Non-Interactive Timestamping in the Bounded Storage Model*

Tal Moran† Ronen Shaltiel‡ Amnon Ta-Shma§

July 9, 2008

Abstract

A timestamping scheme is *non-interactive* if a stamper can stamp a document without communicating with any other player. The only communication done is at validation time. Non-Interactive timestamping has many advantages, such as information theoretic privacy and enhanced robustness. Non-Interactive timestamping, however, is not possible against polynomial time adversaries that have *unbounded storage* at their disposal. As a result, no non-interactive timestamping schemes were constructed up to date.

In this paper we show non-interactive timestamping is possible in the bounded storage model. I.e., if the adversary has *bounded storage*, and a long random string is broadcast to all players. To the best of our knowledge, this is the first example of a cryptographic task that is possible in the bounded storage model, but is *impossible* in the "standard cryptographic setting", even when assuming "standard" cryptographic assumptions.

We give an explicit construction that is secure against all bounded storage adversaries, and a significantly more efficient construction secure against all bounded storage adversaries that run in polynomial time.

Keywords: Timestamping, Bounded Storage model, Unbalanced Expander Graphs, Randomness Extractors.

1 Introduction

The date on which a document was created is often a significant issue. Patents, contracts, wills and countless other legal documents critically depend on the date they were signed, drafted, etc. A timestamp for a document provides convincing proof that it existed at a certain time. For physical documents, many methods are known and widely used for timestamping: publication, witnessed signing and placing copies in escrow are among the most common. Techniques for timestamping digital documents, which are increasingly being used to replace their physical counterparts, have also become necessary.

Loosely speaking, a timestamping scheme consists of two mechanisms: A *stamping mechanism* which allows a user to stamp a document at some specific time t, and a *verification mechanism* which allows a recipient to verify at a later time $t' > t$ that the document was indeed stamped at time t.

1.1 Previous Work

Digital timestamping systems were first introduced by Haber and Stornetta [18], where three timestamping systems are described. In the *naïve timestamping protocol*, the stamper sends the document to all the verifiers during timestamp generation. In the *linking* scheme, the stamper sends a one-way hash of the document to a trusted timestamping server. The server holds a *current hash*, which it updates by hashing it with the

*A preliminary version of this paper appeared in CRYPTO 2004 [24].

†Department of Computer Science, Weizmann Institute of Science. Some of this work was done while at Tel-Aviv University.

‡Department of Computer Science, University of Haifa. Some of this work was done while at the Weizmann Institute of Science and supported by the Koshland Scholarship. This research was also supported by Grant No 2004329 from the United States-Israel Binational Science Foundation (BSF) and by ISF grant 686/06.

§Department of Computer Science, Tel-Aviv University. Supported by the Binational Science Foundation, by the Israel Science Foundation, and by the EU Integrated Project QAP.

value sent by the stamper. This links the document to the previous documents and to the succeeding ones. In the *distributed trust* scheme, the document is used to select a subset of verifiers, to which the stamper sends a hash of the document. Bayer, Haber and Stornetta [3] improve upon the linking scheme, reducing the communication and storage requirements of the system and increasing its robustness, by replacing the linear list with a tree. Further work [19, 7, 6, 8, 5, 4] is mainly focused on additional improvements in terms of storage, robustness and reducing the trust required in the timestamping server(s).

One common feature of all the above protocols is that they require the stamper to send messages to a central authority (or a distributed set of servers) at timestamp generation.

1.2 Non-interactive Timestamping

We call a timestamping scheme *non-interactive* if it does not require the stamper to send messages at timestamp generation. Non-interactive timestamping schemes, if they exist, have a number of obvious advantages over interactive schemes. However, the notion of non-interactive timestamping seems self-contradictory. How can we prevent an adversary from faking timestamps, if no action is taken at timestamp generation? More precisely, suppose that an adversary "learns" some document at time $t' > t$ and wants to convince a verifier that he stamped the document at time t. He can simulate the behavior of an "honest stamper" who signs the document at time t and generates a timestamp for the document. Note that the "honest stamper" does not need to send any messages before time t' and therefore the adversary will be able to convince a verifier that the document was stamped at time t.

We can think of the adversary as an honest stamper who was corrupted at some later time. Therefore it is clear that giving the honest stamper secret information and/or allowing "standard" cryptographic assumptions (that some function is infeasible to compute) will not help: anything the honest stamper can do, the adversary must be able to do as well. Even allowing the verifier to send messages to the stamper (but not vice versa) does not invalidate the simulation argument, since the adversary has access to anything the honest stamper received in the past.

A crucial point in the argument above is that in order to perform this simulation the adversary must store all the information available to the "honest stamper" at time t. We show that non-interactive timestamping *is* possible in a scenario in which parties have bounded storage.

1.3 The Bounded Storage Model

In contrast to the usual approach in modern Cryptography, Maurer's *bounded storage model* [22] bounds the *storage* (memory size) of dishonest players rather than their running time.

In a typical protocol in the bounded storage model a long random string r of length R is initially broadcast and the interaction between the polynomial-time participants is conducted based on storing small portions of r. The security of such protocols should be guaranteed even against dishonest parties which have a lot of storage (much more than the honest parties) as long as they cannot store the whole string. Most of the previous work on the bounded storage model concentrated on private key encryption [22, 10, 2, 1, 14, 15, 21, 34], Key Agreement [10] and Oblivious Transfer [9, 12, 13]. In contrast to the tasks above, the notion of non-interactive timestamping cannot be implemented in the "standard cryptographic setting". To the best of our knowledge this is the first example of a protocol in the bounded storage model which achieves a task that is impossible in the "standard cryptographic setting".

1.4 Non-interactive Timestamping in the Bounded Storage Model

We now explain our setting for non-interactive timestamping in the bounded storage model. We assume that there are ℓ rounds and at every round $1 \leq t \leq \ell$, a long random string r of length R is transmitted.[1]

[1] One can imagine that random bits are transmitted at high rate continuously by a trusted party, and that the string r consists of the bits transmitted between time t and time $t + 1$.

The Stamping Mechanism: To stamp a document *doc* at time t, the scheme specifies a function $\mathrm{Stamp}(doc, r)$ whose output is short. To stamp the document *doc*, the stamper stores $\mathrm{Stamp}(doc, r)$. Intuitively, an adversary (who does not know *doc* at time t) is not able to store the relevant information and therefore is unable to stamp *doc*.

The Verification Mechanism: The verifier stores a short "sketch" of r (denoted by $\mathrm{Sketch}(r)$) for every time t. At a later time the stamper can send the timestamp $\mathrm{Stamp}(doc, r)$ and the verifier checks whether this timestamp is "consistent" with his sketch.

Efficiency of a Timestamping Scheme: We say that a timestamping scheme is (T, V)-efficient if the stamper's algorithm runs online (that is, in one pass) using space T and polynomial time and the verifier's algorithm runs online using space V and polynomial time. We want T and V to be small as functions of R.

1.5 Our Notion of Security

Loosely speaking, we want to ensure that even an adversary with a lot of storage (say storage $M = \delta R$ for some constant $\delta < 1$) cannot forge a timestamp. Note, however, that a stamper with storage $M > T$ can easily stamp $k = M/T$ documents by running the stamping mechanism on some k documents and storing the generated timestamps (each of which has length at most T). We will therefore say that a scheme is secure if no adversary with space M can successfully stamp significantly more than M/T documents. More precisely, security is defined with respect to a parameter M^*_{max} which bounds the storage of the most powerful adversary. The scheme is α-optimal (for a parameter $\alpha > 1$) if for every $M \le M_{max}$, no adversary with space M can successfully stamp more than $\alpha\frac{M}{T}$ documents. (The precise definitions appear in Section 3).

Notice that the definition above requires α–optimality for every $M \le M_{max}$. Requiring α–optimality for M_{max} only, would have allowed adversaries with $M \ll M_{max}$ to produce $\frac{M_{max}}{T}$ stamped documents, contradicting the definition's spirit. The definition in its current form assures us that any adversary, weak or strong, with at most M_{max} memory, can honestly stamp *the same number of documents* if given slightly more resources (storage αM instead of M).

Usefulness of our notion of security Let us illustrate the usefulness of this notion by an example. Suppose that there is a random variable V that is uniformly distributed over a large set. Furthermore, suppose that the value of this variable is known only to some parties at time t and is revealed to everybody at time $t' > t$. (This is a formal way of saying that some parties *do not* know the value of V at time t). Note that parties that do know the value v of the variable V at time t can timestamp the document $doc =$ "The value is v". Such a timestamp indeed convinces a verifier that this party knew the value v at time t. This is because a stamper who did not know the value of V at time t will be caught cheating. Specifically, our definition of security says that there exists a set (that is defined in time t) of at most $\alpha\frac{M}{T}$ of documents which the stamper can successfully stamp. Note that the random variable V is uniformly distributed from the point of view of the stamper at time t. Therefore, the probability that a document containing the correct value of V appears in any of the $\alpha\frac{M}{T}$ documents that the adversary can successfully stamp is very small and it follows that no matter how the stamper acts with high probability he will not be able to convince the verifier at time t' that he stamped the document at time t.

We remark that this argument applies if V is not uniformly distributed, but rather "unpredictable" in the sense that it has high min-entropy.

1.6 Our Results

In this paper we give two explicit constructions of non-interactive timestamping schemes in the bounded storage model. The first is secure in an information-theoretic sense (in the spirit of previous constructions in the bounded storage model). It requires no unproven assumptions and is secure against any adversary with arbitrary computational power as long as its storage capability is bounded. We now state this result (precise definitions appear in Section 3 and the Theorem is restated in Theorem 4.2 with some more precise notation).

Theorem 1.1. *For every $\eta > 0$ and large enough R there exists a timestamping scheme that is $(T = O(R^{1/2+\eta}), V = O(R^{1/2+\eta}))$-efficient and $O(1)$-optimal. More precisely, every adversary with space $M^* \leq M^*_{max} = \Omega(R)$ has probability at most $2^{-R^{\Omega(1)}}$ to successfully stamp more than $O(M^*/T)$ documents. The timestamping scheme allows stamping documents of length $R^{\Omega(1)}$.*

Our second system is more efficient. To achieve this efficiency it relies on cryptographic assumptions and is therefore secure only against adversaries that, in addition to being storage bounded, are required to run in polynomial time. (This Theorem is restated with more precise notation in Theorem 5.2).

Theorem 1.2. *Assume that there exist "strong collision resistant hash functions". Then there exists a timestamping scheme that is $(T = 2^{(\log\log R)^{O(1)}}, V = 2^{(\log\log R)^{O(1)}})$-efficient and $O(\log R)$-optimal. More precisely, every adversary with space $M^* \leq M^*_{max} = \Omega(R)$ and running time polynomial in R has negligible probability to successfully stamp more than $O(\log R \cdot M^*/T)$ documents. The timestamping scheme allows stamping documents of length R.*

We remark that our technique can potentially reduce T and V to $\log^{O(1)} R$. This improvement requires an explicit construction of certain "expander graphs" that is not known today. More details appear in Section 5.

Non-malleable timestamping The notion of security defined above makes sense in certain applications. However, this notion does not take into account that a dishonest stamper may see some correct timestamps of related documents before he attempts to timestamp his document. Note that seeing correct timestamps may indeed give the dishonest stamper additional information on the random string which he did not know at time t.

In Section 6 we present stronger security definitions that address this more complicated setup. The high level idea is that we consider a stronger adversary which first views the random string and stores some information about it, then before attempting to timestamp his own document, the adversary is allowed to request to see correct timestamps of few "hint documents" $doc_1, \ldots, doc_w$ that he may choose in any way that he want as a function of his own document. We require that even such an adversary cannot successfully stamp more than $\alpha\frac{M^*}{T}$ documents.

Our definition allows the adversary to choose the hint documents as a function of his own document. (This guarantees that even seeing correct timestamps of related documents does not help the adversary). Furthermore, note that when we require that the adversary can successfully stamp few documents we allow him to choose different hint documents for different documents that he attempts to stamp.

We show that our information theoretic scheme is secure even with this more general security definition. However, to implement our scheme in this setup we need explicit constructions of expander graphs with certain parameters that are currently not known. Therefore, this result does not give an explicit solution.

1.7 Overview of the "Information-Theoretic" Construction

We now give a high level overview of our two constructions. The setup is the following: A string r of length R is transmitted and the stamper wants to convince a verifier that he "knew" a document prior to the transmission of this string.

Using the Document to Select Indices: We implement the function Stamp(doc, r) as follows: Each document doc specifies some D indices that the stamper will remember from the long string. For that we use a bipartite graph where the left-hand vertices are all possible documents, the right-hand vertices are indices $1 \leq i \leq R$ and every left vertex has D neighbors. The indices selected by a document doc are the neighbors of doc. We want to force a stamper who would like to stamp k documents to store many indices. Intuitively, this is equivalent to the requirement that every k documents on the left have many different neighbors. This naturally leads to using an expander graph. (A bipartite graph is a (K, c)-expander if every $k \leq K$ vertices on the left have at least kc neighbors on the right.)[2]

[2]We stress that we need to use *unbalanced graphs* (graphs which have many more vertices on the left than on the right-hand side). Such graphs were constructed in [32, 31, 17]. However, we need graphs with somewhat different parameters. We explain how to derive such graphs from previous work in Section 7.

To stamp a document *doc*, the stamper stores the content of the long string at the indices specified by *doc*. We use graphs with expansion $c \approx D$ which implies that to correctly stamp k documents simultaneously an honest stamper must store roughly kD bits.

Using Random Sets for Verification: The function Sketch(r) is implemented as follows. The verifier chooses a random subset of size $|\mathcal{H}| \approx R/D$ from the indices of r and stores the content of r at these indices. After the transmission of the random string r, a stamper may send a timestamp of a document *doc* (that consists of the content of r at the D indices defined by *doc*). By the birthday problem, with high probability (over the choice of the verifier's random set) some of these indices were also stored by the verifier. The verifier checks that the content sent by the stamper is consistent with what he stored.

