Yale University
Department of Computer Science

Towards a Theory of Data Entanglement

James Aspnes [1] Joan Feigenbaum [2]
Aleksandr Yampolskiy [3] Sheng Zhong [4]

YALEU/DCS/TR-1277
March 26, 2004

This work was supported by the DoD University Research Initiative (URI) administered by the Office of Naval Research under Grant N00014-01-1-0795.

[1]Department of Computer Science, Yale University, New Haven, CT 06520-8285, USA. Email: aspnes@cs.yale.edu. Supported in part by NSF.

[2]Department of Computer Science, Yale University, New Haven, CT 06520-8285, USA. Email: joan.feigenbaum@yale.edu. Supported in part by NSF and ONR.

[3]Department of Computer Science, Yale University, New Haven, CT 06520-8285, USA. Email: aleksandr.yampolskiy@yale.edu. Supported by NSF.

[4]Department of Computer Science, Yale University, New Haven, CT 06520-8285, USA. Email: sheng.zhong@yale.edu. Supported by NSF.

Towards a Theory of Data Entanglement

James Aspnes * Joan Feigenbaum † Aleksandr Yampolskiy ‡ Sheng Zhong §

Abstract

We propose a formal model for data entanglement as used in storage systems like Dagster [25] and Tangler [26]. These systems split data into blocks in such a way that a single block becomes a part of several documents; these documents are said to be **entangled**. Dagster and Tangler use entanglement in conjunction with other techniques to deter a censor from tampering with unpopular data. In this paper, we assume that entanglement is a goal in itself. We measure the strength of a system by how thoroughly documents are entangled with one another and how attempting to remove a document affects the other documents in the system. We argue that while Dagster and Tangler achieve their stated goals, they do not achieve ours. In particular, we prove that deleting a typical document in Dagster affects, on average, only a constant number of other documents; in Tangler, it affects virtually no other documents. This motivates us to propose two stronger notions of entanglement, called **dependency** and **all-or-nothing integrity**. All-or-nothing integrity binds the users' data so that it is hard to delete or modify the data of any one user without damaging the data of all users. We study these notions in six submodels, differentiated by the choice of users' recovery algorithms and restrictions placed on the adversary. In each of these models, we not only provide mechanisms for limiting the damage done by the adversary, but also argue, under reasonable cryptographic assumptions, that no stronger mechanisms are possible.

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?

To convince you that you will not lose your data at my random whim, I might offer stronger technical guarantees. Two storage systems proposed linking the fate of your data to the data of other users: Dagster [25] and Tangler [26]. The intuition behind these systems is that data are

This work was supported by the DoD University Research Initiative (URI) administered by the Office of Naval Research under Grant N00014-01-1-0795.

*Department of Computer Science, Yale University, New Haven, CT 06520-8285, USA. Email: aspnes@cs.yale.edu. Supported in part by NSF.

†Department of Computer Science, Yale University, New Haven, CT 06520-8285, USA. Email: joan.feigenbaum@yale.edu. Supported in part by NSF and ONR.

‡Department of Computer Science, Yale University, New Haven, CT 06520-8285, USA. Email: aleksandr.yampolskiy@yale.edu. Supported by NSF.

§Department of Computer Science, Yale University, New Haven, CT 06520-8285, USA. Email: sheng.zhong@yale.edu. Supported by NSF.

partitioned into blocks in a way that every block can be used to reconstruct 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 [22] (Tangler). Two documents that share a server block are said to be **entangled**.

Entangling new documents with old ones provides an incentive to retain blocks, as the loss of a particular block might render many important documents inaccessible. Dagster and Tangler use entanglement as one of many mechanisms to discourage negligent or malicious destruction of data; others involve disguising the ownership and contents of documents and (in Tangler) storing documents redundantly. The present work is motivated in part by the question of whether these additional mechanisms are necessary, or whether entanglement by itself can act as an insurmountable barrier to malicious censorship.

We begin by analyzing the use of entanglement in Dagster and Tangler in Section 2. We argue that the notion of entanglement provided by Dagster and Tangler is not by itself sufficiently strong to discourage censorship by punishing data loss, as not enough documents get deleted on average if an adversary destroys a block for some targeted document. In particular, we show in Section 2.3 that destroying a typical document in Dagster requires destroying only a constant number of additional documents on average, even if the adversary is restricted to the very limited attack of deleting a single block chosen uniformly at random from the blocks that make up the document. The situation with Tangler is worse: because Tangler uses 3-out-of-4 secret sharing [22] to store blocks, deleting two blocks from a particular document destroys the document without destroying any others (which lose at most one block each) in the typical case.

These properties of Dagster and Tangler arise from the particular strategies used to partition documents among blocks, which are determined partly by other goals not related to entanglement: the desire to maintain scalability by using only a constant number of existing blocks in both systems and the desire to protect documents against non-targeted faults in Tangler. We can easily imagine modified systems that increase the level of entanglement by sacrificing these goals. However, it will still be the case that such systems are vulnerable to more sophisticated attacks than simply deleting individual blocks (we describe some of these attacks later). Furthermore, the simple block-sharing notion of entanglement used in these systems creates only a tenuous connection between the survival of individual documents. Although it is often the case that destroying one document will destroy some others, there are no specific other documents that will necessarily be destroyed. Thus Dagster and Tangler must rely on additional mechanisms to discourage deletions.

Our objective in the present work is to examine the possibility of obtaining stronger notions of entanglement, in which the fates of specific documents are directly tied together. These stronger notions might be enough by themselves to deter censorship, in that destroying a particular document, even if done by a very sophisticated adversary, could require destroying most or all of the other documents in the system. A system that provides such tight binding between documents gives a weak form of censorship resistance; though we cannot guarantee that no documents will be lost, we can guarantee that no documents will be lost unless the adversary burns down the library. Under the assumption that the adversary can destroy data at will, this may be the best guarantee we can hope to offer.

In Section 3, we define our model for a document-storage system in which the adversary is allowed to modify the common data store after all documents have been stored. Such modifications may include the block-deletion attacks that Dagster and Tangler are designed to resist, but they may also include more sophisticated attacks such as replacing parts of the store with modified data, or superencrypting all or part of the store.

In addition to modifying the data store, in some variants of the model the adversary is permitted to carry out what we call an **upgrade attack** (see Section 3.2), in which the adversary offers all interested users the choice of adopting a new algorithm to recover their data from the common store if they find that the old one no longer works. Allowing such upgrade attacks is motivated by the observation that a selfish user will jump at the chance to get his data back (especially if he has the ability to distinguish genuine from false data) if the alternative is losing the data. Upgrade attacks also exclude dependency mechanisms that rely on excessive fastidiousness in the recovery algorithm, as one might see in a recovery algorithm that politely declines to return its user's data if it detects that some other user's data has been lost.

