \mathbb{N}$ and $m \geq n$.) Data are partitioned in a way that each block becomes a part of several documents. We can draw an **entanglement graph** (see Figure 1), which has an edge (d_j, C_k) if block C_k belongs to document d_j. This connection is rather tenuous – even if (d_j, C_k) is in the graph, it may still be possible to reconstruct d_j from blocks excluding C_k. Document nodes in Dagster's entanglement graph have an out-degree $c + 1$, and those in Tangler's have out-degree 4. Entangled documents share one or more server blocks. In Figure 1, documents d_1 and d_2 are entangled because they share server block C_1; meanwhile, documents d_1 and d_2 are not entangled.

This shared-block notion of entanglement has several drawbacks. Even if document d_j is entangled with a specific document, it may still be possible to delete d_j from the server without affecting that particular document. For example, knowing that d_n is entangled with d_1 (as in Figure 1), and that d_1 is owned by some Very Important Person, may give solace to the owner of d_n, who might assume that no adversary would dare incur the wrath of the VIP merely to destroy d_n. But in the situation depicted in the figure, the adversary can still delete server blocks C_2 and C_m and corrupt d_n but not d_1.

The resulting dependence between documents is thus very weak. In the full paper, we show:

Theorem 1. *In a Dagster server with n documents, where each document is linked with c pre-existing blocks, deleting a block of a random document destroys on average $O(c)$ other documents.*

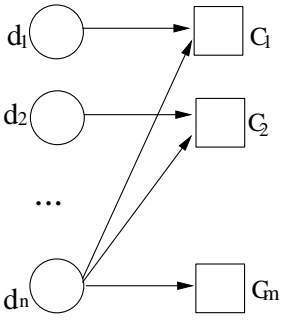


Fig. 1. An entanglement graph is a bipartite graph from the set of documents to the set of server blocks. Edge (d_j, C_k) is in the graph if server block C_k can be used to reconstruct document d_j.

Theorem 2. *In a Tangler server with n documents, deleting two blocks of a random document destroys on average* $O\left(\frac{\log n}{n}\right)$ *other documents.*

Even a small chance of destroying an important document will deter tampering to some extent, but some tamperers might be willing to run that risk. Still more troubling is the possibility that the tamperer might first flood the system with junk documents, so that almost all real documents were entangled only with junk. Since our bounds show that destruction of a typical document will on average affect only a handful of others in Dagster and almost none in Tangler, we will need stronger entanglement mechanisms if entanglement is to deter tampering by itself.

3 Our Model

In Section 3.1, we start by giving a basic framework for systems that entangle data. Specializing the general framework gives specific system models, differentiated by the choice of recovery algorithms and restrictions placed on the adversary. We discuss them in Section 3.2.

Our model abstracts away many details of storage and recovery processes. It concentrates on a single entanglement operation, which takes documents of a finite group of users and intertwines these documents to form a common store. In practice, the server contents would be computed as an aggregation of common stores from multiple entanglement operations. We defer analyzing this more complex case to later work; see the discussion of possible extensions to the model in Section 6.

3.1 Basic Framework

The model consists of an **initialization phase**, in which keys are generated and distributed to the various participants in the system; an **entanglement**

phase, in which the individual users' data are combined into a common store; a **tampering phase**, in which the adversary corrupts the store; and a **recovery phase**, in which the users attempt to retrieve their data from the corrupted store using one or more recovery algorithms. For simplicity of notation, we number the users $\{1, \ldots, n\}$, where every user i possesses a document d_i that he wants to publish.

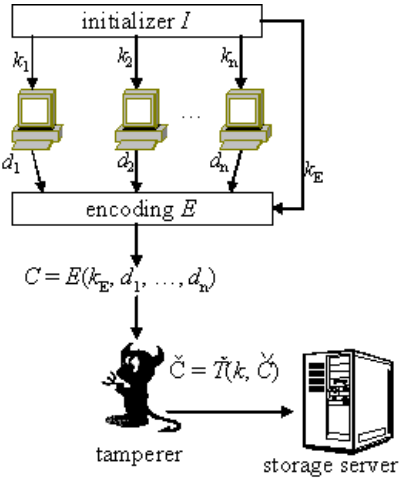


Fig. 2. Initialization, entanglement, and tampering stages.