For a fixed string r and document *doc*, we say that a timestamp is "incorrect" if it differs from the "correct" timestamp of *doc* in many indices. The verification process we described guarantees that, with high probability, the verifier will reject an "incorrect" timestamp.

A Sketch of the Security Proof: The basic intuition for the security proof is the following: Suppose that an adversary is able to successfully stamp some k documents. This means that he correctly stamped these k documents (as otherwise he is caught by the verifier). However, correctly stamping k documents requires storing kD indices, therefore if the storage of the adversary is $kD \leq M < (k+1)D$ he can successfully stamp at most k documents. This is the best we can hope for (by our notion of security) as he could have stamped k documents by simply running the "stamping mechanism" on any k documents.

We remark that the actual argument given in Section 4 is more complex. This is because we allow the adversary to choose the k documents on which he wants to convince the verifier as a function of the random string r. This complicates the proof. To prove the security of our scheme we use a "reconstruction argument" and show that any adversary that breaks the security guarantee can be used to compress the string r into a shorter string in a way that does not lose a lot of information. As the string r is random, we get a contradiction. The details are given in Section 4.

1.8 Overview of the "Computationally-Secure" Construction

In the previous construction we chose $|\mathcal{H}| \approx R/D$ so that a random subset of size $|\mathcal{H}|$ in $[R]$ would intersect a subset of size D. We chose $|\mathcal{H}| = D \approx \sqrt{R}$, allowing both the honest stamper and the verifier to store only $\sqrt{R}$ bits. We now show how to increase the efficiency and reduce the storage of honest parties to only $2^{(\log\log R)^{O(1)}}$ bits.

We use the same index selection mechanism as before. However, this time we choose $D = 2^{(\log\log R)^{O(1)}}$ (this precise choice of parameters corresponds to certain expander graphs). The verifier stores a short "hash" of the string r. When stamping a document the stamper also supplies a short "proof" that the indices he sent are consistent with the hashed value held by the verifier. We implement such a hashing scheme using *Merkle trees* [23]. We show that if collision resistant hash functions exist then a polynomial time adversary with bounded storage cannot produce an incorrect timestamp of a document. More precisely, we show that after the transmission of the random string r, no polynomial time adversary can *generate* many documents and stamp them correctly.

Hashing Documents Before Stamping Them: A bottleneck of our scheme is that when using expanders of degree D we can only handle documents of length D.[3] However, in a computational setting (as we have already assumed the existence of collision resistant hash functions) we can stamp longer documents by first hashing them to shorter strings and then stamping them.

[3]This is because in unbalanced expander graphs, the degree must be logarithmic in the number of left-hand vertices. Thus, shooting for degree D we can at most get that the left-hand set (which is the set of documents) is of size 2^D.

2 Preliminaries

2.1 Notation

The following conventions will be used throughout the paper.

Random String We refer to the random string as r, its length is denoted by R, and we think of it as composed of N blocks of length n denoted $r_1, \ldots, r_N$. For any subset $S \subseteq [N]$, the expression $r_{|S}$ will be taken to mean the string generated by concatenating the blocks r_i for all $i \in S$.

Hamming Distance The *Hamming Distance* between two strings r_1 and r_2 is the number of *blocks* on which the two strings differ (rather than the number of bits).

Online Space We will be interested in procedures that have small memory and read an enormous stream of bits. We say a function f can be computed *online* with space s if there is an algorithm using space at most s which reads its input bits one by one and computes f in one pass.

2.2 Unbalanced Expander Graphs

A graph is expanding if every sufficiently small set has a lot of neighbors. Our timestamping scheme relies on *unbalanced expanders*.

Definition 2.1 (unbalanced expander graphs)**.** *A bipartite graph $G = (V_1, V_2, E)$ is (K_{max}, c)-expanding if, for any set $S \subset V_1$ of cardinality at most K_{max}, the set of its neighbors $\Gamma(S) \subseteq V_2$ is of size at least $c|S|$.*

Note that we do not require that $|V_1| = |V_2|$. In fact, in our timestamping scheme we will use graphs in which $|V_1| \gg |V_2|$. In this paper we need unbalanced expanders with very specific requirements. Loosely speaking we want a $(K_{max}, \Omega(D))$-expanding graph with as small as possible degree D and right-hand side of size roughly $K_{max}D$. Such graphs are closely related to lossless condensers. In a beautiful work ([17], Theorem 1.1), Guruswami, Umans and Vadhan show how to construct such graphs with small degree $D = polylog(|V_1|/\epsilon)$, and relatively small right hand size, being roughly $K_{max}^{1+\alpha}D^2$ for a small constant α. However, we would like the right hand side to be even smaller, of order $K_{max}D$. Our penalty is that we have a much larger degree. We prove:

Theorem 2.2. *There exists a fixed constant $\beta > 0$ such that for every $K_{max} \leq |V_1|$, there exists a bipartite graph $G = (V_1, V_2, E)$ with left degree D that is $(K_{max}, c = \beta D)$-expanding with $D = 2^{O(\log\log|V_1| + (\log\log K_{max})^3)}$, and $|V_2| = 4\beta K_{max}D$. Furthermore, this graph is explicit in the sense that given a vertex $v \in V_1$ and an integer $1 \leq i \leq D$ one can compute the i'th neighbor of v in time polynomial in $\log|V_1| + \log D$.*

We give the proof of Theorem 2.2 in Section 7.

2.3 Collision Resistant Hash Functions And Merkle Hash Trees

We need the notion of collision resistant hash functions which is implied by standard assumptions on the hardness of factoring integers (see [11, 16]). We sometimes need functions that are secure against adversaries that run in *super-polynomial* time (see e.g. [20]) and therefore we use a parameter for the running time of the adversary:

Definition 2.3 (collision resistant hash functions)**.** *A family of functions $H : \{0,1\}^{2n} \to \{0,1\}^n$ is said to be $(\epsilon_H(n), \mu(n))$-collision resistant if, there is a polynomial time algorithm that, given the description of $h \in H$ and an input $x \in \{0,1\}^{2n}$, outputs $h(x)$. Furthermore, given a random function $h \in H$, no algorithm with running time bounded by $\mu(n)$ can find $x \neq y$ such that $h(x) = h(y)$ with probability greater than $\epsilon_H(n)$ where the probability is over the choice of h and the random coins of the algorithm and n is large enough.*

We use Merkle hash trees [23]. We give a detailed definition so that we can later use the notation in our construction.

Definition 2.4 (Merkle hash tree)**.** *Let $H : \{0,1\}^{2n} \to \{0,1\}^n$ be a $(\epsilon_H(n), \mu(n))$-collision resistant family of hash functions and $(r_1, \ldots, r_N) \in \{0,1\}^n \times \cdots \times \{0,1\}^n$ a vector of binary strings (assume N is a power of 2). Fix a key $h \in H$. The* Merkle tree *with key h of $r_1, \ldots, r_N$, denoted $Tree^h(r_1, \ldots, r_N)$, is a binary tree with N leaves, where each vertex is labelled in the following way:*

- *The i^{th} leaf is labelled with r_i.*
- *An internal vertex v with left child labelled by c_1 and right child labelled by c_2 is labelled with $h(c_1, c_2)$.*

$Root^h(r_1, \ldots, r_N)$ denotes the label of the root vertex.

A Merkle path *from a leaf r_i is an ordered collection of all the nodes on the path from r_i to the root of $Tree^h(r_1, \ldots, r_N)$, along with their siblings in the tree. Let $Path_i^h(r_1, \ldots, r_N)$ denote the labelled Merkle-path from r_i where each node is labelled by the labeling of the Merkle hash tree. That is a labelled Merkle path from a string r_i is consists of a path in the tree and collection of strings $c_1, \ldots, c_t$ that can syntactically be interpreted as a Merkle path for some labelling that has the relevant leaf labelled with r_i.*

A labelled Merkle path P from a leaf r_i is valid *if P labels the root with $Root^h(r_1, \ldots, r_N)$, and for every node v in the path from r_i to the root, P labels v by $h(c_1, c_2)$ where the c_1, c_2 are the labels P assigns to the children of v. (Note that the children of v appear in P). A labelled Merkle path P from a leaf r_i is* correct *for a tree $Tree^h(r_1, \ldots, r_N)$ if its label for r_i is identical with the label of r_i in the tree.*

Clearly, every description of the form $Path_i^h(r_1, \ldots, r_N)$ is both valid and correct for $Tree^h(r_1, \ldots, r_N)$. The usefulness of Merkle hash trees stems from the fact that it is infeasible to find an incorrect valid path.

Definition 2.5. *A Merkle tree construction is* $(\epsilon_H(N, n), \mu(N, n))$-secure *if, for any adversary with running time bounded by $\mu(N, n)$, the probability that it can output a set of leaves $(r_1, \ldots, r_N)$, an index $i \leq N$ and a valid path from r_i that is incorrect for $Tree^h(r_1, \ldots, r_N)$ is less than $\epsilon_H(N, n)$ (where the probability is over the choice of h and the random coins of the algorithm).*

The following standard Lemma observes that collision-resistant hash functions are sufficient to build Merkle Trees (we sometimes set N to be super-polynomial in n, and this is why we need very strong collision resistant hash functions):

Lemma 2.6. *[23] Let $H : \{0,1\}^{2n} \to \{0,1\}^n$ be a $(\epsilon_H(n), \mu(n))$-collision resistant family of hash functions. Then the Merkle tree construction based on H is $\left(\epsilon_H(n), \mu(n) - Nn^{O(1)}\right)$-secure.*

Informally, this is because if we can find two valid but differently labelled paths to the root from the same leaf in the tree, we have found a collision in the hash function. Given an adversary that can find a set of leaves and an incorrect but valid path from one of them to the root, we can find a second valid path by computing the correct path to the root for the same set of leaves (computing this second path can take $O(Nn^{O(1)})$ time, since we may have to compute the hash function for each node of the tree).

Merkle Trees and storage: We now note that certain tasks related with Merkle trees can be done with small storage. To check whether a given path is valid requires a verifier to store h and $Root^h(r_1, \ldots, r_N)$, but not the rest of the tree. To prove that r_i is a leaf in $Tree^h(r_1, \ldots, r_N)$, the prover needs to store only $Path_i^h(r_1, \ldots, r_N)$, which is *logarithmic* in N.

We furthermore note that Merkle paths (and in particular the root of the Merkle tree) can be computed online with small storage when given sequential one-way access to $r_1, \ldots, r_N$. More precisely, assuming that the computation of the hash function can be done for free it is possible to compute Merkle-paths online with space $O(n \log N)$. This is done by computing the paths sequentially and keeping in memory only the roots of the subtrees whose leaves have been seen so far (i.e. after we are given r_1 and r_2, we can compute $h(r_1, r_2)$ and forget the values of r_1 and r_2; in the same way after we have seen $r_1, \ldots, r_{2^k}$ we can remember only the label of the common ancestor for these leaves, forgetting all the nodes in the subtree).

3 One Round Timestamping: The Model

In this section we formally define our model for timestamping in the bounded storage model. In order to simplify the presentation we assume that there are only two time frames. Our model and constructions can be easily adapted to the case of multiple time frames (see Remark 3.5).

We now explain the formal model. A long random string r of length R is transmitted. The verifier takes a short sketch $\mathrm{Sketch}(r)$ of the random string and remembers it. An honest stamper, who wants to stamp a document $doc \in \mathcal{DOC}$, calculates $y = \mathrm{Stamp}(doc, r)$. When, at a later stage, the stamper wants to prove he knew the document doc at stamping time, he sends y to the verifier who computes $\mathrm{Verify}(\mathrm{Sketch}(r), doc, y)$ and decides whether to accept or reject. More formally,

Definition 3.1 (Non-Interactive timestamping scheme)**.** *A non-interactive timestamping scheme consists of three functions:*

- *A stamping function* $\mathrm{Stamp}(doc, r)$*.*
- *A sketch function* $\mathrm{Sketch}(r)$ *(we allow* Sketch *to be a probabilistic function).*
- *A verification function* $\mathrm{Verify}(\mathrm{Sketch}(r), doc, y)$*.*

We require that for every string r and document doc, the function $\mathrm{Verify}(\mathrm{Sketch}(r), doc, \mathrm{Stamp}(doc, r))$ *accepts.*

We define efficiency:

Definition 3.2 (Efficiency)**.** *A non-interactive timestamping scheme is* (T, V)-efficient *if* Stamp *can be computed online in space $T = T(R)$ and time polynomial in R, and* Sketch *can be computed online in space $V = V(R)$ and time polynomial in R.*

An honest stamper with space M can easily stamp M/T documents by running the function Stamp in parallel. We require that no adversary with memory M^* can successfully stamp significantly more than M^*/T documents. We first define our model for adversaries:

Definition 3.3 (Adversary)**.** *An adversary consists of two functions:* $\mathrm{Store}^*(r)$*, which produces a "short" string b, and* $\mathrm{Stamp}^*(doc, b)$ *which, given a document doc and b, attempts to produce a timestamp for doc. The space M^* of an adversary is the maximal length of* $\mathrm{Store}^*(r)$.[4] *An adversary* γ-successfully stamps *a document doc at (some fixed) r if*

$$\Pr[\mathrm{Verify}(\mathrm{Sketch}(r), doc, \mathrm{Stamp}^*(doc, \mathrm{Store}^*(r))) = \ \mathit{Accept}] \geq \gamma$$

where this probability is over the coin tosses of Sketch *and the internal random coins of the adversary. Note that when the adversary is not computationally bounded, we can assume w.l.o.g. that the adversary is deterministic (does not use random coins).*

We denote the set of documents that an adversary A γ-successfully stamps for a given string r by $\mathrm{Successful}^{(A,\gamma)}(r)$*.*

We define security as:

Definition 3.4 (Security)**.** *We say that a (T, V)-efficient timestamping scheme is α-optimal (for $\rho > 0$, $\alpha \geq 1$, $\gamma > 0$ and $M^*_{max} < R$) if for every $M^* \leq M^*_{max}$ and every adversary A with space M^*,*

$$\Pr_r\left[|\mathrm{Successful}^{(A,\gamma)}(r)| > \alpha\frac{M^*}{T}\right] \leq \rho$$

Definition 3.4 is very strong. It guarantees that whenever the sketch size is small, no matter how powerful the adversary is, the number of documents the adversary can successfully stamp is very small. Moreover, any strategy the adversary chooses for his Store^* function can be replaced with a legitimate strategy that uses roughly the same storage and successfully stamps the same number of documents.