In Section 4, we propose two stronger notions of entanglement in the context of our model: **document dependency** and **all-or-nothing integrity**. If a document of user A depends on a document of user B, then A can recover her document only if B can. Unlike entanglement, document dependency is an externally-visible property of a system; it does not require knowing how the system stores its data, but only observing which documents can still be recovered after an attack. All-or-nothing integrity is the ultimate form of document dependency, in which every user's document depends on every other user's, so that either every user gets his data back or no user does. Our stronger notions imply Dagster's and Tangler's notion of entanglement but also ensure that an adversary cannot delete a document without much more serious costs.

The main part of the present work examines what security guarantees can be provided depending on the assumptions made in the model. In Section 5, we consider how the possibility or impossibility of providing document dependency depends on the choice of permitted adversary attacks. Section 5.1 shows the property, hinted at above, that detecting tampering using a MAC suffices for obtaining all-or-nothing integrity if all users use a standard "polite" recovery algorithm. Section 5.2 shows that all-or-nothing integrity can no longer be achieved for an unrestricted adversary if we allow upgrade attacks. In Section 5.3, we show how to obtain a weaker guarantee that we call **symmetric recovery** even with the most powerful adversary; here, each document is equally likely to be destroyed by any attack on the data store. This approach models systems that attempt to prevent censorship by disguising where documents are stored. In Section 5.4, we show that it is possible to achieve all-or-nothing integrity despite upgrade attacks if the adversary can only modify the common data store in a way that destroys entropy, a generalization of the block-deleting attacks considered in Dagster and Tangler.

In Section 6, we discuss some related work on untrusted storage systems.

Finally, in Section 7, we discuss the strengths and limitations of our approach, and offer suggestions for future work in this area.

2 Dagster and Tangler

We review how Dagster [25] and Tangler [26] work in Sections 2.1 and 2.2, respectively. We describe these systems at the block level and omit details of how they break a document into blocks and assemble blocks back into a document. In Section 2.3, we analyze the intuitive notion of entanglement provided by these systems, pointing out some of this notion's shortcomings if (unlike in Dagster and Tangler) it is the only mechanism provided to deter censorship. This motivates us to search for 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 Dagster server enumerates the blocks in a Merkle hash tree [17], which makes it easy for the users to verify integrity of any server block. The tree stores the actual data at its leaves, and each internal node contains a hash of its children. Each document in Dagster consists of $c+1$ server blocks: c blocks of older documents and one new block, which is an exclusive-or of previous blocks with the encrypted document. The storage protocol proceeds as follows:

Initialization Upon startup, the server creates approximately $1,000$ random server blocks and adds them to the system.

Entanglement To publish document d_i, user i generates a random key k_i. He then chooses c random server blocks $C_{i_1}, \ldots, C_{i_c}$ and computes a new block

$$C_i = \mathcal{E}_{k_i}(d_i) \oplus \bigoplus_{j=1\ldots c} C_{i_j},$$

where $\mathcal{E}$ is a plaintext-aware encryption function[1] and $\oplus$ is bitwise exclusive-or.

The user periodically queries the hash tree until he is convinced that block C_i was selected by other users. He then releases instructions on recovering d_i, which come in the form of a **Dagster Resource Locator** (DRL), which is a list of hashes of blocks needed to reconstruct d_i:

$$\Big(k_i, \pi\left[H(C_i), H(C_{i_1}), H(C_{i_2}), \ldots, H(C_{i_c})\right]\Big).$$

Here $H(\cdot)$ is a cryptographic hash function and π is a random permutation.

Recovery To recover d_i, the user asks the server for blocks with hashes listed in the DRL of d_i. If the hashes of the blocks returned by the server match the ones in the DRL, the user computes:

$$\mathcal{D}_{k_i}\left(C_i \oplus \bigoplus_{j=1\ldots c} C_{i_j}\right),$$

where $\mathcal{D}$ represents a decryption function. Otherwise, the user exits.

2.2 Tangler

The Tangler storage system employs a network of servers. It derives its name from the use of (3, 4) Shamir secret sharing (see [22] for details) 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 are replicated across a subset of Tangler servers, which is uniquely determined from blocks' cryptographic hashes. A data structure similar to Dagster Resource Locator, called an **inode**, is used to record the hashes of the blocks needed to reconstruct the document. Here is the storage protocol:

Initialization As in Dagster, the server is jump-started with a bunch of random blocks.

[1] A plaintext-aware encryption function is one for which it is computationally infeasible to generate a valid ciphertext without knowing the corresponding plaintext. See [3] for a formal definition.

Entanglement The server blocks in Tangler play the role of Shamir shares. Each block is a pair (x, y), where x is used as an argument to a polynomial over $GF(2^{16})$ and y is the value of the polynomial at x. The polynomial is uniquely determined by any three blocks, and it is constructed in a way that evaluating it at zero yields the actual data.

To publish document d_i, user i downloads two random server blocks: $C_{i_1} = (x_1, y_1)$ and $C_{i_2} = (x_2, y_2)$. He interpolates these blocks together with $(0, d_i)$ to form a quadratic polynomial $f(\cdot)$. Evaluating $f(\cdot)$ at two different nonzero integers produces new server blocks: C'_{i_1} and C'_{i_2}. The user uploads the new blocks to the server. Finally, he adds the hashes of the blocks needed to reconstruct d_i (*viz.*, the old blocks C_{i_1}, C_{i_2} and the new blocks C'_{i_1}, C'_{i_2}) to d_i's inode.

Recovery To recover his document, user i sends a request for blocks listed in d_i's inode to a subset of Tangler servers. Upon receiving three of d_i's blocks, the user can reconstruct $f(\cdot)$ and compute $d_i = f(0)$.

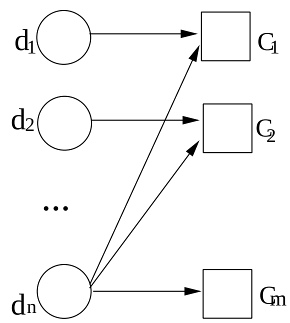

Figure 1: An entanglement graph is a bipartite graph from the set of documents to the set of server blocks. An edge (d_j, C_k) is in the graph if server block C_k can be used to reconstruct document d_j.