An encoding scheme consists of three probabilistic Turing machines, (I, E, R), that run in time polynomial in the size of their inputs and a security parameter s. The first of these, the **initialization algorithm** I, hands out the keys used in the encoding and recovery phases. The second, the **encoding algorithm** E, combines the users' data into a common store using the encoding key. The third, the **recovery algorithm** R, attempts to recover each user's data using the appropriate recovery key.

Acting against the encoding scheme is an adversary $(\check{I}, \check{T}, \check{R})$, which also consists of three probabilistic polynomial-time Turing machines. The first is an **adversary-initialization algorithm** $\check{I}$; like the good initializer I, the evil $\check{I}$ is responsible for generating keys used by other parts of the adversary during the protocol. The second is a **tampering algorithm** $\check{T}$, which modifies the common store. The third is a **non-standard recovery algorithm** $\check{R}$, which may be used by some or all of the users to recover their data from the modified store.

We assume that $\check{I}$, $\check{T}$ and $\check{R}$ are chosen after I, E, and R are known but that a single fixed $\check{I}$, $\check{T}$, and $\check{R}$ are used for arbitrarily large values of s and n. This is necessary for polynomial-time bounds on $\check{T}$ and $\check{R}$ to have any effect.

Given an encoding scheme (I, E, R) and an adversary $(\check{I}, \check{T}, \check{R})$, the **storage protocol** proceeds as follows (see also Figure 2):

1. **Initialization.** The initializer I generates a combining key k_E used by the encoding algorithm and recovery keys $k_1, k_2, \ldots k_n$, where each key k_i is used by the recovery algorithm to recover the data for user i. At the same time, the adversary initializer $\check{I}$ generates the shared key $\check{k}$ for $\check{T}$ and $\check{R}$.

$$k_E, k_1, k_2, \ldots k_n \leftarrow I(1^s, n),$$
$$\check{k} \leftarrow \check{I}(1^s, n).$$

2. **Entanglement.** The encoding algorithm E computes the combined store C from the combining key k_E and the data d_i:

$$C \leftarrow E(k_E, d_1, d_2, \ldots d_n).$$

3. **Tampering.** The tamperer $\check{T}$ alters the combined store C into $\check{C}$:

$$\check{C} \leftarrow \check{T}(\check{k}, C).$$

4. **Recovery.** The users attempt to recover their data. User i applies his recovery algorithm R_i to k_i and the changed store $\check{C}$. Each R_i could be either the standard recovery algorithm R, supplied with the encoding scheme, or the non-standard algorithm $\check{R}$, supplied by the adversary, depending on the choice of the model.

$$d'_i \leftarrow R_i(k_i, \check{C})$$

We say that user i **recovers** his data if the output of R_i equals d_i.

3.2 Adversary Classes

We divide our model on two axes: one bounding the users' choices of reconstruction algorithms and the other bounding the adversary's power to modify the data store. With respect to recovery algorithms, we consider three variants on the basic framework (listed in order of increasing power given to the adversary):

- In the **standard-recovery-algorithm model**, the users are restricted to a single standard recovery algorithm R, supplied by the system designer. Formally, this means $R_i = R$ for all users i; The adversary's recovery algorithm $\check{R}$ is not used. This is the model used to analyze Dagster and Tangler.
- In the **public-recovery-algorithm model**, the adversary not only modifies the combined store, but also supplies a single non-standard recovery algorithm $\check{R}$ to all of the users. Formally, we have $R_i = \check{R}$ for each i. The original recovery algorithm R is not used[1]. We call this an **upgrade attack** by analogy to the real life situation of a company changing the data format of documents processed by its software and distributing a new version of the

[1] Though it may seem unreasonable to prevent users from choosing the original recovery algorithm R, any R can be rendered useless in practice by superencrypting the data store and distributing the decryption key only with the adversary's $\check{R}$. We discuss this issue further in Section 5.2.

software to read them. We believe such an attack is a realistic possibility, because most self-interested users will be happy to adopt a new recovery algorithm if it offers new features or performance, or if the alternative is losing their data.
- In the **private-recovery-algorithm model**, the adversary may choose to supply the non-standard recovery algorithm $\check{R}$ to only a subset of the users. The rest continue to use the standard algorithm R. Formally, this model is a mix of the previous two models: $R_i = R$ for some i and $R_i = \check{R}$ for others.