We now discuss some more issues that come up in the security definition.

[4] Note that we do not require Store^* to run online in space M^*. Instead, we only restrict the output length of Store^*. Furthermore, note that we place no restrictions on Store^* and in particular do not assume that it is feasibly computable.

Remark 3.5 (Multiple time frames)**.** *The definition stated above has only two time frames: "before" and "after" the random string was transmitted. It can be easily extended to the case of multiple time frames assuming a random string is transmitted at any time frame.*

Remark 3.6 (What does *doc* depend on?)**.** *One thing that is hard to model is how the adversary gets hold of the document doc that it attempts to stamp. We stress that in the definition above the adversary is allowed to choose doc as a function of the random string r. (This can be seen from the fact that the set* Successful *is allowed to depend on r). We remark that the security proofs of the next sections could be significantly simplified if this was not the case. The reason we insist on the stronger notion of security is that it would have been bad if for many random strings r the adversary had many documents that he would be able to successfully stamp.*

Remark 3.7. *Another issue is that the adversary may later on receive additional information about the random string r, beyond the value of* $\mathrm{Store}^*(r)$*. This is because the adversary may see correct timestamps of some other documents. We find the following example helpful. Suppose Alice correctly stamped a patent saying that she is the inventor. When revealing the timestamp it is undesirable that a cheating party Eve can now use this information to produce a timestamp of the same document but for replacing the name "Alice" by "Eve". In Section 6, we suggest a stronger notion of security that attempts to handle such situations.*

3.1 Security Against Time Bounded Adversaries

The definition used above does not bound the computational power of the adversary in any way. The only assumption we made is on the storage capacities of the adversary (which is reflected in the output length of Store^*). We now consider adversaries which are also time bounded.

Definition 3.8 (Time bounded adversary)**.** *We say that an adversary runs in time t if the functions* $\mathrm{Store}^*, \mathrm{Stamp}^*$ *associated with it are randomized machines running in time t.*

We now want to define timestamps that are secure against time bounded adversaries. Recall that an unbounded adversary is considered successful if there *exist* many documents for which he can convince the verifier. When the adversary is time-bounded, it is possible that Definition 3.4 does not hold, yet the system is still secure because it is infeasible to find the documents he can illegally stamp.

This leads us to a weaker notion of security. In this setup an adversary is considered successful it is possible to *feasibly generate* many documents on which the adversary can convince the verifier.

This is captured by requiring that there exists a probabilistic polynomial time machine $\mathrm{Generate}^*_{\mathrm{c}}$ which, on input r and an integer k, outputs k documents (which, intuitively, are the documents the adversary thinks he can succeed on). The formal definition follows:

Definition 3.9 (Security against feasibly generated documents)**.** *We say that a (T, V)-efficient timestamping scheme is α-optimal (for $\rho > 0$, $\alpha \geq 1$, $\gamma > 0$ and $M^*_{max} < R$) against feasibly generated documents, if for every $M^* \leq M^*_{max}$, every adversary A with space M^*, and every machine* $\mathrm{Generate}^*_{\mathrm{c}}(r, k)$ *that runs in time polynomial in the length of r:*

$$\Pr\left[A\ \gamma\text{-}\textit{successfully stamps at } r \textit{ the documents } \mathrm{Generate}^*_{\mathrm{c}}(r, \alpha\tfrac{M^*}{T})\right] \leq \rho$$

where the probability is over the choice of r and the random coins of $\mathrm{Generate}^*_{\mathrm{c}}$ *and A.*

Note that in the definition above we don't explicitly require that the adversary is time bounded, however we will use it mainly for time bounded adversaries. Furthermore, note that there are no storage bounds on $\mathrm{Generate}^*_{\mathrm{c}}$.

4 A Scheme with Information-Theoretic Security

In this section we describe a timestamping scheme which is information theoretically secure against arbitrary adversaries with small storage.

4.1 The Stamping Scheme

Let R, N and n be integers such that $R = N \cdot n$. Given a string $r \in \{0,1\}^R$, we partition it into N blocks of n bits. We use r_i to denote the i'th block of r. Let $\mathcal{DOC}$ denote the set of all documents which can be stamped. Let G be a $(K_{max}, \beta D)$ bipartite expander (V_1, V_2, E) with left degree D, where the "left" set V_1 is $\mathcal{DOC}$ and the "right" set V_2 is $[N]$. We define the three procedures Sketch, Stamp and Verify:

- $\text{Stamp}(doc, r) = r_{|\Gamma(doc)}$. (Recall that $\Gamma(S)$ is the set of neighbors of S).
- $\text{Sketch}(r) = \mathcal{H}, r_{|\mathcal{H}}$ where $\mathcal{H}$ is a multi-set that has $|\mathcal{H}|$ elements selected at random (with repetiton) from $[N]$. (Note that Sketch is probabilistic and relies on internal random coins of the verifier that are used to choose $\mathcal{H}$).
- $\text{Verify}(\text{Sketch}(r), doc, y) = \begin{cases} Accept & \text{Sketch}(r)_{|\mathcal{H}\cap\Gamma(doc)} = y_{|\mathcal{H}\cap\Gamma(doc)} \\ Reject & otherwise \end{cases}$. In words, the adversary compares the content of r in the indices he stored to the indices sent by the stamper and accepts if the two strings are consistent.

Notice that $\text{Sketch}(r)$ contains the restriction of r to the indices of $\mathcal{H}$, and therefore in particular contains the restriction of r to the indices of $\mathcal{H} \cap \Gamma(doc)$, and y contains the restriction of r to $\Gamma(doc)$ and therefore in particular contains the restriction of r to the indices of $\mathcal{H} \cap \Gamma(doc)$.

To assist the reader we include a reference of all the parameters in Table 1. The following Theorem states requirements on the parameters with which the system is secure. It is immediately followed by a corollary in which we plug in specific parameters.

Theorem 4.1. *Let G be a $(K_{max}, \beta D)$-expanding graph with left degree D and let $\gamma > 0$. If $n \geq max(\frac{5}{\beta}, \log N)$ and $\log |\mathcal{DOC}| \leq \beta D n/5$ and $D|\mathcal{H}| \geq \frac{5N}{\beta}\ln(\frac{1}{\gamma})$ then the scheme is $(T = Dn, V = 2|\mathcal{H}|n)$-efficient and α-optimal for $\alpha = \frac{5}{\beta}$, $\rho = 2^{-\beta n D/5}$ and $M^*_{max} = \beta D n K_{max}/5$.*

Plugging in parameters, a corollary of this is the following restatement of Theorem 1.1.

Theorem 4.2. *For every $\eta > 0$ and large enough R we construct a timestamping scheme which is $(T = R^{1/2+\eta}, V = R^{1/2+\eta})$-efficient and $O(1)$-optimal with $\rho = 2^{-R^{\Omega(1)}}$, $\gamma = 2^{-R^{\Omega(1)}}$ and $M^*_{max} = \Omega(R)$. The timestamping scheme allows stamping documents of length $R^{\Omega(1)}$.*

We prove Theorem 4.2 in Subsection 4.6. We now turn to proving Theorem 4.1.

4.2 Efficiency

The verifier first chooses a random set $\mathcal{H}$ and stores it, and then stores $r_{\mathcal{H}}$. This can indeed be done online with space $V = 2|\mathcal{H}|n)$. We now explain how the stamper can run online in space $T = Dn$. Observe that it can calculate the indices it will need to store *before* the random string goes by (since it knows *doc* before it sees the random string). As the indices take $D \log N < Dn$ space, it can work in place, replacing each index with the contents of the block as it goes by. We now turn to proving security.

4.3 Security

The proof below is somewhat technical and involves many parameters and we therefore start with a high level outline.

High level outline of the argument: Our goal is to show that no adversary with space $M^* \leq M^*_{max}$ can successfully stamp more than $k = \alpha M^*/T$ documents. The notion of "successfully stamping" speaks about whether or not the verifier accepts the timestamp. The first stage is to show that whenever the adversary "successfully stamps" a document he must provide a string which is close in Hamming distance to the correct timestamp. In such a case we say that the adversary "correctly stamps" the document. The second stage is to show that an adversary with bounded storage cannot correctly stamp too many documents.

Name	Description	Notes
R	length of the random string in bits	
N	Number of blocks in random string	
n	Length of each block in random string	$R = N \cdot n$. Typically n is very small while N is large.
M^*	The amount of storage of the adversary	$M^* \leq M^*_{max}$ which is a bound on the storage of the most powerful adversary.
G	A bipartite graph over the sets $\mathcal{DOC}$ and $[N]$	
D	Left degree of the graph	
β	The expansion factor of the graph	
K_{max}	The expansion threshold of the graph	Sets of size up to K_{max} expand by a factor of β.
T	Length of timestamp	$T = D \cdot n$
V	The amount of storage of the verifier	$V = \lvert\mathcal{H}\rvert(n + \log N)$ where $\mathcal{H}$ is a random set chosen by the verifier.
γ	The adversary is considered successful on a document if the verifier is convinced with prob $> \gamma$ (over coin tosses of the verifier)	
ρ	Threshold on the probability that the adversary successfully stamps many documents	
α	The optimality factor	An adversary with memory $M^* \leq M^*_{max}$ cannot successfully stamp more than $\alpha M^*/T$ documents.
g	The fraction of errors above which the verifier catches the adversary	This is an internal parameter usually set to 1/5.
k	Internal parameter for number of documents	Set to $\alpha M^*/T$ in the actual proof.
$\mathcal{H}$	A multi-set of randmly chosen indices	Chosen by the verifier.

Table 1: Parameters used in the construction

The first stage in our plan is relatively simple. Recall that the set H is chosen by the verifier in a way that is independent of the adversary's actions. If the adversary provides a timestamp which is far in Hamming distance than the correct one, then with high probability (over the verifier's coins) some of the incorrect indices will be selected by the verifier and the verifier will not accept the timestamp. The first stage follows.

We now turn our attention to the second phase. Our security definitions allow the adversary to attempt to stamp different k documents for different strings r. Nevertheless, for the sake of simplicity let us first rule out an adversary for which there is a set S (that does not depend on r) of k documents such the adversary can successfully stamp all of them on any choice of r.

By the expansion properties of the graph we have that $|\Gamma(S)| \geq \beta D k$ indices which translate into $\beta D n k = \beta T k = \beta \alpha M^*$ bits. Thus, choosing α slightly larger than $1/\beta$ one gets that the documents in S require more storage than is available to the adversary.

In the general case the adversary may choose the set S as a function of r (see Remark 3.6). This technically complicates the argument for the second phase. What we show is that an adversary that successfully stamps many documents using short storage can be translated into a "compression scheme" that can compress the string r into a shorter string and that there is a decompression scheme that decompresses correctly with high probability. It follows that such adversaries do not exist. The remainder of the section gives the formal proofs.

We now start implementing the plan described above. Without loss of generality, we assume the adversary is deterministic (this can be done as the adversary is computationally unbounded). Fix some adversary and let $\mathcal{R}_{\text{successful}}(k) = \mathcal{R}^{\gamma}_{\text{successful}}(k)$ denote the set of strings r on which the adversary γ-successfully stamps at least k documents. We would like to prove that $\mathcal{R}_{\text{successful}}$ has small probability. We first define a notion of "correct stamping":

Definition 4.3. *An adversary* correctly stamps *a document doc at r if* $\text{Stamp}^*(doc, \text{Store}^*(r)) = r_{|\Gamma(doc)}$. *An adversary* correctly stamps *a document doc at r with at most err errors, if the Hamming distance between* $\text{Stamp}^*(doc, \text{Store}^*(r))$ *and* $r_{|\Gamma(doc)}$ *is at most err.*

Let $g = 1/5$ and let $\mathcal{R}_{\text{correct}}(k) = \mathcal{R}^{\text{err}=\text{g}\beta\text{D}}_{\text{correct}}(k)$ denote the set of random strings r for which there are at least k documents that the adversary correctly stamps with at most $err = g\beta D$ errors.

The security proof has two parts. We first show that if the adversary successfully stamps many documents then he correctly stamps many documents. That is, we relate $\mathcal{R}_{\text{correct}}$ and $\mathcal{R}_{\text{successful}}$.

Lemma 4.4. *Assume $D|\mathcal{H}| \geq \frac{N}{g\beta}\ln(\frac{1}{\gamma})$. For every k, $\mathcal{R}_{\text{successful}}(k) \subseteq \mathcal{R}_{\text{correct}}(k)$.*

The proof of Lemma 4.4 appears in Section 4.4. The next Lemma states that any adversary with small space cannot correctly stamp many documents.