2.3 Analysis of entanglement

Let us take a "snapshot" of the contents of a Dagster or Tangler server. The server contains a set of blocks $\{C_1, \ldots, C_m\}$ that comprise 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 depict this documents-blocks relationship using an **entanglement graph** (see Figure 1). The graph contains an edge (d_j, C_k) if block C_k can be used to reconstruct document d_j. Note that even if the graph contains (d_j, C_k), it may still be possible to reconstruct d_j from other 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 four. Entangled documents share one or more server blocks. In Figure 1, documents d_1 and d_n are entangled because they share server block C_1; meanwhile, documents d_1 and d_2 are not entangled.

This notion of entanglement, provided by Dagster and Tangler, 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, owned by some Very Important Person, may give solace to the owner of d_n (refer to Figure 1), who might assume that no adversary would dare incur the wrath of the VIP merely to destroy d_n. However, as depicted in the figure, the adversary can still delete server blocks C_2 and C_m and corrupt d_n but not d_1.

Moreover, the user does not get to choose the documents to be entangled with his document; these documents are chosen randomly. While destroying a user's document is likely to destroy some others, there are no specific other documents that will necessarily be destroyed. If few other documents get destroyed on average, the small risk of accidentally corrupting an important document will be unlikely to deter the adversary from tampering with data.

We now derive an upper bound on how many documents get destroyed if we delete a random document from a Dagster or Tangler server. Intuitively, the earlier the document was uploaded onto the server, the more documents it is entangled with and the more other documents will get destroyed. Without loss of generality, we can assume that documents are numbered in the order in which they were uploaded; namely, for all $1 \leq j < n$, document d_j was uploaded to the server before d_{j+1}. We consider a restricted adversary, who randomly chooses several blocks of d_j (one block in Dagster; two in Tangler) and overwrites them with zeroes.

2.3.1 Deleting a targeted document

First, we show the expected side-effects of deleting the j-th document; in the following section we use this to calculate the effects of deleting a document chosen uniformly at random.

Theorem 1 *In a Dagster server with $n_0 = O(1)$ initial blocks and n documents, where each document is linked with c pre-existing blocks, deleting a random block of document d_j $(1 \leq j \leq n)$ destroys*

$$O\left(c \log\left(\frac{n}{j}\right)\right)$$

other documents.

Proof: There are altogether $m = n_0 + n$ blocks stored on the server: n_0 initial blocks and n data blocks. We label the data blocks $C_1, \ldots, C_n$. The initial blocks exist on the server before any data blocks have been added. We label them $C_{-n_0+1}, \ldots, C_0$.

Every document d_j consists of c pre-existing blocks[2] and a data block C_j that is computed during the entanglement stage. Consider an adversary who destroys a random block C_i of d_j. This will destroy d_j, but it will also destroy any documents with edges outgoing to C_i in the entanglement graph. We would like to compute the number of such documents, N_i.

If C_i is a data block (*i.e.*, $i \geq 1$), then

[2]These may be either initial blocks or data blocks of documents added earlier than d_j (*i.e.*, d_k with $k < j$).

$$\begin{aligned}
E[N_i] &= \sum_{k=i}^{n} \Pr[d_k \text{ has an edge to } C_i] \\
&= 1 + \sum_{k=i+1}^{n} \left(1 - \binom{k-2+n_0}{c} \Big/ \binom{k-1+n_0}{c}\right) \\
&< 1 + c \sum_{j=i+n_0}^{n+n_0-1} \frac{1}{j} \\
&= 1 + c(H_{n+n_0-1} - H_{i+n_0-1}) \\
&= 1 + \Theta\left(c \log \frac{n+n_0-1}{i+n_0-1}\right) \\
&= O\left(c \log\left(\frac{n}{i}\right)\right) \text{ under the assumption that } n_0 \text{ is a constant.}
\end{aligned}$$

Meanwhile, if C_i is an initial block (*i.e.*, $i < 1$), it can be linked by any of the documents:

$$\begin{aligned}
E[N_i] &= \sum_{k=1}^{n} \Pr[d_k \text{ has an edge to } C_i] \\
&= O(c \log n).
\end{aligned}$$

The number of documents deleted on average when the adversary destroys a *random* block of d_j is

$$\begin{aligned}
N_{avg} &= \frac{1}{j+n_0} \cdot \left(\sum_{i=-n_0}^{j} E[N_i]\right) \\
&< \frac{1}{j} \cdot \left(O(c \log n) + \sum_{i=1}^{j} O\left(c \log\left(\frac{n}{i}\right)\right)\right).
\end{aligned}$$

We can use Stirling's formula to bound the leading term in (1):

$$\begin{aligned}
\sum_{i=1}^{j} O\left(c \log\left(\frac{n}{i}\right)\right) &= O\left(c \log\left(\prod_{i=1}^{j} \frac{n}{i}\right)\right) \\
&= O\left(cj \log\left(\frac{n}{j/e}\right)\right) \\
&= O\left(cj \log\left(\frac{n}{j}\right)\right).
\end{aligned}$$

The theorem follows.

■

Theorem 2 *In a Tangler server with $n_0 = O(1)$ initial blocks and n documents, deleting two random blocks of document d_j $(1 \le j \le n)$ destroys*

$$O\left(\frac{1}{j}\right)$$

other documents.

Proof: The server contains $m = n_0 + 2n$ blocks, n_0 of which are initial and $2n$ are data blocks. We label the blocks as in the proof of Theorem 1. The initial blocks are $C_{-n_0+1}, \ldots, C_0$ and the data blocks are $C_1, C_2, \ldots, C_{2n-1}, C_{2n}$.

In Tangler, every document d_j consists of two *old* blocks of pre-existing documents and two *new* blocks C_{2j-1} and C_{2j}, computed during the entanglement stage. Suppose an adversary deletes any two out of four blocks comprising d_j; call these blocks C_i, C_t. Then any document d_k $(k \neq j)$ that contains both C_i and C_t (*viz.*, has edges outgoing to C_i and C_t in the entanglement graph), will also get destroyed. We would like to compute the number of such documents, N_{avg}.