We also differentiate between two types of tamperers:

- An **arbitrary tamperer** can freely corrupt the data store and is not restricted in any way. Most real-life systems fit into this category as they place no restrictions on the tamperer.
- A **destructive tamperer** can only apply a transformation to the store whose range of possible outputs is substantially smaller than the set of inputs. The destructive tamperer can superimpose its own encryption on the common store, transform the store in arbitrary ways, and even add additional data, provided that the cumulative effect of all these operations is to decrease the entropy of the data store. Though a destructive tampering assumption may look like an artificial restriction, it subsumes natural models of block deletion or corruption, and either it or some similar assumption is needed to achieve all-or-nothing integrity in the private-recovery-algorithm model.

An **adversary class** specifies what kind of tamperer $\check{T}$ is and which users, if any, receive $\check{R}$ as their recovery algorithm. Altogether, we consider 6 ($= 3 \times 2$) adversary classes, each corresponding to a combination of constraints on the tamperer and the recovery algorithms.

4 Dependency and All-or-Nothing Integrity

We now give our definition of document dependency for a particular encoding scheme and adversary class. We first discuss some basic definitions and assumptions in Section 4.1. Our strong notions of entanglement, called **dependency** and **all-or-nothing integrity**, are defined formally in Section 4.2.

4.1 Preliminaries

Because we consider protocols involving probabilistic Turing machines, we must be careful in talking about probabilities. Fix an encoding (I, E, R), an adversary $A = (\check{I}, \check{T}, \check{R})$, and the recovery algorithm R_i for each user i. An **execution** of the resulting system specifies the inputs k_i and d_i to E, the output of E, the tamperer's input $\check{k}$ and output $\check{C}$, and the output of the recovery algorithm R_i ($R(k_i, \check{C})$ or $\check{R}(\check{k}, k_i, \check{C})$ as appropriate) for each user. The set of possible executions of the storage system is assigned probabilities in the obvious way: the

probability of an execution is taken over the inputs to the storage system and the coin tosses of the encoding scheme and the adversary. It will be convenient to consider multiple adversaries with a fixed encoding scheme. In this case, we use $\Pr_A(Q)$ to denote the probability that an event Q occurs when A is the adversary.

During an execution of the storage system, the tamperer alters the combined store from C into $\check{C}$. As a result, some users end up recovering their documents while others do not. We summarize which users recover their documents in a **recovery vector**, which is a vector-valued random variable $\boldsymbol{r}$ in which $r_i = 1$ if $R_i(k_i, \check{C}) = d_i$ (i.e., if user i recovers his document) and 0 otherwise. For example, if the server contains documents d_1, d_2, and d_3 and in an execution we recover only d_1 and d_2, then $\boldsymbol{r} = 110$.

4.2 Our Notions of Entanglement

In Section 2, we observed that the block-sharing notion of entanglement provided by Dagster and Tangler is not adequate for our purposes. This motivates us to propose the notion of **document dependency**, which formalizes the idea that "if my data depends on yours, I can't get my data back if you can't." In this way, the fates of specific documents become linked together: specifically, if document d_i depends on document d_j, then whenever d_j cannot be recovered neither can d_i.

Given just one execution, either users i and j each get their data back or they don't. So how can we say that the particular outcome for i depends on the outcome for j? Essentially, we are saying that we are happy with executions in which either j recovers its data (whether or not i does) or in which j does not recover its data and i does not either. Executions in which j does not recover its data but i does are bad executions in this sense. We will try to exclude these bad executions by saying that they either never occur or occur only with very low probability. Formally:

Definition 1. *A document d_i **depends on** a document d_j with respect to a class of adversaries $\mathcal{A}$, denoted $d_i \overset{\mathcal{A}}{\hookrightarrow} d_j$, if, for all adversaries $A \in \mathcal{A}$,*

$$\Pr_A[r_i = 0 \vee r_j = 1] \geq 1 - \epsilon.$$

Remark 1. Hereafter, ϵ refers to a negligible function of the security parameter s [2].

The ultimate form of dependency is **all-or-nothing integrity**. Intuitively, a storage system is all-or-nothing if either every user i recovers his data or no user does:

[2] A function $\epsilon : \mathbb{N} \mapsto (0, 1)$ is **negligible** if for every $c > 0$, for all sufficiently large s, $\epsilon(s) < 1/s^c$. See any standard reference, such as [10], for details.