Lemma 4.5. *Assume $\log|\mathcal{DOC}| \leq g\beta Dn$, $err = g\beta D$ and $n \geq \frac{1}{g\cdot\beta}$. For every $k \leq K_{max}$ and any adversary with space $M^* \leq (1-4g)k\beta Dn$ we have $\Pr_r[r \in \mathcal{R}_{\text{correct}}(k)] \leq 2^{-g\beta nDk}$.*

The proof of Lemma 4.5 appears in Section 4.5. We now show that the two lemmata give the proof of Theorem 4.1:

Proof. (of Theorem 4.1) We need to show that no adversary with space M^* can γ-successfully stamp more than $k = \alpha\frac{M^*}{Dn}$ documents. Notice that for $M^* \leq M^*_{max}$ it follows that $k = \frac{\alpha M^*}{Dn} \leq \frac{\alpha M^*_{max}}{Dn} = K_{max}$ and $M^* = \frac{kDn}{\alpha} = kDn(1-4g)\beta$. Hence, $\Pr_r[r \in \mathcal{R}_{\text{successful}}(k)] \leq \Pr_r[r \in \mathcal{R}_{\text{correct}}(k)] \leq 2^{-g\beta nDk} \leq 2^{-g\beta nD}$ where the first inequality follows by Lemma 4.4 and the second inequality follows by Lemma 4.5. The third inequality is because $k \geq 1$. ☐

4.4 The Proof of Lemma 4.4

Claim 4.6. *Fix an adversary, a string r and a document doc. If the adversary γ-successfully stamps doc at r then it correctly stamps doc at r with at most $\frac{N}{|\mathcal{H}|}\ln(\frac{1}{\gamma})$ errors.*

Proof. We prove the contrapositive. Suppose for some $doc \in \mathcal{DOC}$ and r, the timestamp provided by the adversary for doc has $err^* > err$ incorrect indices. Denote by $\mathcal{BAD}_{doc} \subset [N]$ the set of incorrect indices. The verifier catches the adversary iff $\mathcal{BAD}_{doc} \cap \mathcal{H} \neq \emptyset$, i.e. if one of the incorrect indices is in $\mathcal{H}$ (the set of indices stored by Sketch). For each index in $\mathcal{H}$, the probability that it hits $\mathcal{BAD}_{doc}$ is $\frac{err^*}{N}$, and the probability that none of them hits $\mathcal{BAD}_{doc}$ is $(1-\frac{err^*}{N})^{|\mathcal{H}|} \leq e^{-\frac{err^*|\mathcal{H}|}{N}}$. Hence, the adversary γ-successfully stamps doc with $\gamma \leq e^{-\frac{err|\mathcal{H}|}{N}}$. Turning that around, if the adversary γ-successfully stamps doc, then $err \leq \frac{N}{|\mathcal{H}|}\ln(\frac{1}{\gamma})$. ☐

In particular, for every r and doc for which the stamper is γ successful, $err \leq \frac{N}{|\mathcal{H}|}\ln(\frac{1}{\gamma}) \leq g\beta D$. Hence, the stamper correctly stamps doc at r with at most $err = g\beta D$ errors. It follows that $\mathcal{R}_{\text{successful}}(k) \subseteq \mathcal{R}_{\text{correct}}(k)$ as desired.

4.5 The Proof of Lemma 4.5

We first define a compression function $\text{Com}(r)$ for $r \in \mathcal{R}_{\text{correct}}(k)$. Let $r \in \mathcal{R}_{\text{correct}}(k)$. Suppose $doc_1, \ldots, doc_k$ are the documents that the adversary correctly stamps at r with at most err errors. Denote $\Gamma = \cup_{1\leq i\leq k}\Gamma(doc_i)$, that is the set of all indices which are selected by one of the k documents, and let $\bar{\Gamma} = [N] \setminus \Gamma$ denote the remaining indices. Let $\mathcal{BAD}(doc_i)$ denote the set of indices that appear in the timestamp of doc_i and whose content is incorrect. let $\mathcal{BAD} = \cup_{1\leq i\leq k}\mathcal{BAD}(doc_i)$, that is the set of all indices which are bad for at least one of the k documents. We call an index $j \in \Gamma \setminus \mathcal{BAD}$ useful. We choose $\text{Com}(r)$ to be:

$$\text{Com}(r) = (doc_1, \ldots, doc_k; \text{Store}^*(r); r_{|\bar{\Gamma}}; \mathcal{BAD}; r_{|\mathcal{BAD}})$$

We define a "decompression" function $\text{Dec}(a)$ that gets as input $\text{Com}(r)$ and tries to recover r. Let r be a string from $\mathcal{R}_{\text{correct}}$, i.e., a string on which the stamper correctly stamps k-documents with at most err errors. From $doc_1, \ldots, doc_k$, that appear in $\text{Com}(r)$, we recover the set Γ, and from $\text{Com}(r)$ we learn which indices are in the subset $\mathcal{BAD} \subset \Gamma$. Now, for every $1 \leq j \leq N$ we recover r_j as follows:

- If $j \notin \Gamma$ then we use the information in $r_{|\bar{\Gamma}}$ to find r_j.

- If $j \in \mathcal{BAD}$ then we use the information in $r_{|\mathcal{BAD}}$ to find r_j.
- If $j \in \Gamma \setminus \mathcal{BAD}$ then we find an i such that $j \in \Gamma(doc_i)$. We run $\text{Stamp}^*(doc_i, \text{Store}^*(r))$ and take r_j from its output.

The only case where we do not take the value of r_j directly from $\text{Com}(r)$ is for $j \in \Gamma \setminus \mathcal{BAD}$. However, all such indices j are useful, and therefore we correctly decode them. Therefore, for every $r \in \mathcal{R}_{\text{correct}}$ we have $\text{Dec}(\text{Com}(r)) = r$.

We now analyze the output length of the compression function Com. The documents $doc_1, \ldots, doc_k$ take $k \log |\mathcal{DOC}|$ bits space. By definition we have that $|\text{Store}^*(r)| \le M^*$. We have that G is expanding and $k \le K_{max}$. It follows that $|\Gamma| \ge \beta kD$ and therefore $r_{|\bar{\Gamma}} \le R - \beta kDn$. We represent $\mathcal{BAD} \subseteq \Gamma$ by a binary vector of length $|\Gamma| \le kD$ which has a "one" for indices in $\Gamma \cap \mathcal{BAD}$ and a "zero" for indices in $\Gamma \setminus \mathcal{BAD}$. Each of the k documents is correctly stamped at r with at most err errors, and therefore for every such document doc_i we have $|\mathcal{BAD}_{doc_i}| \le err$ and $|\mathcal{BAD}| \le k \cdot err$. The representation of $r_{|\mathcal{BAD}}$ is therefore bounded by $k \cdot err \cdot n$. We conclude that the total length of the output of Com is at most $k \log |\mathcal{DOC}| + M^* + R - \beta kDn + kD + k \cdot err \cdot n$. We denote this quantity $R - \Delta$.

As every $r \in \mathcal{R}_{\text{correct}}$ has a small description (of length $R - \Delta$) we have $|\mathcal{R}_{\text{correct}}| \le 2^{R-\Delta}$ and therefore $Pr[r \in \mathcal{R}_{\text{correct}}] \le 2^{-\Delta}$. We have required that $n \ge 1/g\beta$ and therefore $kD \le g\beta kDn$. We also have $err = g\beta D$ and by our assumption $\log |\mathcal{DOC}| \le g\beta nD$. Altogether, $R - \Delta \le R - \beta Dkn(1-3g) + M^*$. We get that $\Delta \ge (1-3g)\beta Dkn - M^*$. As $M^* \le (1-4g)\beta kDn$, we get $\Delta \ge g\beta kDn$ as desired.

4.6 Proof of Theorem 4.2

We plug into Theorem 4.1 the $(K_{max}, \beta D)$-expanding graph of Theorem 2.2 to get a concrete result. The result then follows by choosing the remaining parameters. More precisely, let $\beta > 0$ be the constant that is guaranteed in Theorem 2.2. We are going to set the parameter n to $n = \log N$. Note that for large enough N, $n \ge 5/\beta$.

Let $\eta > 0$ be some constant. We set $|\mathcal{H}| = D = N^{1/2+\eta}$ and $\gamma = 2^{-N^\eta}$. Thus, $D|\mathcal{H}| = N^{1+2\eta} \ge \frac{5N}{\beta} \ln(\frac{1}{\gamma})$, where the inequality follows because for large enough N, $N^\eta \ge \log N \ge \frac{5}{\beta}$. We use the explicit expander of Theorem 2.2 where:

- we set $K_{max} = \frac{N}{4D\beta}$,
- and we let $|V_1| = |\mathcal{DOC}|$ be the largest integer such that the expander of Theorem 2.2 gives an expander with degree D.

We first show that we can achieve $\log |\mathcal{DOC}| \ge D^{\Omega(1)}$. By Theorem 2.2 we have that:

$$D \quad \le \quad 2^{a[\log\log \mathcal{DOC} + (\log\log K_{max})^3]} \tag{1}$$

for some constant a. Setting $\log |\mathcal{DOC}| = D^{\frac{1}{2a}}$ we can get a graph with degree bounded by $\sqrt{D} \cdot 2^{a(\log\log K_{max})^3} \le \sqrt{D} 2^{a(\log\log N)^3} \le D$ as required (where the two last inequalities follow by $K_{max} \le N$ and the fact that $D \ge \sqrt{N}$).

We now apply Theorem 4.1. Note that by our choice of parameters $\log |\mathcal{DOC}| \le D \le 5Dn\beta$ where the last inequality follows as $n \ge \frac{5}{\beta}$. We have that the scheme is $(T = V = N^{1/2+\eta} n)$-efficient and α-optimal for $\alpha = \frac{5}{\beta}$, $\rho = 2^{-\beta nD/5} \le 2^{-D}$, and $M^*_{max} = \beta DnK_{max}/5 = nN/20 = R/20$.

This is very close to what we need to prove in Theorem 4.2 except that $T, V = N^{1/2+\eta} n$ and not $(Nn)^{1/2+\eta}$ as required. Nevertheless, as $n = \log N$ we can meet the required values for T to V by slightly decreasing the constant η. Theorem 4.2 follows.

5 An Efficient Scheme Secure Against Polynomial Time Adversaries

The scheme suggested in Section 4 requires the honest parties (stamper and verifier) to store many bits, namely $TV > R$ where T is the stamp size, V the sketch size and R the random string length. In other words, if the stamp size is small (say $T = \log R$) then the sketch size V is almost all of the random string. Our second scheme has a very small sketch and stamp size. This is achieved by using the previous stamping scheme with a small T and using a different verification method that allows the verifier to use much less storage. This verification method is valid only against computationally bounded adversaries and takes advantage of the bounded computational capabilities of the cheating party.

5.1 The Stamping Scheme

Let H be a family of collision resistant hash functions and $R = \log|H| + Nn$. We partition a string $r \in \{0,1\}^R$ into $N+1$ blocks, denoted $r_0, r_1, \ldots, r_N$ where r_0 is of length $\log|H|$ and for $i > 0$, r_i is of length n. The string r_0 (which didn't appear in the previous scheme) serves as a "key" to the hash function. We use the same "index selection" mechanism as in Section 4; G is a bipartite graph with left degree D, where the left set is the set $\mathcal{DOC}$ and the right set is the set $[N]$. We now describe the stamp, sketch and verify procedures: (We use the subscript "c" to distinguish these functions from the ones given in Section 4)

Sketch$_c$ The verifier stores r_0 and the root of a Merkle hash tree over $r_1, \ldots, r_N$ with hash function r_0. That is $\text{Sketch}_c(r) = (r_0, \mathit{Root}^{r_0}(r_1, \ldots, r_N))$. Note that Sketch$_c$ is *deterministic* (unlike the case of the previous section where Sketch is probabilistic).

Stamp$_c$ Given a document $doc \in \mathcal{DOC}$ we define $\text{Stamp}_c(doc, r) = \left(\mathit{Path}_j^{r_0}(r_1, \ldots, r_N)\right)_{j \in \Gamma(doc)}$. I.e., the stamper uses the function Stamp of the previous section, and for every $j \in \Gamma(doc)$ adds a "Merkle proof" $\left(\mathit{Path}_j^{r_0}(r_1, \ldots, r_N)\right)$ that he stamped it correctly.

Verify$_c$ Given a document doc, a "root" a and a stamp y composed of D Merkle-paths $p_1, \ldots, p_D$, the function $\text{Verify}_c(a, doc, y)$ accepts iff all paths are valid.

Theorem 5.1. *There exists a polynomial $p(\cdot)$ such that the following holds: Let H be a $(\epsilon_H(n), \mu(n))$-collision resistant hash family which can be computed in space $O(Dn \log N)$. Assume the adversary runs in time at most $\frac{\mu(n)}{p(R)}$. Let G be a $(K_{max}, \beta D)$-expanding graph. Assume $\log|\mathcal{DOC}| \le \beta Dn/5$ and $n > \log N \ge \frac{5}{\beta}$. The scheme is $(T = O(Dn \log N), V = O(Dn \log N))$-efficient and $O(\alpha \cdot \log N)$-optimal against feasibly generated documents for $\alpha = \frac{5}{\beta}$, $\rho = 2^{-\beta n D/5} + \epsilon_H$ and $M^*_{max} = \beta Dn K_{max}/5$.*[5]

Plugging in the expander construction of Theorem 2.2 and collision resistant hash functions we obtain a scheme with efficiency $2^{(\log\log R)^{O(1)}}$. This is stated formally in the next theorem. (We remark that Theorem 5.2 is almost a formal restatement of Theorem 1.2. The only difference is that the scheme in Theorem 5.2 only allows to stamp short documents of length $2^{(\log\log R)^{O(1)}}$ rather than documents of length R. We will deal with this problem separately in Section 5.5).