In our analysis, we consider whether deleted blocks C_i, C_t are new or old to d_j and d_k. We distinguish between five cases:

Case 1: C_i, C_t are old to both d_j and d_k. Then the number of deleted documents is

$$\sum_{k=1}^{j-1} \frac{1}{\binom{2j-2+n_0}{2}} + \sum_{k=j+1}^{n} \frac{1}{\binom{2k-2+n_0}{2}}.$$

Case 2: C_i, C_t are old to d_j. However, only one of them is old to d_k, while the other is new to d_k. Note that, in this case, we must have $k < j$.

$$\sum_{k=1}^{j-1} \frac{4}{\binom{2j-2+n_0}{2}}.$$

Case 3: C_i, C_t are old to d_j, but new to d_k. Note that, in this case, we must also have $k < j$.

$$\sum_{k=1}^{j-1} \frac{1}{\binom{2j-2+n_0}{2}}.$$

Case 4: One of C_i, C_t is old to d_j, the other is new to d_j. Note that both C_i and C_t must be old to d_k (because otherwise we would have $k < j$, which implies that the block in C_i, C_t that is new to d_j is not linked to d_k, which further implies that d_k will not get deleted).

$$\sum_{k=j+1}^{n} \frac{4}{\binom{2k-2+n_0}{2}}.$$

Case 5: C_i, C_t are new to d_j. In this case, both C_i and C_t must be old to d_k (for the same reason as in Case 4).

$$\sum_{k=j+1}^{n} \frac{1}{\binom{2k-2+n_0}{2}}.$$

Summing up the five cases, gives us the total number of documents destroyed:

$$
\begin{aligned}
N_{avg} &= \frac{6(j-1)}{\binom{2j-2+n_0}{2}} + \sum_{k=j+1}^{n} \frac{6}{\binom{2k-2+n_0}{2}} \\
&< \frac{6}{2j-3} + \sum_{k=j+1}^{n} \frac{3}{(k-\frac{3}{2})^2} \\
&< \frac{6}{2j-3} + \int_{j}^{n} \frac{3}{(x-\frac{3}{2})^2}\, dx \\
&= \frac{12}{2j-3} - \frac{6}{2n-3} \\
&= O\left(\frac{1}{j}\right) \text{ for large } n.
\end{aligned}
$$

∎

2.3.2 Deleting a random document

Using Theorem 1, we can show that destroying a typical document in Dagster affects few other documents on average:

Corollary 3 *In a Dagster server, deleting a block of a random document destroys on average $O(c)$ other documents.*

Proof: Suppose the server contains n documents, each document linked with c server blocks. According to Theorem 1, deleting document d_j $(1 \le j \le n)$ affects

$$
O\left(c \log\left(\frac{n}{j}\right)\right)
$$

other documents.

Therefore, deleting a typical d_j will affect

$$
\begin{aligned}
\frac{1}{n}\sum_{j=1}^{n} O\left(c\log\left(\frac{n}{j}\right)\right) &= O\left(\frac{c}{n}\log\left(\prod_{j=1}^{n}\frac{n}{j}\right)\right) \\
&= O\left(\frac{c}{n}\log\left(\frac{n^n}{(n/e)^n}\right)\right) \\
&= O(c)
\end{aligned}
$$

documents. ∎

Intuitively, one might expect the corollary to follow immediately from the fact that each document depends on c blocks, which means (when n is large enough that the number of blocks and documents are proportional) that the average block is used by $\Theta(c)$ documents. While such an argument does tell us what happens if the adversary deletes a block chosen uniformly at random, choosing a document uniformly at random and then choosing a block from that document biases the choice of block toward earlier blocks that are used in more documents; nonetheless, the corollary shows that this bias provides at most a constant increase in the destructive effect.

Meanwhile, Theorem 2 tells us that destroying a document d_j from the Tangler server affects $O(1/j)$ other documents on average. From this bound it is immediate that:

Corollary 4 *In a Tangler server, deleting two blocks of a random document destroys on average* $O\left(\frac{\log n}{n}\right)$ *other documents.*

We illustrate the limited effectiveness of this form of entanglement by itself as a censorship deterrent with a simple example:

Example: *Suppose there are $n = 500$ documents stored on a Dagster server, where each document is linked with $c = 15$ blocks. Further, suppose five of the documents belong to a VIP. From Corollary 3, we know that deleting a block of a typical document destroys roughly 15 other documents. The probability that one of the destroyed documents will belong to the VIP is*

$$1 - \frac{\binom{495}{15}}{\binom{500}{15}} \approx \frac{1}{7}.$$

The adversary has a six in seven chance of deleting a typical document without affecting the VIP's data. $\square$

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 modeling systems such as Dagster and Tangler that entangle data. Specializing the general framework gives three specific system models, differentiated by the choice of recovery algorithms. We discuss these models, along with different kinds of adversaries that we consider, in Section 3.2.

As we have mentioned, Dagster and Tangler use many worthwhile techniques in conjunction with entanglement to provide censorship-resistance. Our model abstracts away many such details of storage and recovery processes. Instead, we concentrate on a single entanglement operation performed by these systems. An entanglement operation takes documents of a finite group of users and intertwines these documents to form a common store. The server contents are computed as an aggregation of common stores from multiple entanglement operations. We number the group's users $\{1, \ldots, n\}$ and assume every user i possesses a document d_i that he wants to publish.[3] With the

[3]In practice, a censorship-resistant system should provide some degree of anonymity to its users. The numbers we assign to users play the role of pseudonyms. While the adversary may be able to distinguish documents owned by user i from documents owned by user j, he should not be able to tell who the real people behind the pseudonyms are. Further, if the adversary becomes biased against user i and decides to delete all his data, our mechanisms will ensure that the adversary will also destroy the data of many other users. We refer the reader to [21] for further discussion of anonymizing censorship-resistant systems.

help of the model, we study the security provided by entanglement operation in and of itself. Our model can potentially be expanded, albeit at the expense of significantly increased complexity. One such extension could be a multiround model which would look at multiple entanglement operations rather than just one operation. We discuss some possible extensions to our model in Section 7.

3.1 Basic framework

Our 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.

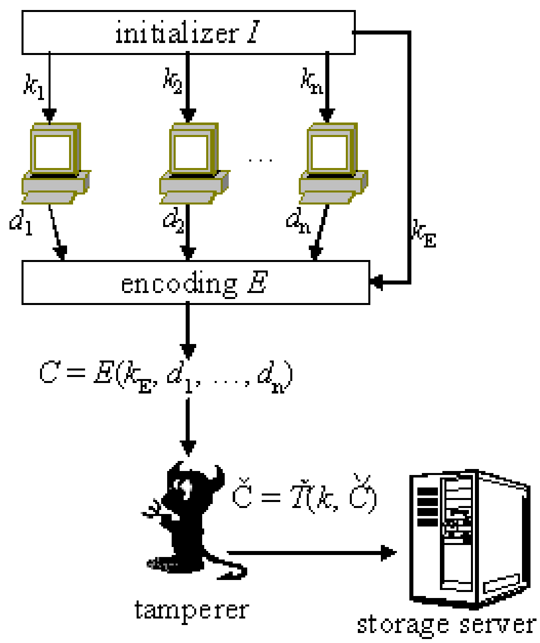

Figure 2: Initialization, entanglement, and tampering stages.

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. We also assume the security parameter s is chosen large enough that the number of users is polynomial in s.