Definition 2. *A storage system is* ***all-or-nothing*** *with respect to a class of adversaries $\mathcal{A}$ if, for all $A \in \mathcal{A}$,*

$$\Pr_A[\boldsymbol{r} = 0^n \vee \boldsymbol{r} = 1^n] \geq 1 - \epsilon.$$

It is easy to show that

Theorem 3. *A storage system is all-or-nothing with respect to a class of adversaries $\mathcal{A}$ if and only if, for all users i, j, $d_i \overset{\mathcal{A}}{\hookrightarrow} d_j$.*

All-or-nothing integrity is a very strong property. In some models, we may not be able to achieve it, and we will accept a weaker property called **symmetric recovery**. Symmetric recovery requires that all users recover their documents with equal probability:

Definition 3. *A storage system has* ***symmetric recovery*** *with respect to a class of adversaries $\mathcal{A}$ if, for all $A \in \mathcal{A}$ and all users i and j,*

$$\Pr_A[r_i = 1] = \Pr_A[r_j = 1].$$

Symmetric recovery says nothing about what happens in particular executions. For example, it is consistent with the definition for exactly one of the data items to be recovered in every execution, as long as the adversary cannot affect which data item is recovered. This is not as strong a property as all-or-nothing integrity, but it is the best that can be done in some cases.

5 Possibility and Impossibility Results

The possibility of achieving **all-or-nothing integrity** (abbreviated AONI) depends on the class of adversaries we consider. In Sections 5.1 through 5.3, we consider adversaries with an **arbitrary tamperer**. We show that AONI cannot always be achieved in this case. Then in Section 5.4, we look at adversaries with a **destructive tamperer**. We give a simple interpolation scheme that achieves all-or-nothing integrity for a destructive tamperer in all three recovery models.

5.1 Possibility of AONI for Standard-Recovery-Algorithm Model

In the standard-recovery-algorithm model, all users use the standard recovery algorithm R; that is $R_i = R$ for all users i.

This model allows a very simple mechanism for all-or-nothing integrity based on Message Authentication Codes (MACs)[3]. The intuition behind this mechanism is that the encoding algorithm E simply tags the data store with a MAC

[3] Informally, a **MAC** consists of a key generator GEN, a tagging algorithm TAG that associates a tag σ with message m, and a verification algorithm VER that can be used to check if (m, σ) is a valid message/tag pair. A MAC is **existentially unforgeable** under chosen message attacks if the adversary cannot forge a valid message/tag pair even after interacting polynomially many times with a signing oracle (see [11] for details).

using a key known to all the users, and the recovery algorithm R returns an individual user's data only if the MAC on the entire database is valid.

We now give an encoding scheme (I, E, R) based on a MAC scheme (GEN, TAG, VER):

Initialization. The initialization algorithm I computes $k_{MAC} = GEN(1^s)$. It then returns an encoding key $k_E = k_{MAC}$ and recovery keys $k_i = (i, k_{MAC})$.

Entanglement. The encoding algorithm E generates an n-tuple $m = (d_1, d_2, \ldots, d_n)$ and returns $C = (m, \sigma)$ where $\sigma = TAG(k_{MAC}, m)$.

Recovery. The standard recovery algorithm R takes as input a key $k_i = (i, k_{MAC})$ and the (possibly modified) store $\check{C} = (\check{m}, \check{\sigma})$. It returns $\check{m}_i$ if $VER(k_{MAC}, \check{m}, \check{\sigma}) =$ accept and returns a default value $\perp$ otherwise.

The following theorem states that this encoding scheme achieves all-or-nothing integrity with standard recovery algorithms:

Theorem 4. *Let (GEN, TAG, VER) be a MAC scheme that is existentially unforgeable against chosen message attacks, and let (I, E, R) be an encoding scheme based on this MAC scheme as above. Let $\mathcal{A}$ be the class of adversaries that does not provide non-standard recovery algorithms $\check{R}$. Then there exists some minimum s_0 such that for any security parameter $s \geq s_0$ and any inputs $d_1, \ldots, d_n$ with $\sum |d_i| \leq s$, (I, E, R) is all-or-nothing with respect to $\mathcal{A}$.*

5.2 Impossibility of AONI for Public and Private-Recovery-Algorithm Models

In both these models, the adversary modifies the common store and distributes a non-standard recovery algorithm $\check{R}$ to the users (either to all users or only to a few select accomplices). Let us begin by showing that all-or-nothing integrity cannot be achieved consistently in either case:

Theorem 5. *For any encoding scheme (I, E, R), if $\mathcal{A}$ is the class of adversaries providing non-standard recovery algorithms $\check{R}$, then (I, E, R) is not all-or-nothing with respect to $\mathcal{A}$.*

The essential idea of the proof is to have the adversary supply a defective recovery algorithm $\check{R}$ that fails randomly with probability $1/n$. This yields a constant probability converging to $1/e$ that some document is not returned, while all others are.