Theorem 5.2. *Let H be a $(\epsilon_H(n), \mu(n))$-collision resistant hash family with $\mu(n) = \frac{1}{\epsilon_H(n)} = 2^{2^{(\log n)^{\Omega(1)}}}$ and $c > 1$ be some constant. Assume the adversary has running time polynomial in R. There exists a scheme that is $(T = 2^{(\log\log R)^{O(1)}}, V = 2^{(\log\log R)^{O(1)}})$-efficient and $O(\log R)$-optimal against feasibly generated documents for $\rho = \text{neg}(R)$ and $M^*_{max} = \Omega(R)$. The scheme allows stamping documents of length $2^{(\log\log R)^c}$.*

Here $\text{neg}(R)$ is a negligible function of R (a function is negligible in R if for any constant c there exists some R_c such that $\forall R > R_c \colon \text{neg}(R) < R^{-c}$). Letting ρ be a negligible function in R is a natural requirement when dealing with polynomially bounded adversaries: the probability of breaking the scheme by repeating

[5] Recall that in this scheme the verifier is *deterministic*. Thus, γ is unimportant, and we therefore omit it in this section. To be strictly formal all the propositions in this section apply for any $\gamma > 0$.

an action that succeeds with negligible probability remains negligible even after a polynomial number of repetitions.

We prove Theorem 5.2 in Subsection 5.4. It is possible to get an even more efficient scheme with $T = V = (\log R)^{O(1)}$. However, this result requires a better graph than the one constructed in Theorem 2.2. It is folklore that such graphs exist by a probabilistic argument. In Subsection 5.6 we survey the parameters that can be achieved if such graphs are explicitly constructed. In the remainder of the section we prove Theorem 5.1.

5.2 Efficiency

The efficiency parameters are derived from the fact that it is possible to compute a path of a Merkle tree online in space $O(n \log N)$, when not taking the complexity of the hash function into account. The hash function can be computed, by our assumption, in space $O(Dn \log N)$ and therefore the verifier (who stores only the root) can run online in space $O(n \log N) + O(Dn \log N) = O(Dn \log N)$. The stamper calculates the indices it will need before it sees r. It computes the Merkle-paths of these leaves, and stores the labels for the paths it will need for stamping as it goes along. There are D paths where each is of length $O(n \log N)$. By using additional $O(Dn \log N)$ space to compute the hash function, the stamper can run online in space $O(Dn \log N)$. This proves the efficiency of the scheme. We now turn to proving security.

5.3 Security

We follow the outline of the correctness proof of the information-theoretic version of Section 4, except that now we work with security for generated documents. We show that if the adversary *successfully stamps* many documents then he *correctly stamps* many documents which is impossible by the "reconstruction argument" of the previous section.

Fix some adversary with memory M^* and running time to be specified later. We use *coins* to denote the concatenation of the random coins used by $\mathrm{Store}_c^*, \mathrm{Stamp}_c^*$ and $\mathrm{Generate}_c^*$. We define $\mathcal{R}^{\mathrm{comp}}_{\mathrm{correct}}(k)$ to be the set of pairs $(r, coins)$ such that, for every $1 \le i \le k$, the Merkle paths output by $\mathrm{Stamp}_c^*(\mathrm{Store}_c^*(r), \mathrm{Generate}_c^*(r,k)_i)$ are correct.

We now want to define the computational analogue of $\mathcal{R}_{\mathrm{successful}}$ and relate it to $\mathcal{R}^{\mathrm{comp}}_{\mathrm{correct}}$. We define: $\mathcal{R}^{\mathrm{comp}}_{\mathrm{successful}}(k) = \mathcal{R}^{\mathrm{comp},\gamma=1}_{\mathrm{successful}}(k)$ to be the set of all pairs $(r, coins)$ for which the adversary 1-successfully stamps the k documents $\mathrm{Generate}_c^*(r,k)$ (i.e. all the Merkle paths output by $\mathrm{Stamp}_c^*(\mathrm{Store}_c^*(r), \mathrm{Generate}_c^*(r,k)_i)$ are valid). This definition of success corresponds to the notion of security in Definition 3.9. We prove:

Lemma 5.3. *There exists a polynomial $p(\cdot)$ such that if the adversary runs in time at most $\frac{\mu(n)}{p(R)}$. Then, for every $k \le R$*

$$\Pr_{r,coins}\left[(r, coins) \in \mathcal{R}^{\mathrm{comp}}_{\mathrm{successful}}(k) \setminus \mathcal{R}^{\mathrm{comp}}_{\mathrm{correct}}(k)\right] \le \epsilon_H$$

Proof. Assume not. Then, given a random h we generate a random r by setting $r_0 = h$ and choosing $r_1, \ldots, r_N$ at random and pick *coins* at random. We run $\mathrm{Store}_c^*(r)$ (using *coins*) and compute $\mathrm{Generate}_c^*(r,k)$ to get k documents $doc_1, \ldots, doc_k$. We now simulate the adversary's timestamp on these documents by running Stamp_c^*. In addition we compute the correct timestamps on these documents and we also run the verifier to check whether the timestamps are accepted.

We know that with probability at least ϵ_H we have that there is at least one document on which the adversary was successful but did not provide the correct timestamp. We can identify this document (as we have the correct timestamps) and check the merkle paths associated with it. Note that on this document the adversary has to provide a merkle path that is valid with respect to $r_1, \ldots, r_N$ but is incorrect.

Thus, the procedure we described finds a valid, incorrect path for $Tree^h(r_1, \ldots, r_N)$ with probability ϵ_H. Let us examine the running time of this procedure. There exist some polynomial $p(\cdot)$ such that this procedure runs in time $p(R)$ times the running time of the adversary. As we assume that the running time of the adversary is less than $\frac{\mu(n)}{p(R)}$ we have that this procedure runs in time bounded by $\mu(n)$ contradicting the security promised by Lemma 2.6. $\square$

Thus, we have that $\Pr[\mathcal{R}^{\text{comp}}_{\text{successful}}(k)] \leq \Pr[\mathcal{R}^{\text{comp}}_{\text{correct}}(k)] + \epsilon_H$. We want to use Lemma 4.5 from the previous section to bound the probability of $\mathcal{R}^{\text{comp}}_{\text{correct}}(k)$. A technicality is that this lemma deals with a more general setup where the definition of $\mathcal{R}_{\text{correct}}$ also involves an error parameter. For this technical reason we now introduce a parameter g (that is used in the Lemma) and set it to 1/5. This will allow us to show:

Lemma 5.4. *Assume* $\log|\mathcal{DOC}| \leq g\beta Dn$, *err* $= g\beta D$ *and* $n \geq \frac{1}{g\cdot\beta}$. *For any adversary with space* $M^* \leq (1-4g)\beta kDn$ *and* $k \leq K_{max}$ *we have* $\Pr_{r,coins}[(r, coins) \in \mathcal{R}^{\text{comp}}_{\text{correct}}(k)] \leq 2^{-g\beta nDk}$.

Proof. By an averaging argument there exists some fixed value $coins'$ of $coins$, such that

$$\Pr_r[(r, coins') \in \mathcal{R}^{\text{comp}}_{\text{correct}}(k)] \geq \Pr_{r,coins}[(r, coins) \in \mathcal{R}^{\text{comp}}_{\text{correct}}(k)]$$

Let $\mathcal{R}_{\text{correct}}(k) = \{r | (r, coins') \in \mathcal{R}^{\text{comp}}_{\text{correct}}(k)\}$. Note that $\mathcal{R}_{\text{correct}}(k) \subseteq \mathcal{R}^{\text{err=0}}_{\text{correct}}(k)$ where the latter is the set defined in Section 4 with zero errors. This follows because the adversary with coins fixed to $coins'$ outputs correct paths on k different documents, and every such path contains the correct leaf. Thus, if there exists an adversary A for which $Pr_{r,coins}\ [(r, coins) \in \mathcal{R}^{\text{comp}}_{\text{correct}}(k)]\ > 2^{-g\beta nDk}$, then there exists a deterministic algorithm A' (which may be non-uniform), for which $Pr_r[r \in \mathcal{R}_{\text{correct}}(k)] > 2^{-g\beta nDk}$. By Lemma 4.5, for any adversary with space $M^* \leq (1-4g)\beta kDn$ and $k \leq K_{max}$,

$$\Pr_r[r \in \mathcal{R}_{\text{correct}}(k)] \quad \leq \quad 2^{-g\beta nDk}$$

(In fact the parameters are slightly better because we have $err = 0$ and therefore $\mathcal{BAD}$ is empty). Note that Lemma 4.5 is true for *any* adversary – it makes no assumptions about its running time or uniformity – therefore our use of A' instead of A is allowed. ∎

Together, Lemmas 5.3 and 5.4 imply that $\Pr_{r,coins}[(r, coins) \in \mathcal{R}^{\text{comp}}_{\text{successful}}(k)] \leq 2^{-g\beta nDk} + \epsilon_H$. Let $T = cDn\log N$ (where c is the constant hidden in the $O(\cdot)$ notation in the assumption of Theorem 5.1). Then for $k = c\alpha \log N \frac{M^*}{T} \leq \alpha \log N \frac{M^*_{max}}{T} = K_{max}$, the probability that an adversary can generate and stamp more than k documents is less than $2^{-g\beta nDk} + \epsilon_H \leq 2^{-g\beta nD} + \epsilon_H = \rho$, which proves Theorem 5.1.

5.4 Proof of Theorem 5.2

Let $\eta > 0$ be a constant. We use the explicit expander of Theorem 2.2, with left-hand set $\mathcal{DOC}$, right-hand set of size N, left degree D, and which is $(K_{max}, \beta D)$-expanding, where we set

- $K_{max} = \frac{N}{4\beta D}$ and
- $|V_1| = |\mathcal{DOC}| = 2^{D^{1/2a}}$, where the constant $a > 0$ is the constant hidden in the $O(\cdot)$ in Theorem 2.2.

The degree of this graph is

$$2^{a[\log(D^{1/2a}) + (\log\log K_{max})^3]} = \sqrt{D}\cdot 2^{a(\log\log K_{max})^3} \leq \sqrt{D}2^{a(\log\log N)^3} \leq D$$

where the inequalities follow by $K_{max} \leq N$, and by choosing $D = 2^{(\log\log N)^v}$ where $v > 3$ is a constant to be chosen later. We choose $n = 2^{(\log\log N)^{2/a'}}$, where a' is the constant hidden in the $\Omega(\cdot)$ in the condition on μ in Theorem 5.2. We observe that $\mu(n)$ and $\frac{1}{\epsilon_H(n)}$ are greater than any polynomial in N and that $N = \frac{R}{n} \geq \sqrt{R}$. We will make sure that $e > 2/a'$ so that the space needed to compute the hash function (which is some fixed polynomial in $n = 2^{(\log\log N)^{2/a'}}$) is less than $D = 2^{(\log\log N)^v}$. Thus, the space of the hash function is bounded by $D \leq Dn\log N$ as required. We meet all the requirements of Theorem 5.1 and conclude that the scheme is (T, V)-efficient for $T = V = O(Dn\log N) \leq 2^{(\log\log N)^{3v}} \leq 2^{(\log\log R)^{2v}} = 2^{(\log\log R)^{O(1)}}$. Furthermore, $M^*_{max} = \Omega(K_{max}Dn) = \Omega(Nn) = \Omega(R)$ as required, and the scheme is $O(\log R)$-optimal for $\rho = 2^{-D} = \text{neg}(R)$. The scheme can stamp documents of length $\beta Dn/5 \geq D$. By setting $v \geq 2c$ we have that $D = 2^{(\log\log N)^v} \geq 2^{(\log\log R)^c}$.

5.5 Stamping Longer Documents

The timestamping system described in this section has good security parameters but only works if $\log|\mathcal{DOC}| = 2^{(\log\log R)^a}$ for a small constant a. This is somewhat small (Recall that honest parties run in time polynomial in R). However, using the standard technique of collision resistant hashing we can convert any timestamping system which works for short documents into a system which works for longer documents by first hashing the documents and then stamping them. When applying this to the scheme of Theorem 5.2 we get a scheme which allows stamping documents of length R. This requires the same assumption on collision resistant hash functions which was used to derive Theorem 5.2.

We now explain this transformation more formally. Let Stamp_c, Sketch_c and Verify_c be the stamping and verification functions for the original scheme, which stamps documents of length $\log|\mathcal{DOC}|$. Assume there exists $H : \{0,1\}^R \to \{0,1\}^n$ that is an $(\epsilon_H(n), \mu(n))$-collision resistant hash family. We will use this hash function with $n = \log|\mathcal{DOC}|$.

Given r, we partition it into h, r', where $|h| = \log|H|$ will be used to select a function from H. We define the new procedures $\mathrm{Sketch}_{\mathrm{long}}, \mathrm{Stamp}_{\mathrm{long}}$ and $\mathrm{Verify}_{\mathrm{long}}$.

- $\mathrm{Sketch}_{\mathrm{long}}(r) = h, \mathrm{Sketch}_c(r')$
- $\mathrm{Stamp}_{\mathrm{long}}(doc, r) = \mathrm{Stamp}_c(h(doc), r')$.
- $\mathrm{Verify}_{\mathrm{long}}(\mathrm{Sketch}_{\mathrm{long}}(r), doc, y) = \mathrm{Verify}_c(\mathrm{Sketch}(r'), h(doc), y)$.

Theorem 5.5. *Suppose the original system is a (T, V)-efficient timestamp system that is α-optimal against feasibly generated documents with parameters ρ and γ for adversaries with running time polynomial in R. Assume $\mu(\log|\mathcal{DOC}|)$ is larger than any polynomial in R and that $Space(H) \le T$. Then the derived scheme allows stamping documents of length R. The scheme is also $(T, V + \log|H|)$-efficient and α-optimal against feasibly generated documents for $\rho' = \rho + \epsilon_H(\log|\mathcal{DOC}|)$ and $\gamma' = \gamma$ when considering polynomial time adversaries.*

Proof. Suppose for a particular r and random coins the adversary can generate and γ-successfully stamp $k = \alpha\frac{M^*}{T}$ documents in the new system. That is that there is a machine $\mathrm{Generate}_{\mathrm{c}}^*$ that on input (r, k) outputs k documents $doc_1, \dots, doc_k$ that the adversary γ-successfully stamps. Consider the set of hashed documents $\{h(doc_1), \dots, h(doc_k)\}$. If this set has k distinct values then this adversary can be used to γ-successfully stamp $\alpha\frac{M^*}{T}$ documents in the original system. Therefore the probability that this event occurs is at most ρ (the adversary against the original system runs the new adversary and then hashes each document it outputs). Otherwise, by the pigeonhole principle there must be $i \neq j$ for which $h(doc_i) = h(doc_j)$, thus we can find a collision for h in time polynomial in R. Therefore the probability that this event occurs is at most $\epsilon_H(\log|\mathcal{DOC}|)$. The total probability of success is thus bounded by $\rho + \epsilon_H(\log|\mathcal{DOC}|)$. □

Combining Theorem 5.5 with the scheme of Theorem 5.2 allows us to obtain a scheme with essentially the same properties while stamping longer documents. This is stated formally in the next theorem which is a formal restatement of Theorem 1.2.