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 model

We divide our model on two axes: one bounding the ability of the reconstruction algorithms and the other bounding the ability of the adversary 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 adopted by 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.[4]

  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 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.

  For example, such an adversary might erase parts of the data store without being able to selectively modify the store's contents, a restriction that could be implemented using write-once media. Though requiring only destructive tampering may look like an artificial restriction, we will need it to achieve all-or-nothing integrity in the private-recovery-algorithm model.

Recall that an adversary is a tuple $(\check{I}, \check{T}, \check{R})$, which consists of an initializer $\check{I}$, a tamperer $\check{T}$, and a non-standard recovery algorithm $\check{R}$. 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 (= 3×2) adversary classes, each corresponding to a combination of aforementioned constraints on the tamperer and the recovery algorithms. We shall see that the possibility of achieving all-or-nothing integrity ultimately depends on the adversary class we consider.

4 Dependency and all-or-nothing integrity

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

[4]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 further discuss this issue in Section 5.2.

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. The **recovery vector** of an execution specifies which documents were successfully recovered in that execution. Formally:

Definition 5 *The **recovery vector** of execution α, denoted $\vec{\rho}(\alpha)$, is a bit vector*

$$\vec{\rho}(\alpha) = (\rho_1, \rho_2, \ldots, \rho_n),$$

where

$$\rho_i = \begin{cases} 1 & \textit{if } R_i(k_i, \check{C}) = d_i, \\ 0 & \textit{otherwise} \end{cases}$$

To illustrate, suppose the server contains three documents: d_1, d_2, and d_3. If in execution α we recover only documents d_1 and d_2, we then have: $\vec{\rho}(\alpha) = 110$.

When we think of α as a random variable (having fixed a particular encoding algorithm and adversary), we will use $\vec{r}$ as a shorthand for the random variable $\vec{\rho}(\alpha)$.

4.2 Our notions of entanglement

In Section 2, we observed that the block-sharing notion of entanglement provided by Dagster and Tangler does not by itself provide strong security guarantees. Two documents may be entangled together in this sense even though it is still possible to delete one of them without affecting the other. Block-sharing entanglement also makes sense only in the context of a system that uses block-sharing and requires observing the internal workings of the system. We would like a definition that applies to a much wider class of potential systems, which requires considering only externally-observable properties.

This motivates us to propose the notion of **document dependency**. Document dependency 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 ether. 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, in terms of a recovery vector, this becomes:

Definition 6 *We say that d_i **depends on** 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: Hereafter, ϵ refers to a negligible function in the security parameter s.[5]

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:

Definition 7 *A storage system is **all-or-nothing** with respect to a class of adversaries $\mathcal{A}$ if, for all $A \in \mathcal{A}$,*

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

It is perhaps not surprising that we have the all-or-nothing property if and only if all documents depend on each other:

Lemma 8 *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, $d_i \overset{\mathcal{A}}{\hookrightarrow} d_j$.*

Proof: Fix an adversary in $\mathcal{A}$. Let E be the event that an execution of the storage system is not all-or-nothing, and F_{ij} the event that document d_i was recovered in an execution and d_j was not. Then $E = \{\vec{r} \neq 0^n \wedge \vec{r} \neq 1^n\}$ and $F_{ij} = \{r_i = 1 \wedge r_j = 0\}$.

($\Rightarrow$) : If the system is all-or-nothing, then $\Pr[E] < \epsilon$. Clearly, for all i, j, we have $F_{ij} \subseteq E$, which means $\Pr[F_{ij}] \leq \Pr[E] < \epsilon$. This in turn implies $d_i \overset{\mathcal{A}}{\hookrightarrow} d_j$.

($\Leftarrow$) : If for all i, j, $d_i \overset{\mathcal{A}}{\hookrightarrow} d_j$, then $\Pr[F_{ij}] < \epsilon$. We can choose $\epsilon < \epsilon'/n^2$ for a negligible ϵ'. Notice that $E \subseteq \bigcup_{i,j} F_{ij}$. Therefore, $\Pr[E] \leq \sum_{i,j} \Pr[F_{ij}] < n^2\epsilon < \epsilon'$. Hence, $\Pr[E^c] \geq 1 - \epsilon'$ and so the storage system is all-or-nothing.

∎

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 9 *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] 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 [11], for details.

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

- In the **standard-recovery-algorithm model**, a simple application of Message Authentication Codes (MACs) achieves all-or-nothing integrity.
- In the **public-recovery-algorithm model**, all-or-nothing integrity is no longer possible. The best that can be done is to prevent the adversary from targeting specific users by hiding the location of each user's data within the store.
- In the **private-recovery-algorithm model**, even the weak guarantees of the public-recovery-algorithm model are no longer possible, because the adversary can **superencrypt** the data store and refuse to distribute the decryption key to users he doesn't like.

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. Both Dagster and Tangler assume this model.

This model allows a very simple mechanism for all-or-nothing integrity based on Message Authentication Codes (MACs). The intuition behind this mechanism is that the encoding algorithm E simply tags the data store with a MAC 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 begin by recalling some standard definitions [12]:

Definition 10 *A **MAC** consists of three algorithms (GEN, TAG, VER) such that:*

1. *A **key generator** $GEN : 1^s \mapsto k_{MAC}$, generates an s-bit key on input a security parameter s (encoded in unary);*
2. *A **tagging algorithm** $TAG : k_{MAC}, m \mapsto \sigma$ computes a MAC σ for any message m with $|m| \leq s^c$ for some fixed c; and*
3. *A **verification algorithm** $VER : k_{MAC}, m, \sigma \mapsto \{accept, reject\}$, with the property that $VER(k_{MAC}, m, TAG(k_{MAC}, m)) = accept$ for all m.*

Definition 11 *A MAC scheme is said to be **existentially unforgeable** against **chosen message attacks** if there is no polynomial-time forger F that generates a new message-MAC pair (m', σ') that is accepted by VER with probability exceeding $O(s^{-c})$ for any $c > 0$, even if F is given a sample of valid message-signature pairs (m_i, σ_i), where m_i is chosen by the adversary. Here the requirement that (m', σ') is "new" means simply that m' is not equal to any of the supplied m_i.*

We now show that MACs that are existentially unforgeable against chosen message attacks give all-or-nothing integrity with standard recovery algorithms. The encoding scheme is as follows:

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 is all-or-nothing:

Theorem 12 *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 $\mathcal{A}$.*

Proof: Fix some $c > 0$. Recall that the adversary changes the combined store from $C = (m, \sigma)$ to $\check{C} = (\check{m}, \check{\sigma})$. We consider two cases, depending on whether or not $\check{m} = m$.