This proof is rather trivial, which suggests that letting the adversary substitute an error-prone recovery algorithm in place of the standard one gives the adversary far too much power. But it is not at all clear how to restrict the model to allow the adversary to provide an improved recovery algorithm without allowing for this particular attack. Allowing users to apply the original recovery algorithm R can be defeated by superencrypting the data store and burying the

decryption key in the error-prone $\check{R}$; defeating this attack would require analyzing $\check{R}$ to undo the superencryption and remove the errors, a task that is likely to be difficult in practice[4].

On the other hand, we do not know of any general mechanism to ensure that no useful information can be gleaned from $\check{R}$, and it is not out of the question that there is an encoding so transparent that no superencryption can disguise it for sufficiently large inputs, given that both $\check{R}$ and the adversary's key $\check{k}$ are public.

5.3 Possibility of Symmetric Recovery for Public-Recovery-Algorithm Model

As we have seen, if we place no restrictions on the tamperer, it becomes impossible to achieve all-or-nothing integrity in the public-recovery-algorithm model. We now show that we can still achieve symmetric recovery.

Because we cannot prevent mass destruction of data, we will settle for preventing targeted destruction. The basic intuition is that if the encoding process is symmetric with respect to permutations of the data, then neither the adversary nor its partner, the non-standard recovery algorithm, can distinguish between different inputs. Symmetry in the encoding algorithm is not difficult to achieve and basically requires not including any positional information in the keys or the representation of data in the common store. One example of a symmetric encoding is a trivial mechanism that tags each input d_i with a random k_i and then stores a sequence of (d_i, k_i) pairs in random order.

Symmetry in the data is a stronger requirement. We assume that users' documents d_i are independent and identically distributed (i.i.d.) random variables. If documents are not i.i.d (in particular, if they are fixed), we can use a simple trick to make them appear i.i.d.: Each user i picks a small number r_i independently and uniformly at random, remembers the number, and computes $d'_i = d_i \oplus G(r_i)$, where G is a pseudorandom generator. The new d'_i are also uniform and independent (and thus computationally indistinguishable from i.i.d.). The users can then store documents d'_i $(1 \leq i \leq n)$ instead of the original documents d_i. To recover d_i, user i would retrieve d'_i from the server and compute $d_i = d'_i \oplus G(r_i)$.

We shall need a formal definition of symmetric encodings:

Definition 4. *An encoding scheme (I, E, R) is **symmetric** if, for any s and n, any inputs $d_1, d_2, \ldots d_n$, and any permutation π of the indices 1 through n, if the joint distribution of $k_1, k_2, \ldots, k_n$ and C in executions with user inputs $d_1, d_2, \ldots d_n$ is equal to the joint distribution of $k_{\pi_1}, k_{\pi_2}, \ldots, k_{\pi_n}$ and C in executions with user inputs $d_{\pi_1}, d_{\pi_2}, \ldots d_{\pi_n}$.*

[4] Whether it is difficult from a theoretical perspective depends on how well $\check{R}$ can be obfuscated; though obfuscation is impossible in general [3], recovering useful information from $\check{R}$ is likely to be difficult in practice, especially if the random choice to decrypt incorrectly is not a single if-then test but is the result of accumulating error distributed throughout its computation.

Using this definition, it is easy to show that any symmetric encoding gives symmetric recovery:

Theorem 6. *Let (I, E, R) be a symmetric encoding scheme. Let $\mathcal{A}$ be a class of adversaries as in Theorem 5. Fix s and n, and let $d_1, \ldots, d_n$ be random variables that are independent and identically distributed. Then (I, E, R) has symmetric recovery with respect to $\mathcal{A}$.*

5.4 Possibility of AONI for Destructive Adversaries

Given the results of the previous sections, to achieve all-or-nothing integrity we need to place some additional restrictions on the adversary.