Theorem 5.6. *Let H be a $(\epsilon_H(n), \mu(n))$-collision resistant hash family with $\mu(n) = \frac{1}{\epsilon_H(n)} = 2^{2^{(\log n)^{\Omega(1)}}}$. Assume the adversary has running time polynomial in R. There exists a scheme that is $(T = 2^{(\log\log R)^{O(1)}}, V = 2^{(\log\log R)^{O(1)}})$-efficient and $O(\log R)$-optimal against feasibly generated documents for $\rho = \mathrm{neg}(R)$ and $M^*_{max} = \Omega(R)$. The scheme allows stamping documents of length R.*

Proof. Let a be the constant hidden in the $\Omega(\cdot)$ in the definition of μ. We use the scheme guaranteed in Theorem 5.2 choosing $c = 2/a$. (Note that we can indeed implement this scheme as our assumption gives us a hash function with sufficiently good parameters). This scheme allows stamping documents of length $\ell = 2^{(\log\log R)^c}$. To apply Theorem 5.5 we need to make sure that $\mu(\ell)$ is larger than any polynomial in R. Indeed,

$$\mu(\ell) = 2^{2^{(\log\log R)^{ca}}} = 2^{2^{(\log\log R)^2}}$$

is larger than any polynomial in R. We also need to check that space used to compute H on documents of length ℓ is at most T. We have that $T = 2^{(\log\log R)^{a'}}$ for some constant $a' > 0$. By increasing this constant if necessary we can make sure that T is larger than any polynomial in ℓ and so as the hash function runs in polynomial time it must use polynomial space that is smaller than T. It follows that we can apply Theorem 5.5 and obtain a scheme that allows stamping documents of length R. Let us examine the parameters of the scheme. The only losses over the original scheme is that V is longer by a polynomial in ℓ which is $2^{(\log\log R)^{O(1)}}$ (that we can afford) and that ρ increased by an additive factor of ϵ_H that is negligible in R. $\square$

5.6 A Scheme Using a Non-Explicit Expander Graph

It is folklore that there exist very good non-explicit expanders (in fact, almost every random graph is a good expander). For completeness, we give the proof for the following lemma:

Lemma 5.7. *For any constant $0 < \epsilon < 1$, any $D > \frac{1}{-\epsilon \log(1-\epsilon)}$, $N \leq 2^D$ and $M > D$ there exists a $(K_{max}, (1-\epsilon)D)$ expanding bipartite graph $G = (V_1, V_2, E)$, for $K_{max} = \frac{1}{(2e^2)^{\frac{1}{\epsilon}}} \frac{M}{D}$ left degree D, left-hand side having $|V_1| = N$ vertices and right hand side having $|V_2| = M$ vertices.*

Proof. We use a probabilistic argument to show that the probability that a random graph with the above parameters is not an expander is less than 1. Let $G = (V_1, V_2, E)$, $|V_1| = N$, $|V_2| = M$ be a random bipartite graph with left degree D constructed by choosing independently and uniformly at random D neighbors (not necessarily distinct) for each vertex $v \in V_1$. The resulting graph may not be D regular, however if it is an expander adding edges will not lower the expansion factor (hence it is enough to show that such a graph is an expander with the required properties). Note that if G is not $(K_{max}, (1-\epsilon)D)$ expanding, there must exist a set $S \subset V_1$ of size less than K_{max} and a set $T \subset V_2$ of size at less than $|S|(1-\epsilon)D$ such that $\Gamma(S) \subseteq T$. We will bound the probability that such a pair of sets exists.

For a specific pair of sets $S \subset V_1$ and $T \subset V_2$, the probability that all of S's neighbors are in T is $\left(\frac{|T|}{M}\right)^{|S|D}$. By the union bound, the probability that there exists a pair of sets S, T where S is exactly of size $i \leq K_{max}$ is less than

$$\begin{aligned} p_i &\leq \binom{N}{i}\binom{M}{i(1-\epsilon)D}\left(\frac{i(1-\epsilon)D}{M}\right)^{iD} \\ &\leq \left(\frac{Ne}{i}\right)^{i}\left(\frac{Me}{i(1-\epsilon)D}\right)^{i(1-\epsilon)D}\left(\frac{i(1-\epsilon)D}{M}\right)^{iD} \\ &= X^i \end{aligned}$$

where

$$\begin{aligned} X &= \left(\frac{Ne}{i}\right)\left(\frac{Me}{i(1-\epsilon)D}\right)^{(1-\epsilon)D}\left(\frac{i(1-\epsilon)D}{M}\right)^{D} \\ &\leq Ne^{1+(1-\epsilon)D}\left(\frac{i(1-\epsilon)D}{M}\right)^{D\epsilon} \\ &\leq 2^D e^{2D}\left(\frac{i(1-\epsilon)D}{M}\right)^{D\epsilon} \\ &= Y^D \end{aligned}$$

for

$$Y = 2e^2\left(\frac{i(1-\epsilon)D}{M}\right)^{\epsilon} \leq 2e^2\left(\frac{K_{max}D}{M}\right)^{\epsilon}(1-\epsilon)^{\epsilon} = (1-\epsilon)^{\epsilon}$$

For $D > \frac{1}{-\epsilon\log(1-\epsilon)}$, this means that $X < \frac{1}{2}$. Using the union bound again, the total probability that there exists a pair of sets (S, T) which contradicts the expansion property of the graph is at most

$$\sum_{i=1}^{K_{max}} X^i \leq \sum_{i=1}^{K_{max}} 2^{-i} < 1$$

□

We show that an explicit construction of such a graph implies much better non-interactive time-stamping schemes. Specifically,

Theorem 5.8. *Assume a graph G as above can be explicitly constructed for some $\beta = (1-\epsilon) > 0$. Let H be a $(\epsilon_H(n), \mu(n))$-collision resistant hash family with $\epsilon_H(n) = 2^{-n^{\Omega(1)}}$ and $\mu(n) = 2^{n^{\Omega(1)}}$. Assume the adversary has running time polynomial in R. For every $c > 0$, there exists a scheme that is $(T = (\log R)^{O(1)}, V = (\log R)^{O(1)})$-efficient and $O(\log R)$-optimal against feasibly generated documents for $\rho = 2^{-(\log^c R)}$, $M^*_{max} = \Omega(R)$. The scheme allows stamping documents of length $\log^c R$.*

Proof. We now use non-explicit expanders. We choose n to be large enough so that $\mu(n)$ and $1/\epsilon_H(n)$ are greater than $N^{\log^c N}$. We can meet this requirement with $n = (\log N)^{O(1)}$. This makes $N = R/n \geq \sqrt{R}$. Let d be a constant to be chosen later. We choose $D = \log^d R$. We meet all the requirements of Theorem 5.1 and conclude that the scheme is $((\log R)^{O(1)}, (\log R)^{O(1)})$-efficient for $M^*_{max} = \Omega(K_{max}Dn) = \Omega(Nn) = \Omega(R)$ as required and that it is $O(\log R)$-optimal for $\rho = 2^{-D} \leq 2^{-\log^c R}$ for sufficiently large d. The scheme can stamp documents of length $D^{\Omega(1)} \geq \log^c R$ for large enough d. □

We can now stamp much longer documents (even up to length R) using the techniques of Subsection 5.5.

6 Non-Malleable Timestamping

For several applications of timestamping, Definitions 3.4 and 3.9 do not seem to suffice. The reason is that they do not preclude the following scenario:

Alice stamps a document *doc* and stores the timestamp Stamp(doc, r). Eve stores the output of Store$^*(r)$. At verification time, Alice publishes *doc* and Stamp(doc, r). At this point Eve may be able to use the timestamp for *doc* in order to successfully stamp a different document (for instance, a document almost identical to *doc* but which contains the word "Eve" instead of the word "Alice"). Definitions 3.4 and 3.9 do not apply in this case because they only specify which documents Eve can stamp without additional information. In this section we present modified definitions and constructions that work even against adversaries that see many timestamps of documents before being requested to timestamp some other document. Throughout this section we refer to the timestamps seen by the adversary as "hints".

Unfortunately, when the adversary is unbounded we cannot expect to achieve "perfect" non-malleability, in which receiving additional hints does not allow the adversary to stamp anything but those documents. This is because there must exist a document doc_1 whose timestamp contains some non-negligible amount of information about the timestamp of a different document doc_2. If the adversary stores all but the shared information from doc_1's timestamp, the additional hint of doc_2's timestamp will allow the adversary to stamp two documents (while storing less than two timestamps). In addition, there must clearly be a limit to the number of hint documents against which the system remains secure – this is because given the timestamps of enough documents, the adversary can exceed the storage bound.

A slightly weaker version of non-malleability is still possible. Under this definition of non-malleability, we want the addition of "hint" documents to be equivalent to giving the adversary more memory ahead of time (but no hints). This is similar to the idea behind Definition 3.4.

6.1 Security against w hints

To formally define non-malleable security of a timestamp scheme, we must first formally define the adversary in this model. Our definition is similar to Definition 3.3, however in the new model the adversary is also given a "hint oracle", H. The hint oracle returns the correct timestamp for any document (except for the document the adversary is attempting to stamp). The adversary successfully stamps a document with w hints if it can successfully stamp the document in the sense of Definition 3.3 with at most w queries to the oracle (the hints may depend on the document to be stamped and on the responses to previous hints). More formally:

Definition 6.1 (Adversary)**.** *An adversary with hints consists of two functions:* $\mathrm{Store}^*(r)$*, which produces a short string* b*, and* $\mathrm{Stamp}^{*H}(doc, b)$ *which, given a document doc,* b *and access to the hint oracle* H *attempts to produce a timestamp for doc. The space* M^* *of an adversary is the maximal length of* $\mathrm{Store}^*(r)$*.*[6] *An adversary* γ-successfully stamps *a document doc at (some fixed)* r *with* w *hints if it makes at most* w *queries to* H *and*

$$\Pr[\mathrm{Verify}(\mathrm{Sketch}(r), doc, \mathrm{Stamp}^{*H}(doc, \mathrm{Store}^*(r)) = \ \mathit{Accept}] \geq \gamma$$

where this probability is over the coin tosses of Sketch *and the internal random coins of the adversary.*

We denote the set of documents that an adversary A γ*-successfully stamps with* w *hints for a given string* r *by* $\mathrm{Successful}_w^{(A,\gamma)}(r)$

We can now use essentially the same definition of security as before:

Definition 6.2 (Security against w hints)**.** *We say that a* (T, V)*-efficient timestamping scheme is* α*-optimal against* w *hints (for* $\rho > 0$*,* $\alpha \geq 1$*,* $\gamma > 0$ *and* $M^*_{max} < R$*) if for every* $T \leq M^* \leq M^*_{max}$ *and every adversary* A *with space* M^* *and* w *hints,*

$$\Pr_r \left[|\mathrm{Successful}_w^{(A,\gamma)}(r)| > \alpha \frac{M^*}{T} \right] \leq \rho$$

Remark 6.3. *The notion of* α *optimality defined in Definition 6.2 is a little different than that in Definition 3.4, in the sense that we now make the additional restriction and require that* $M^* \geq T$ *(that is, we only bound adversaries with memory that is sufficient to make at least one honest timestamp). This restriction is added as we do not know how to meet the original definition. Making this additional restriction says that, in the new definition, we do not rule out that "weak" adversaries with memory* $M^* \ll T$ *can somehow stamp* α *documents. However, as timestamps are of length* T*, it seems that interesting adversaries are those that have at least this much storage; this is why we feel this modification is not significant.*

Remark 6.4 (Usefulness of this security notion)**.** *Let us demonstrate the usefulness of this notion by considering the scenario described in the introduction. That is, we assume that there is a random variable* V *that is uniformly distributed over a large set and its value is not known to Eve at the time of the transmission of the random string. After the transmission of the random string, Eve sees a correct timestamp made by Alice who knew the value of* V *before the random string was transmitted. That is Eve sees a timestamp of the document:*

$$doc = \text{“I Alice say that the value is } v\text{”}$$

Eve would like to timestamp the document doc' *in which the word "Alice" is replaced with "Eve". In our definition of security this is captured by allowing Eve to see a correct timestamp of the hint document* doc' *before she attempts to stamp the document doc. Our definition of security says that even in such a scenario the number of documents which Eve can successfully stamp is bounded by* $\alpha\frac{M^*}{T}$*. As* V *is uniformly distributed, it follows that the probability that the value of* V *happens to appear in one of these documents is very small, and therefore Eve cannot successfully stamp the document* doc' *even after seeing Alice's timestamp.*

6.2 A non malleable timestamping scheme

In this section we revisit the information theoretic timestamping scheme defined in Section 4 and show that it is non-malleable although with weaker parameters. We remark that it is also possible to extend the definition and construction in the computational setting to be secure with hints.