In the first case, $\check{m} = m$. Suppose $R(k_i, \check{C}) = d_i$ but $R(k_j, \check{C}) \neq d_j$. Then $R(k_j, \check{C}) = \perp$, which implies that $V(k_{MAC}, m, \check{\sigma}) \neq$ accept when computed by $R(k_j, \check{C})$ and thus that $\check{\sigma} \neq \sigma$. But $R(k_i, \check{C}) = d_i$ only if $V(k_{MAC}, m, \check{\sigma}) =$ accept when computed by $R(k_i, \check{C})$. It follows that $(m, \check{\sigma})$ is a message-MAC pair not equal to (m, σ) that V accepts in the execution of $R(k_i, \check{C})$; by the security assumption this occurs for a particular execution of V only with probability $O(s^{-c'})$ for any fixed c'. If we choose c' and s_0 so that the $O(s^{-c'})$ term is smaller than $\frac{1}{2n}s^{-c}$ for $s \geq s_0$, then the probability that *any* of the n executions of V in the recovery stage accepts $(m, \check{\sigma})$ in some case where $m = \check{m}$, is bounded by $\frac{1}{2}s^{-c}$.

In the second case, $m \neq \check{m}$. Now $(\check{m}, \check{\sigma})$ is a message-MAC pair not equal to (m, σ). If every execution of V rejects $(\check{m}, \check{\sigma})$, then all $R(d_i, \check{C})$ return $\perp$ and the execution has a recovery vector 0^n. The only bad case is when at least one execution of V erroneously accepts $(\check{m}, \check{\sigma})$. But using the security assumption and choosing c', s_0 as in the previous case, we again have that the probability that V accepts $(\check{m}, \check{\sigma})$ in any of the n executions of R is at most $\frac{1}{2}s^{-c}$.

Summing the probabilities of the two bad cases gives us the desired bound: $\Pr_A[\vec{r} = 0^n \vee \vec{r} = 1^n] > 1 - s^{-c}$. ∎

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. In the public-recovery-algorithm model, all users get the new $\check{R}$, whereas in the private-recovery-algorithm model, $\check{R}$ is given only to a few select adversary's buddies. Let us begin by showing that all-or-nothing integrity cannot be achieved consistently in either case:

Theorem 13 *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 $\mathcal{A}$.*

Proof: Let the adversary initializer $\check{I}$ be a no-op and let the tamperer $\check{T}$ be the identity transformation. We will rely entirely on the non-standard recovery algorithm to destroy all-or-nothing integrity.

Let $\check{R}$ flip a biased coin that comes up tails with probability $1/n$, and return the result of running R on its input if the coin comes up heads and $\perp$ if the coin comes up tails. Then exactly one document is not returned with probability $n \cdot (1/n) \cdot (1 - 1/n)^{n-1}$, which converges to $1/e$ in the limit. Because this document is equally likely to be any of the n documents by symmetry, we get each of the recovery vectors described in the theorem with (non-negligible) probability $1/en$.

The outcome is all-or-nothing only if all instances of $\check{R}$ flip the same way, which occurs with probability $\Pr_A[\vec{r} = 0^n \vee \vec{r} = 1^n] < 1 - 1/en$. ∎

The proof of Theorem 13 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.

One possibility would be to allow users to choose between applying the original recovery algorithm and the adversary's new and improved version; but in practice this approach is easily defeated by a tamperer $\check{T}$ that encrypts C (which renders $\check{C}$ unusable as input to R) coupled with an error-prone $\check{R}$ that reverses the encryption (when its coin comes up heads) before applying R.

A more sophisticated approach would be to allow R to analyze $\check{R}$ to attempt to undo whatever superencryption may have been performed and extract a recovery algorithm that works all the time. Unfortunately, this approach depends on being able to extract useful information about the workings of an arbitrary Turing machine. While it has been shown that program obfuscation is impossible in general [2], even in a specialized form this operation is likely to be very difficult, especially if the random choice to decrypt incorrectly is not a single if-then test but is the result of accumulating error distributed throughout the computation of $\check{R}$.

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 symmetric encodings 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. For the moment, 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 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 ($1 \le i \le n$) 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 14 *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 $d_{\pi_1}, d_{\pi_2}, \ldots d_{\pi_n}$.*

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

Theorem 15 *Let (I, E, R) be a symmetric encoding scheme. Let $\mathcal{A}$ be a class of adversaries as in Theorem 13. 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 $\mathcal{A}$.*

Proof: Fix i and j. From Definition 14 we have that the joint distribution of the k_i and C is symmetric with respect to permutation of the user indices; in particular, for any fixed d, S and x,

$$\Pr[C = S, k_i = x \,|\, d_i = d] = \Pr[C = S, k_j = x \,|\, d_j = d]. \tag{1}$$

We also have, from the assumption that the d_i are i.i.d.,

$$\Pr[d_i = d] = \Pr[d_j = d]. \tag{2}$$

Using (1) and (2), we get

$$\begin{aligned}
&\Pr[\check{R}(\check{k}, k_i, \check{T}(C)) = d_i] \\
&= \sum_{x,S,d} \Pr[\check{R}(\check{k}, x, \check{T}(S)) = d] \Pr[C = S, k_i = x, d_i = d] \\
&= \sum_{x,S,d} \Pr[\check{R}(\check{k}, x, \check{T}(S)) = d] \Pr[C = S, k_i = x \,|\, d_i = d] \Pr[d_i = d] \\
&= \sum_{x,S,d} \Pr[\check{R}(\check{k}, x, \check{T}(S)) = d] \Pr[C = S, k_j = x \,|\, d_j = d] \Pr[d_j = d] \\
&= \Pr[\check{R}(\check{k}, k_j, \check{C}) = d_j].
\end{aligned}$$

This is simply another way of writing $\Pr_A[r_i = 1] = \Pr_A[r_j = 1]$. ∎

5.4 Possibility of AONI for destructive adversaries

Unfortunately, neither all-or-nothing integrity nor symmetric recovery can be achieved in the private-recovery-algorithm model for an arbitrary tamperer. The adversary can always superencrypt the data store and distribute a useless recovery algorithm that refuses to return the data. We need to place some additional restrictions on the adversary.

Definition 16 *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.*

Remark: 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 detailed statement of this result is found in Corollary 18). 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 can't overlap in more than $(n-1)$ places without being equal. This limits how much packing the adversary can do.

As in Theorem 15, 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. Recall that, in Section 5.2, we argued how to get rid of the assumption that the d_is are i.i.d. That argument can also be used here to get rid of the assumption that each d_i is independent and uniform.