A tampering algorithm $\check{T}$ is **destructive** if the range of $\check{T}$ when applied to an input domain of m distinct possible data stores has size less than m. The amount of destructiveness is measured in bits: if the range of $\check{T}$ when applied to a domain of size m has size r, then $\check{T}$ destroys $\lg m - \lg r$ bits of entropy. Note that it is not necessarily the case that the outputs of $\check{T}$ are smaller than its inputs; it is enough that there be fewer of them.

Below, we describe a particular encoding, based on polynomial interpolation, with the property that after a sufficiently destructive tampering, the probability that *any* recovery algorithm can reconstruct a particular d_i is small. While this is trivially true for an unrestrained tamperer that destroys all $\lg m$ bits of the common store, our scheme requires only that with n documents the tamperer destroy slightly more than $n \lg(n/\epsilon)$ bits before the probability that any of the data can be recovered drops below ϵ (a formal statement of this result is found in Corollary 1). Since n counts only the number of users and not the size of the data, for a fixed population of users the number of bits that can be destroyed before all users lose their data is effectively a constant independent of the size of the store being tampered with.

The encoding scheme is as follows. It assumes that each data item can be encoded as an element of Z_p, where p is a prime of roughly s bits.

Initialization. The initialization algorithm I chooses $k_1, k_2, \ldots k_n$ independently and uniformly at random *without replacement* from Z_p. It sets $k_E = (k_1, k_2, \ldots, k_n)$ and then returns $k_E, k_1, \ldots k_n$.

Entanglement. The encoding algorithm E computes, using Lagrange interpolation, the coefficients $c_{n-1}, c_{n-2}, \ldots c_0$ of the unique degree $(n-1)$ polynomial f over Z_p with the property that $f(k_i) = d_i$ for each i. It returns $C = (c_{n-1}, c_{n-2}, \ldots c_0)$.

Recovery. The standard recovery algorithm R returns $f(k_i)$, where f is the polynomial whose coefficients are given by C.

Intuitively, the reason the tamperer cannot remove too much entropy without destroying all data is that it cannot identify which points $d = f(k)$ correspond to actual user keys. When it maps two polynomials f_1 and f_2 to the same corrupted store $\check{C}$, the best that the non-standard recovery algorithm can do is return one of $f_1(k_i)$ or $f_2(k_i)$ given a particular key k_i. But if too many polynomials

are mapped to the same $\check{C}$, the odds that $\check{R}$ returns the value of the correct polynomial will be small.

A complication is that a particularly clever adversary could look for polynomials whose values overlap; if $f_1(k) = f_2(k)$, it doesn't matter which f the recovery algorithm picks. But here we can use that fact that two degree $(n-1)$ polynomials cannot overlap in more than $(n-1)$ places without being equal. This limits how much packing the adversary can do.

As in Theorem 6, we assume that the user inputs $d_1, \ldots, d_n$ are chosen independently and have identical distributions. We make a further assumption that each d_i is chosen uniformly from Z_p. This is necessary to ensure that the resulting polynomials span the full p^n possibilities[5].

Theorem 7. *Let (I, E, R) be defined as above. Let $A = (\check{I}, \check{T}, \check{R})$ be an adversary where $\check{T}$ is destructive: for a fixed input size and security parameter, there is a constant M such that for each key $\check{k}$,*

$$|\{\check{T}(\check{k}, f)\}| \leq M,$$

where f ranges over the possible store values, i.e. over all degree-$(n-1)$ polynomials over Z_p. If the d_i are drawn independently and uniformly from Z_p, then the probability that at least one user i recovers d_i using $\check{R}$ is

$$\Pr_A[\boldsymbol{r} \neq 0^n] < \frac{2n^2 + nM^{1/n}}{p},$$

even if all users use $\check{R}$ as their recovery algorithm.

We can use Theorem 7 to compute the limit on how much information the tamperer can remove before recovering any of the data becomes impossible:

Corollary 1. *Let (I, E, R) and $(\check{I}, \check{T}, \check{R})$ be as in Theorem 7. Let $\epsilon > 0$ and let $p > 4n^3/\epsilon$. If for any fixed $\check{k}$, tamperer $\check{T}$ destroys at least $n \lg(n/\epsilon) + 1$ bits of entropy, then*

$$\Pr_A[\boldsymbol{r} = 0^n] \geq 1 - \epsilon.$$

6 Conclusion and Future Work

Our results are summarized below:

	Destructive Tamperer	Arbitrary Tamperer
Standard Recovery	all-or-nothing	all-or-nothing
Public Recovery	all-or-nothing	symmetric recovery
Private Recovery	all-or-nothing	no guarantees possible

[5] The assumption that the documents are i.i.d. does not constrain the applicability of our results much, because the technique to get rid of it described in Section 5.2 can also be used here.