We use a graph that is $(K_{max}, \beta D)$ expanding for β that is very close to 1, (say $\beta = 1 - (1/(w+1)^2))$). Loosely speaking, in this setting if the adversary receives w hints then this amounts to receiving wD indices "for free". We now claim that for any additional document that the adversary wants to stamp, most indices of this document were not revealed by the hints. This is because if one looks at the set of the hint documents and an additional document then this set expands to size $(w+1)\beta D$ and we note that

$$(w+1)\beta D - wD = (w+1) \cdot \left(1 - \frac{1}{(w+1)^2}\right) D - wD = \left(1 - \frac{1}{w+1}\right) D$$

[6] Note that the adversary is not required to run *online* in space M^*. The function $\mathrm{Store}^*(r)$ can be an arbitrary function of r.

So, intuitively, the hints do not help the adversary by much as he needs to store the additional indices. A more formal argument follows. We prove the following Theorem (which is analogous to Theorem 4.1).

Theorem 6.5. *Let G be a $(K_{max}, \beta D)$-expanding graph with left degree D, let $\gamma > 0$ and let w be such that $w \le \frac{1}{8(1-\beta)}$ and $w \le K_{max}/2$. Let $g = 1/100(w+1)$. Assume that $\log|\mathcal{DOC}| \le \frac{Dn}{200(w+1)}$ and $n \ge max(100(w+1), \log N)$ and $D|\mathcal{H}| \ge \frac{N}{g\beta}\ln(\frac{1}{\gamma})$. Then the scheme is $(T = Dn, V = 2|\mathcal{H}|n)$-efficient and α-optimal against w hints. For $\alpha = 16w^2$, $\rho = 2^{-nD/6(w+1)}$ and $M^*_{max} = K_{max}Dn/32w^2$.*

The parameters in Theorem 6.5 resemble those of Theorem 4.1. The main differences are:

- The new theorem requires a graph with better expansion properties. The expansion factor β is now $1 - 1/8w$ and in particular needs to be close to one, whereas Theorem 4.1 allows β to be small. In particular, when using the explicit graph of Theorem 2.2, we have β that is a small positive constant, which is not good enough for our purposes.
- The optimality factor α is now $O(w^2)$ whereas in Theorem 4.1 the optimality factor can be made $O(1)$ when β is a constant.
- The parameter g (which measures the fractions of errors above which the verifier rejects) is now set to $1/100(w+1)$ whereas in Theorem 4.1 it was set to $1/5$. This creates losses that depend on w in several of the parameters whereas previously these losses were constants.
- The maximal memory of the adversary M^*_{max} is smaller by a multiplicative factor of $1/w^2$ compared to Theorem 4.1. This means that even in an optimal graph where $K_{max}D \approx N$ the adversary will only be allowed to store R/w^2 bits from the random string. This should be compared to Theorem 4.1 which allows M^*_{max} to approach R when picking $g = o(1)$.

To simplify the parameters let us focus on $w = O(1)$; that is, achieving security against a constant number of hints. In this case, if we have a graph that meets the more stringent requirements of Theorem 6.5, then the timestamping scheme we get has comparable parameters to that obtained in Theorem 4.1.

However, as noted earlier, Theorem 2.2 does not give a graph with an expansion factor β that is close to one. Using non-explicit graphs (such as the one in section 5.6) it is possible to get a scheme with parameters comparable to those in Theorem 4.2 that is secure against a constant number of hints.

Remark 6.6 (Toward explicit constructions of expander graphs)**.** *It seems plausible that combining the technique used in our proof of Theorem 2.2 with the idea of "adding a buffer" (see [26, 31]) would allow us to construct graphs in which the expansion factor β is $1 - o(1)$ and the degree is comparable to the degree in our Theorem 2.2.*

6.3 The security proof

We prove Theorem 6.5 in a similar way to the proof of Theorem 4.1. Fix some adversary A with w hints, we denote the set of strings r on which the adversary γ-successfully stamp k documents by $\mathcal{R}_{\text{successful}}{}^{\gamma}(k, w)$. We now define the notion of "correct stamping with w hints" in a similar way to Definition 4.3.

Definition 6.7. *Let $g > 0$ be a parameter. An adversary* correctly stamps *a document doc at r with w hints if* $\text{Stamp}^{*H}(doc, \text{Store}^*(r)) = r_{|\Gamma(doc)}$ *and it makes at most w queries to H. An adversary* correctly stamps *a document doc at r with w hints and at most err errors, if the Hamming distance between* $\text{Stamp}^{*H}(doc, \text{Store}^*(r))$ *and $r_{|\Gamma(doc)}$ is at most err and it makes at most w queries to H.*

Let $\mathcal{R}_{\text{correct}}(k, w) = \mathcal{R}^{\text{err}=\text{g}\beta\text{D}}_{\text{correct}}(k, w)$ denote the set of random strings r for which there are at least k documents that the adversary correctly stamps with at most $err = g\beta D$ errors and w hints.

Note that the timestamping system is the same as that in Section 4. Thus, from the point of view of the verifier, nothing has changed and Lemma 4.4 applies. In our terminology we have that:

Lemma 6.8. *Assume $D|\mathcal{H}| \ge \frac{N}{g\beta}\ln(\frac{1}{\gamma})$. For every w, k,*

$$\mathcal{R}_{\text{successful}}{}^{\gamma}(k, w) \subseteq \mathcal{R}_{\text{correct}}(k, w).$$

We now show that any adversary that correctly stamps too many documents with w hints can be used to compress the string r, which in turn gives a contradiction.

Lemma 6.9. *Assume* $\log|\mathcal{DOC}| \leq Dn/200(w+1)$, $err = g\beta D$, $g \leq 1/100(w+1)$, $n \geq 100(w+1)$ *and* $\beta \geq (1 - 1/4(w+1))$. *For every* k *such that* $4(w+1)^2 \leq k \leq K_{max} - w$ *and any adversary with space* $M^* \leq kDn/4(w+1)$ *we have* $\Pr_r[r \in \mathcal{R}_{\text{correct}}(k,w)] \leq 2^{-nDk/6(w+1)}$.

The overall structure of the proof is similar to the one used in the proof in Lemma 4.5. However, the situation here is somewhat more complicated, as the compression procedure is not allowed to use hints. Instead, we show how to find a set of k documents (which may be different than the one which the adversary is correct on) such that we can use the adversary and some additional information (used to simulate hints) to correctly reconstruct the content of indices adjacent to these documents even without hints. The interplay between the various parameters is also somewhat different than the situation in Lemma 4.5 as we need much more additional memory in order to compensate for not having hints. The proof of Lemma 6.9 appears in Section 6.4. We now conclude the proof of Theorem 6.5.

Proof. (of Theorem 6.5) The efficiency requirement follows immediately as in the proof of Theorem 4.1. We need to show that no adversary with space $T \leq M^* \leq M^*_{max}$ can γ-successfully stamp more than $k = \frac{\alpha M^*}{Dn}$ documents. Given an adversary that γ-successfully stamps k documents, we meet the requirements of Lemma 6.8 and therefore this adversary correctly stamps k documents with at most $err = g\beta D$ errors. We now apply Lemma 6.9. We first check that the parameters we chose meet the requirements. This is immediate for the first five requirements. For the sixth requirement we note that we have required that $M^* \geq T = Dn$ and therefore $k = \frac{\alpha M^*}{Dn} \geq \alpha = 16w^2 \geq 4(w+1)^2$. We also need to meet the requirement that $k \leq K_{max} - w$. We have that

$$k = \frac{\alpha M^*}{Dn} \leq \frac{\alpha M^*_{max}}{Dn} \leq \frac{K_{max}}{2} \leq K_{max} - w$$

where the second inequality follows from the definition of M^*_{max} and α, and the third inequality follows from the requirement that $w \leq K_{max}/2$. We also meet the requirement that $M^* \leq kDn/4(w+1)$ which holds as $k = \alpha M^*/Dn = 16w^2 M^*/Dn$. We thus conclude that the probability that the adversary successfully stamps k documents is at most $\rho = 2^{-nDk/6(w+1)} \leq 2^{-nD/6(w+1)}$ as $k \geq 1$. ☐

Remark 6.10. *We remark that we could have set the parameters differently and improved the optimality factor from* $O(w^2)$ *to* $O(w)$, *at the cost of requiring that* $M^* \geq Tw$. *This would have enabled us to meet the requirement that* $k \geq 4(w+1)^2$ *even when setting* $\alpha = \Theta(w)$, *as in that case* $k = \frac{\alpha M^*}{Dn} \geq \alpha w = \Theta(w^2) \geq 4(w+1)^2$.

6.4 Proof of Lemma 6.9

We first define a compression function $\text{Com}(r)$ for $r \in \mathcal{R}_{\text{correct}}(k,w)$. Let $r \in \mathcal{R}_{\text{correct}}(k,w)$ and let S_r denote the set of documents that the adversary correctly stamps with w hints and at most $err = g\beta D$ errors. We conduct the following process:

- We start with $B_r \leftarrow \emptyset$,
- At each step we take a document *doc* out of S_r. We examine all the documents queried by Stamp^{*H} when given $\text{Store}^*(r)$ and attempting to stamp *doc*.
- We add *doc* to B_r and call it a "base document" and we also add to B_r all the hint documents queried by Stamp^{*H} that are not already in B_r and call them "hint documents". We delete from S_r all the hint documents that appear in it, and we also remove *doc* from S_r.
- We continue this process until B_r has at least k documents. (Note that at this point B_r contains at most $k+w$ documents).

We define $\Gamma = \cup_{doc \in B_r} \Gamma(doc)$. That is the set of all indices which are neighbors of documents in B_r. For a base document $doc \in B_r$ we define $\mathcal{BAD}(doc)$ to be the indices of blocks in which Stamp^{*H} makes an error when attempting to stamp *doc* using correct hints. Let $\mathcal{BAD}$ be the union of $\mathcal{BAD}(doc)$

over all base documents. Let q be the number of base documents in B_r and enumerate the elements of B_r by $doc_1, \ldots, doc_{|B_r|}$ where the first q elements are base documents. Let v denote the concatenation of $\text{Stamp}(r, doc)$ over all hint documents. (That is the correct timestamping of all hint documents). We define:

$$\text{Com}(r) = (doc_1, \ldots, doc_{|B_r|}; \text{Store}^*(r), r|_{\bar{\Gamma}}, \mathcal{BAD}, r|_{\mathcal{BAD}}, v, q)$$

We now show that for every $r \in \mathcal{R}_{\text{correct}}(k, w)$ it is possible to reconstruct r given $\text{Com}(r)$. When given an index j we reconstruct r_j as follows:

- If $j \notin \Gamma$ then we have r_j stored.
- If $j \in \Gamma$, then it is a neighbor of some document $doc \in B_r$. Using the stored information we can check whether doc is a base document or a hint document.
  - If doc is a hint document then we have its timestamp and we can reconstruct r_j.
  - If doc is a base document then we can compute its timestamp as follows: Simulate Stamp^{*H} when attempting to stamp doc (note that we have $\text{Store}^*(r)$). Whenever an oracle query is made, the queried document is in B_r. If it is a hint document then we can answer the query. If it is a base document we recursively start to compute its timestamp. When we complete the simulation of Stamp^{*H} we "error-correct" this value by checking which indices of the timestamp are in $\mathcal{BAD}$ and replacing their content with the correct content that we have stored.

We need to show that:

Claim 6.11. *The simulation described above stops, and it stops with the correct timestamp of the given base document.*

Proof. (of claim) We define the following relations on base documents: We say that doc_1 appeared after doc_2 denoted by $doc_1 \succ doc_2$ if in the process of constructing B_r, doc_1 was taken from S_r after doc_2. We say that doc_1 calls doc_2 denoted by $doc_1 \Rightarrow doc_2$ if doc_2 is one of the hints that Stamp^{*H} queries when attempting to stamp doc_1. Let $\Rightarrow_*$ be the transitive closure of $\Rightarrow$. That is, we say that doc_1 eventually calls doc_2 denoted by $doc_1 \Rightarrow_* doc_2$ if there is a sequence of calls that starts from doc_1 and ends at doc_2.

We claim that if doc_1 eventually calls doc_2 then doc_1 appeared after doc_2. Before verifying this claim we note that $\succ$ is an ordering of the base documents, and therefore the recursion does not run into an infinite loop and will eventually end. It can be verified by induction that each time the recursive procedure "returns" a timestamp, then this timestamp is correct.

We are left with proving that if $doc_1 \Rightarrow_* doc_2$ then $doc_1 \succ doc_2$. We prove this by induction on the number of calls. For the base case note that if $doc_1 \prec doc_2$ then it could not be the case that $doc_1 \Rightarrow doc_2$ as when removing doc_1 from S_r we have removed from S_r all the documents such that $doc_1 \Rightarrow doc_2$ and thus could not select doc_2 at a later stage. The same reasoning applies to any number of calls by induction on the number of calls. □

We now bound the output length of $\text{Com}(r)$.

- Storing $|B_r|$ documents takes at most $(k + w) \log |\mathcal{DOC}|$ bits.
- The length of $\text{Store}^*(r)$ is at most M^*.
- The length of $r|_{\bar{\Gamma}}$ is at most $R - k\beta Dn$. This is because $|\Gamma| \geq k\beta D$.
- We represent the set $\mathcal{BAD}$ by a binary vector of length kD, composed of k blocks (one for each base document in B_r) and the i'th bit in a block of a document is set if the document is a base document and the i'th block in its timestamp is incorrect.
- The length of $r|_{\mathcal{BAD}}$ (in bits) is at most $kg\beta Dn$. This is because the size of $\mathcal{BAD}$ is at most $k \cdot err \leq kg\beta D$.

- The length of v is at most $(1 - 1/(w+1)) \cdot (k+w) \cdot Dn$. This is because at least a $1/(w+1)$ fraction of the documents in $\{0,1\}_r$ are base documents (as in each step in the process one out of at most $w+1$ documents added to B_r is a base document). Thus, out of at most $k+w$ documents in B_r a $1-1/(w+1)$ fraction are hint documents and the time stamp of each one takes Dn bits.

- The length of q is $\log k$.