Under these conditions, sufficiently destructive tampering prevents recovery of any information with high probability. We will show an accurate but inconvenient bound on this probability in Theorem 17 and give a cruder but more useful statement of the bound in Corollary 18.

Theorem 17 *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

$$\Pr_A[\vec{r} \neq 0^n] < \frac{2n^2 + nM^{1/n}}{p}, \tag{3}$$

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

Proof: Condition on $\check{k}$ and the outcome of all coin-flips used by $\check{T}$ and $\check{R}$. Then, there are exactly $p^n \binom{p}{n}$ possible executions, each of equal probability, determined by the p^n choices for the d_i and the $\binom{p}{n}$ choices for the k_i. For each i, we will show that the number of these executions in which $\check{R}(\check{k}, k_i, \check{C}) = d_i$ is small.

For each degree-$(n-1)$ polynomial f, define f^* to be the function mapping each k in Z_p to $\check{R}(\check{k}, k, \check{T}(\check{k}, f))$. Note that f^* is deterministic given that we are conditioning on $\check{k}$ and all coin-flips in $\check{T}$ and $\check{R}$. Define C_f, the **correct inputs** for f, to be the set of keys k for which $f(k) = f^*(k)$.

The adversary produces a correct output only if at least one of the n user keys appears in C_f. For a given f, the probability that *none* of the keys appear in C_f is

$$\begin{aligned}
\frac{\binom{p-|C_f|}{n}}{\binom{p}{n}} &> \frac{(p-|C_f|-n)^n}{p^n} \\
&= \left(1 - \frac{|C_f|+n}{p}\right)^n \\
&> 1 - \frac{n(|C_f|+n)}{p},
\end{aligned}$$

and so the probability that *at least one* key appears in C_f is at most $\frac{n}{p}|C_f| + \frac{n^2}{p}$. Averaging over all f then gives

$$\Pr\left[f^*(k_i) = d_i \text{ for at least one } i\right] < \frac{n^2}{p} + \frac{n}{p^{n+1}} \sum_f |C_f|. \tag{4}$$

We will now use the bound on the number of distinct f^* to show that $\sum_f |C_f|$ is small.

Consider the set of all polynomials $f_1, f_2, \ldots f_m$ that map to a single function f^*, and their corresponding sets of correct keys $C_{f_1}, C_{f_2}, \ldots C_{f_m}$. Because any two degree $(n-1)$ polynomials are equal if they are equal on any n elements of Z_p, each n-element subset of Z_p can be contained in at most one of the C_{f_i}. On the other hand, each C_{f_i} contains exactly $\binom{|C_{f_i}|}{n}$ subsets of size n. Since there only $\binom{p}{n}$ subsets of size n to partition between the C_{f_i}, we have

$$\sum_i \binom{|C_{f_i}|}{n} \leq \binom{p}{n},$$

and summing over all M choices of f^* then gives

$$\sum_f \binom{|C_f|}{n} \leq M \binom{p}{n}.$$

We now wish to bound the maximum possible value of $\sum_f |C_f|$ given this constraint.

Observe that $\binom{|C_f|}{n} > \frac{(|C_f|-n)^n}{n!}$ when $|C_f| \geq n$, from which it follows that

$$\sum_{f:|C_f|\geq n} (|C_f| - n)^n < n! \sum_f \binom{|C_f|}{n} < n! M \binom{p}{n}. \tag{5}$$

Now, $(|C_f|-n)^n$ is a convex function of $|C_f|$, so the left-hand side is minimized for fixed $\sum_f |C_f|$ by setting all $|C_f|$ equal. It follows that $\sum_f |C_f|$ is *maximized* for fixed $\sum_{f:|C_f|\geq n}$ when all $|C_f|$ are equal.

Setting each $|C_f| = c$ and summing over all p^n values of f, we get

$$p^n (c-n)^n < n! M \binom{p}{n}$$

from which it follows that

$$c < \frac{1}{p} \left(n! M \binom{p}{n} \right)^{1/n} + n,$$

and thus that

$$\sum_f |C_f| \leq p^n c < p^{n-1} \left(n! M \binom{p}{n} \right)^{1/n} + np^n.$$

Plugging this bound back into (4) then gives

$$\begin{aligned}
\Pr_A [\vec{r} \neq 0^n] &= \Pr [f^*(k_i) = d_i \text{ for at least one } i] \\
&< \frac{2n^2}{p} + \frac{n}{p^2} \left(n! M \binom{p}{n} \right)^{1/n} \\
&< \frac{2n^2}{p} + \frac{n}{p^2} (M p^n)^{1/n} \\
&= \frac{2n^2 + nM^{1/n}}{p}.
\end{aligned}$$

∎

Using Theorem 17, it is not hard to compute a limit on how much information the tamperer can remove before recovering any of the data becomes impossible:

Corollary 18 *Let* (I, E, R) *and* $(\check{I}, \check{T}, \check{R})$ *be as in Theorem 17. Let* $\epsilon > 0$ *and let* $p > \frac{4n^3}{\epsilon}$. *If for any fixed* $\check{k}$, $\check{T}$ *destroys at least* $n \lg(n/\epsilon) + 1$ *bits of entropy, then*

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

Proof: Let $\epsilon' = \epsilon \,/\, \left(\frac{1}{2n} + 2^{-1/n}\right)$. If $\check{T}$ destroys at least $n \lg(n/\epsilon') + 1$ bits of entropy, then we have

$$M \leq p^n \cdot 2^{-(n \lg(n/\epsilon')+1)} = \frac{1}{2} p^n (n/\epsilon')^{-n} = \frac{1}{2} \left(p\epsilon'/n\right)^n. \tag{6}$$

Plug this into (3) to get:

$$\begin{aligned}
\Pr[\text{some } d_i \text{ is recovered}] &\leq \frac{2n^2 + nM^{1/n}}{p} \\
&\leq \frac{2n^2 + n\left(\frac{1}{2}(p\epsilon'/n)^n\right)^{1/n}}{p} \\
&= \frac{2n^2}{p} + 2^{-1/n}\epsilon' \\
&< \frac{2n^2}{4n^3/\epsilon'} + 2^{-1/n}\epsilon' \\
&= \epsilon' \left(\frac{1}{2n} + 2^{-1/n}\right) \\
&= \epsilon.
\end{aligned}$$

We thus have:

$$\Pr_A[\vec{r} = 0^n] = 1 - \Pr[\text{some } d_i \text{ is recovered}] \geq 1 - \epsilon.$$

∎

6 Related work

The problem of building reliable storage using untrusted servers has been studied for a long time. Previous work has offered several interpretations of what this means. We distinguish between three basic approaches: replication, tamper detection, and entanglement. We also discuss the all-or-nothing transform, an unkeyed cryptographic tool that provides guarantees similar to all-or-nothing integrity in a restricted version of the destructive tampering model.