They show that it is possible in principle to achieve all-or-nothing integrity with only mild restrictions on the adversary. Whether it is possible in practice is a different question. Our model abstracts away most of the details of the storage and recovery processes, which hides undesirable features of our algorithms such as the need to process all data being stored simultaneously and the need to read every bit of the data store to recover any data item. Some of these undesirable features could be removed with a more sophisticated model, such as a round-based model that treated data as arriving over time, allowing combining algorithms that would touch less of the data store for each storage or retrieval operation at the cost of making fewer documents depend on each other. The resulting system might look like a variant of Dagster or Tangler with stronger mechanisms for entanglement. But such a model might permit more dangerous attacks if the adversary is allowed to tamper with data during storage, and finding the right balance between providing useful guarantees and modeling realistic attacks will be necessary. We have made a first step towards this goal in the present work, but much still remains to be done.

References

1. R. J. Anderson. The eternity service. In *Proceedings of PRAGOCRYPT 96*, pages 242–252, 1996.
2. J. Aspnes, J. Feigenbaum, A. Yampolskiy, and S. Zhong. Towards a theory of data entanglement. Technical Report YALEU/DCS/TR-1277, March 2004. Available at `http://www.cs.yale.edu/~aspnes/entanglement-abstract.html`.
3. B. Barak, O. Goldreich, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology - Proceedings of CRYPTO 2001*, 2001.
4. M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, pages 173–186, 1999.
5. I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed information storage and retrieval system. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 46–66, 2000.
6. K. Fu, F. Kaashoek, and D. Mazieres. Fast and secure distributed read-only file system. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, pages 181–196, 2000.
7. G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A cost-effective, high-bandwidth storage architecture. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 92–103, 1998.
8. E. Goh, H. Shacham, N. Mdadugu, and D. Boneh. Sirius: Securing remote untrusted storage. In *Proceedings of the Internet Society (ISOC) Network and Distributed Systems Security (NDSS) Symposium*, pages 131–145, 2003.
9. A. Goldberg and P. Yianilos. Towards an archival intermemory. In *Proceedings of the IEEE International Forum on Research and Technology, Advances in Digital Libraries (ADL '98)*, pages 147–156. IEEE Computer Society, 1998.

10. S. Goldwasser and M. Bellare. Lecture notes on cryptography. Summer Course "Cryptography and Computer Security" at MIT, 1996–1999, 1999.
11. S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen message attack. *SIAM Journal on Computing*, 17(2):281–308, 1988.
12. U. Maheshwari and R. Vingralek. How to build a trusted database system on untrusted storage. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, pages 135–150, 2000.
13. D. Mazieres and D. Shasha. Don't trust your file server. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems*, pages 99–104, 2001.
14. D. Mazieres and D. Shasha. Building secure file systems out of Byzantine storage. In *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing*, pages 108–117, 2002.
15. D. Mazieres and M. Waldman. Tangler : A censorship-resistant publishing system based on document entanglements. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 126–135, 2001.
16. R. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Security and Privacy*, pages 122–134, 1980.
17. Mojo Nation. Technology overview.
Online at `http://www.mojonation.net/docs/technical_overview.shtml`, 2000.
18. R. Rivest. All-or-nothing encryption and the package transform. In *Fast Software Encryption*, volume 1267 of *Lecture Notes in Computer Science*, pages 210–218, 1997.
19. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
20. J. Strunk, G. Goodson, M. Scheinholtz, C. Soules, and G. Ganger. Self-securing storage: Protecting data in compromised systems. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, pages 165–180, 2000.
21. A. Stubblefield and D. S. Wallach. Dagster: Censorship-resistant publishing without replication. Technical Report TR01-380, Rice University, 2001.
22. M. Waldman and D. Mazieres. Tangler: A censorship-resistant publishing system based on document entanglements. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 126–135, 2001.
23. M. Waldman, A. Rubin, and L. Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proceedings of 9th USENIX Security Symposium*, 2000.