6.1 Systems based on replication

Anderson, in his seminal paper describing an "Eternity Service" [1], was among the first to propose building a network of tamper-resistant servers, spread around the world. Stored documents would be redundantly replicated across the network, thereby making them **censorship-resistant**: difficult to delete without the cooperation of all servers. Many subsequent storage systems [5, 10, 18] followed in Anderson's footsteps and relied on replication to protect the stored data. One such system, Free Haven [6], uses Rabin's information dispersal algorithm (IDA) [19] to break the document into n shares, any k of which are sufficient to reconstruct the document, which are then distributed to a community of servers. Publius [27] uses a similar idea and splits the key used to encrypt documents into shares that are distributed to different servers. BFS [5] (an abbreviation of "Byzantine File System") also uses replication to develop a storage system that tolerates Byzantine faults of up to 1/3 of the servers.

All these systems solve a problem that is different from, and complementary to, the one considered in this report. We assume that the users treat the storage service as a monolithic entity and cannot distinguish between good and bad servers within a larger unified service. Our reasons for making this assumption are two-fold. First, in most of the systems described above, new servers can join freely, which allows a resourceful and committed adversary to seize control by sending in a few thousand of his best and closest friends. Second, we are interested in guarantees to individual users from the *system as a whole*, without regard to the underlying implementation. Our definition of all-or-nothing integrity applies just as well to distributed storage mechanisms as to centralized ones and reflects the concerns of users who care more about whether they will get their data back than how. Thus, even in a system that promises data availability through replication, all-or-nothing integrity provides additional assurance to the users by guaranteeing that any failure to fulfill this promise will carry a very high cost.

6.2 Systems based on tamper detection

A second approach to providing secure storage is based on detecting tampering. As long as trusted users can detect invalid modifications made to their data, the stored data is considered safe. Two common tools used for tamper detection are digital signatures and Merkle hash trees [17]. Merkle tree stores the actual data at the leaves of the tree, and at each intermediate node is the hash of the concatenation of the data at the previous level. TDB [13], for example, leverages a small amount of trusted memory to store a collision-resistant hash of the entire database. Another storage system, SUNDR [14,15] uses Merkle trees to ensure that all users have a consistent view of the shared data — if the server delays one user from seeing a change by another, the two users would never again see each other's changes. Other storage systems, such as BFS [5], NASD [8], S4 [24], SiRiUS [9], and SFSRO [7], have also relied on hash trees and signatures for tamper detection.

The guarantee provided by these systems is rather weak. A user whose data is lost is likely to notice with or without being notified by the system. However, as we saw in Section 5.1, tamper detection can be leveraged in to give all-or-nothing integrity if all users run standard recovery algorithms by the simple expedient of having all users politely refuse to recover their data if the store is tampered with. That some users might insist on recovering their uncorrupted data anyway points out a fundamental limitation of both the standard-recovery-algorithm model and the tamper detection approach.

6.3 Systems based on entanglement

To prevent impolite users from recovering their own data even if other users' data have been lost, two storage systems have been proposed that create dependencies between blocks of data belonging to different users: Dagster [25] and Tangler [16]. Because of their close connection to our work, we described these systems and gave rigorous analysis of their security properties in Section 2.

	Destructive Tamperer	Arbitrary Tamperer
Standard Recovery	all-or-nothing	all-or-nothing
Public Recovery	all-or-nothing	symmetric recovery
Private Recovery	all-or-nothing	—

Table 1: Summary of results. "All-or-nothing" means that all-or-nothing integrity can be achieved in this model; "symmetric recovery" means that all-nothing integrity cannot be achieved, but symmetric recovery can; "—" means that no guarantees are possible.

6.4 All-or-nothing transforms

Motivated by security problems in block ciphers, Rivest [20] proposed a cryptographic primitive called **all-or-nothing transform** (AONT). An l-AONT is an efficiently computable transformation that satisfies two conditions:

- Its inverse transformation is also efficiently computable.
- If l bits of an image are lost, then it is infeasible to obtain any information about the preimage.

Papers such as [4, 23] further examined Rivest's AONT. AONT is similar to our notion of all-or-nothing integrity in the sense that either all bits of the preimage can be recovered (if the image is available) or none can be (if l bits of the image are lost). However, AONT is radically different from all-or-nothing integrity because it does not involve multiple users, who possess individual keys. Moreover, AONT does not consider the possibility that the image may be corrupted in other ways than some bits being deleted, such as the adversary superencrypting the entire data store.

7 Conclusion and future work

Entangling documents of different users is a promising idea for strengthening the integrity of individual users' data. However, existing systems such as Dagster and Tangler only have an intuitive notion of entanglement that is insufficient by itself to provide the supposedly increased security. In this paper, we rigorously analyze the probability of destroying one document without affecting any other documents in these systems. Our analysis shows that the security they provide is not strong, even if we limit the class of attacks permitted against the entangled data.

Motivated by the desire to improve the security provided by entanglement, we defined the stronger notion of document dependency, in which destroying some document is guaranteed to destroy specific other documents, and all-or-nothing integrity, in which destroying some document is guaranteed to destroy all other documents. We considered a variety of potential attacks and showed for each what level of security was possible. These results are summarized in Table 7; they show that it is possible in principle to achieve all-or-nothing integrity with only limited 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 or 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 treats data as arriving over time, allowing combining algorithms that require using 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. Finding the right balance between providing useful guarantees and modeling realistic attacks will be necessary. We believe that 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] 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.

[3] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology - Proceedings of CRYPTO 98*, 1998.

[4] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-resilient functions and all-or-nothing transforms. In *Advances in Cryptology - Proceedings of EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 453–469, 2000.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] S. Goldwasser and M. Bellare. Lecture notes on cryptography. Summer Course "Cryptography and Computer Security" at MIT, 1996–1999, 1999.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] R. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Security and Privacy*, pages 122–134, 1980.

[18] Mojo Nation. Technology overview. Online at `http://www.mojonation.net/docs/technical_overview.shtml`, 2000.

[19] M. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the Association for Computing Machinery*, 36(2):335–348, 1989.

[20] 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.

[21] Andrei Serjantov. Anonymizing censorship resistant systems. In *Proceedings of the 1st International Peer To Peer Systems Workshop (IPTPS 2002)*, March 2002.

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

[23] D.R. Stinson and T. Trung. Some new results on key distribution patterns and broadcast encryption. *Designs, Codes and Cryptography*, 14(3):261–279, 1998.

[24] 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.

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

[26] 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.

[27] 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.