Overall the total length of the output of $\text{Com}(r)$ is bounded by

$$\begin{aligned}
|\text{Com}(r)| &= |(doc_1, \ldots, doc_{|B_r|}, \text{Store}^*(r), r|_{\bar{\Gamma}}, \mathcal{BAD}, r|_{\mathcal{BAD}}, v, q)| \\
&= \underbrace{|doc_1, \ldots, doc_{|B_r|}|}_{\le (k+w)\log|\mathcal{DOC}|} + \underbrace{|\text{Store}^*(r)|}_{\le M^*} + \underbrace{|r|_{\bar{\Gamma}}|}_{\le R-k\beta Dn} + \underbrace{|\mathcal{BAD}|}_{\le kD} + \underbrace{|v|}_{\le (1-\frac{1}{w+1})\cdot(k+w)\cdot Dn} + \underbrace{|q|}_{\le \log k}
\end{aligned}$$

We note that $|\text{Com}(r)| \le R - \Delta$ for

$$\Delta = k\beta Dn - ((k+w)\log|\mathcal{DOC}| + M^* + kD + kg\beta Dn + (1-1/(w+1)) \cdot (k+w) \cdot Dn + \log k) \tag{2}$$

Thus, we have that $|\mathcal{R}_{\text{correct}}(k,w)| \le 2^{R-\Delta}$, and therefore the probability that a randomly chosen r lands in $\mathcal{R}_{\text{correct}}(k,w)$ is at most $2^{-\Delta}$. We are left with checking that the requirements in the Lemma give that $\Delta > \frac{kDn}{100(w+1)}$. We define $A = \frac{kDn}{(w+1)}$

Let us examine the expression in (2). Intuitively, we set the parameters so that all the terms apart from the first and sixth terms are small compared to A, and the fact that β is very close to 1 will give that the first term minus the sixth term is $\Omega(A)$. (This is the best we can get because no matter how we set the parameters the first term minus the fourth term is at most A).

We first go over each of the other terms and verify that the requirements we have made make them small compared to A. We have required that $\log|\mathcal{DOC}| \le Dn/200(w+1)$ and we have that $w \le k$ which gives that $(k+w)\log|\mathcal{DOC}| \le 2k\log|\mathcal{DOC}| \le A/100$. We have required that $M^* \le A/4$. We have required that $n \ge 100(w+1)$ which gives that $kD \le A/100$. Note that $\log k \le kD$, and therefore $\log k \le A/100$. We have required that $g \le 1/100(w+1)$ which gives that $kg\beta Dn \le A/100$.

Overall we have that $\Delta \ge k\beta Dn - (1-1/(w+1)) \cdot (k+w) \cdot Dn - A/3$. We now use the requirement that $\beta \ge 1 - 1/4(w+1)$ and compute:

$$k\beta Dn - (1-1/(w+1)) \cdot (k+w) \cdot Dn \ge Dn\left(k(1 - \frac{1}{4(w+1)}) - (k+w)(1-\frac{1}{w+1})\right)$$

$$\ge Dn\left(k(1-\frac{1}{4(w+1)}) - k(1-\frac{1}{w+1}) - w\right) \ge Dn\left(\frac{3k}{4(w+1)} - w\right) \ge Dn\left(\frac{k}{2(w+1)}\right) \ge \frac{A}{2}$$

where the last inequality follows from the requirement $k \ge 4 \cdot (w+1)^2$ which entails $w \le k/4(w+1)$. Thus, we conclude that $\Delta \ge A/2 - A/3 = A/6$ as required.

7 An Explicit Expander Construction

7.1 Extractors

In this section we prove Theorem 2.2. The proof uses techniques from the area of "randomness extractors". We only briefly describe the needed background from this area and the interested reader is referred to the survey papers [25, 29].

We begin with some definitions. For a distribution X we denote $X(a) = \Pr[X = a]$. We measure the distance between two distributions X and Y over the same domain Λ, by their ℓ_1 distance, $d(X,Y) = \sum_{a\in\Lambda}|X(a) - Y(a)|$. We say X is ϵ-close to Y if their distance is at most ϵ. The *min-entropy* of a distribution X is the maximal k such that for every a, $X(a) = \Pr[X = a] \le 2^{-k}$. Randomness extractors are given an input drawn from some unknown distribution X that is guaranteed to have at least k min-entropy. They also use a *short* independent random string. The goal is to output a short string (of length k or smaller) such that the output distribution is close to uniform. Formally.

Definition 7.1 (Strong Extractor)**.** *We say $F : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a strong (k, ϵ)-extractor if for every distribution X over $\{0,1\}^n$ that has min-entropy at least k, the distribution $F(X, U_d) \circ U_d$ is ϵ-close to uniform (where $F(X, U_d) \circ U_d$ is the distribution obtained by picking x according to the distribution X, picking y uniformly at random from $\{0,1\}^d$ and outputting $F(x, y) \circ y$).*

7.2 Universal Extractors

It is easy to see that a (k, ϵ) strong extractor gives rise to a bipartite graph in which every set S of size $K = 2^k$ on the left hand side expands (where the expansion factor depends on the parameters of the extractor). (We elaborate on this relationship in Section 7.3). The notion of expander graphs used in this paper makes the stronger requirement that there is some threshold K_{max} such that every set of size *at most* K_{max} expands. While extractors do not suffice for this, *universal extractors*, that we soon define, give rise to such graphs.

The extractor property guarantees that every distribution X that has a lot of entropy, is converted to (close to) the uniform distribution. We now extend this notion to distributions with less entropy as follows. Given an input from an unknown distribution X with some min-entropy $k \leq k_{max}$, we require that the randomness-extractor output has a restriction to $m(k)$ bits (where $m(k)$ is some function of k) that is close to uniform. We say $F' : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{m'}$ is a restriction of $F : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, if there exists some set $S \subseteq \{1, .., m\}$ of size m' such that $F'(x, y) = F(x, y)|_S$, that is F' outputs only the indices of F that are in S. We define:

Definition 7.2 (universal extractor)**.** *[27, 28, 31]*
$F : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{m(k_{\max})}$ is a strong $(k_{\max}, k \to m(k), \epsilon)$ universal *extractor if for every $k' \leq k_{\max}$ there is a restriction $F' : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{m(k')}$ of F which is a strong (k', ϵ)–extractor. If $m(k) = k - \Delta$, we say F has Δ entropy loss.*

Raz, Reingold and Vadhan [28] show variants of Trevisan's extractor [33] with optimal entropy loss. Raz et. al. note that some of the constructions they give are universal.[7] In particular, the following construction is universal:

Fact 7.3. *For every $\epsilon > 0$ and $k_{max} \leq n$, $E_{RRV} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{m(k_{max})}$ is a strong $(k_{\max}, k \to m(k), \epsilon)$* universal *extractor, with $m(k) = k - \Delta$, $\Delta = 2\log(\frac{1}{\epsilon}) + O(1)$ and $d = O(\log^2(\frac{n}{\epsilon}) \log k)$.*

We would like to reduce the seed length of the above strong extractor from $d = O(\log^3(n/\epsilon))$ to $d = O(\log n + \log^3(k_{max}/\epsilon))$. Given an unknown distribution X with $k \leq k_{max}$ entropy, [17] showed how to condense it while essentially retaining all its min-entropy. Formally, they show a function $F : \{0,1\}^n \times \{0,1\}^{d_1 = 4(\log(n/\epsilon))} \to \{0,1\}^{m = 2(d_1 + k_{max})}$ such that for every distribution X over $\{0,1\}^n$ with $k \leq k_{max}$ min-entropy, $F(X, U_d)$ is ϵ-close to a distribution with $k + d_1$ min-entropy (see Theorem 4.2 there when setting $\alpha = 1$). Furthermore $F(x, y) = (y, F'(x, y))$ for some function F', i.e., the output contains the seed. Thus, the condenser uses $d_1 = O(\log(n/\epsilon))$ truly random bits to condense the distribution X (over n bits) to a new distribution X' over just $m = O(d_1 + k_{max})$ bits while essentially retaining all the min-entropy of the distribution.

If we first apply apply such a lossless condenser $F(x, y) = (y, F'(x, y))$ and then a strong universal extractor on the second register, we get a strong universal extractor. To see that, notice that for every distribution X with min-entropy $k \leq k_{max}$, $F(X, U_d)$ is ϵ–close to a distribution B with $k + d_1$ min-entropy. In B at most ϵ of y get weight larger than $2 \cdot 2^{-d_1}$. For all other y's, $(B|Y = y)$ has min-entropy at least $k - 1$. Then, by the universal extractor property, there is a restriction of the output that is close to uniform. In particular, by first applying the [17] condenser and then the extractor of Fact 7.3, we get:

Lemma 7.4. *(based on [28, 17]) For every $\epsilon > 0$ and $k_{\max} \leq n$ there exists an explicit strong $(k_{\max}, k \to m(k), \epsilon)$ universal extractor $F : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{m(k_{\max})}$ with $d = O(\log n + \log^3(k_{\max}/\epsilon))$, $m(k) = k - \Delta$ and $\Delta = 2\log(\frac{1}{\epsilon}) + O(1)$.*

[7] Raz et. al. remark on this property without giving it a name. See Remark 12 in [28]. The name *universal* first appears in [31].

7.3 Graphs and functions

Randomness extractors are, in fact, equivalent to unbalanced expander graphs, and here we repeat this well known connection. We translate a function $F : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^m$ to a bipartite graph $G = (V_1, V_2, E)$ with $|V_1| = N = 2^n$, $|V_2| = D \cdot 2^m$ and left degree $D = 2^d$, where we put an edge $(v_1, (v_2, y)) \in E$ iff $F(v_1, y) = v_2$. A simple observation is:

Lemma 7.5. *Assume F is a strong $(k_{\max}, k \rightarrow m(k), \epsilon)$ universal extractor with Δ entropy loss. Then G is $(K_{max} = 2^{k_{max}}, \beta)$ expanding, with $\beta = \frac{1}{2}\frac{1}{2^\Delta}(1-\epsilon)$.*

Proof. Fix any subset $A \subseteq V_1$ of cardinality $K \leq K_{max}$. Let $K = 2^k$. Notice that we can associate with the subset $A \subseteq V_1$ a distribution which is uniform over the elements of A, and this distribution has $\log(|A|) = k$ min-entropy. In particular, $(F(A, U_D), U_D)$ is ϵ-close to uniform. This means that A has at least $(1-\epsilon)|V_2| = (1-\epsilon)D2^{m(k)}$ different neighbors in V_2 (the factor D appears because F is strong, and so the output contains the seed). In particular, the number of neighbors is $\frac{1}{2}(1-\epsilon)D\frac{K}{2^\Delta}$ as desired (the extra 1/2 factor is because the cardinality of A may not be a power of two). □

Having that, Theorem 2.2 follows from Lemma 7.4 by plugging in $\epsilon = \frac{1}{2}$.

8 Discussion and Open Problems

8.1 Advantages of Our Non-interactive Timestamping Scheme

Let us discuss some obvious advantages of non-interactive timestamping over more standard notions of timestamping.

- The only communication made before the verification process is the transmission of the random string r. This allows the timestamping system to be used in situations where communication is infeasible or undesirable. E.g., communication may be asymmetric: one central agency can broadcast to all other users, while the users can not send messages to the agency.
- Everyone can stamp and everyone can verify and no central control or acquaintance between stamper and verifier is needed. The decentralized nature of this scheme overcomes many of the "trust" problems with interactive timestamping systems. Even in distributed interactive systems, some measure of trust must be given to third parties. Our non-interactive timestamping system requires only that the random string be truly random and receivable by all parties.
- Privacy. The scheme hides the fact that timestamping occurred at all, e.g., an inventor can safeguard her inventions without revealing even the fact of their existence. This also ensures privacy in an information-theoretic sense.
- Our schemes solve some of the robustness problems that plague interactive timestamping systems. In particular, it is much more difficult to mount a denial-of-service attack as there is no central point that can shut down the system, and even temporarily shutting down communications will not prevent the creation of new timestamps. The lack of communication also makes it difficult for an attacker to tell whether such an attack was successful.

8.2 Open Problems

Dealing with Errors: Most protocols in the bounded storage model, and ours among them, assume the broadcast random string is received identically and without errors by all parties. However, in many natural implementations of such protocols, this assumption may not be realistic (e.g. when the random string has a natural source).

Our information-theoretic scheme can be made to work even with errors (provided the error rate is low enough) by allowing the verifier to accept a timestamp even if the blocks in the intersection differ by a small amount. The proof of Lemma 4.5 already allows the adversary to make some errors when stamping, and still be considered successful. Increasing the error rate by a small amount will not invalidate the lemma (although the parameters suffer slightly).

The computational scheme, on the other hand, currently requires the random string to be received perfectly by all parties. It is an interesting open question whether this requirement can be removed.

Removing the Need for Constant Monitoring: Our timestamping schemes require the verifier to run the Sketch function in every round for which it may, someday, want to verify documents. The verifier must therefore constantly monitor the random string, which is too much to ask from a casual user of the system.

An implementation of our timestamping systems can overcome this difficulty by using "verification centers": dedicated third parties who act as verifiers. In some sense, such third parties appear in all previous timestamping protocols. This raises the issue of how much trust the user must place in the verification center.

In the computational version of our protocol, the verification center is also easily auditable by casual users: the verifier is deterministic and has no secret information. Any user can act as a verifier for a single round, and compare its state to that of the verification center: any inconsistency will be instantly visible.

Online Versus Locally-Computable: The strategies for the honest players are efficient in the sense that they work online using small space and polynomial time. A stronger notion of efficiency called "locally-computable" was suggested in [34]. It requires the honest players to store a small substring of the string r. More precisely, the players need to choose a subset $S \subseteq [R]$ before the random string is transmitted and only store $r|_S$. We point out that the "information-theoretic" scheme (Section 4) has this additional property, whereas the "computationally-secure" scheme (Section 5) does not.[8] Natural open problems are whether the "information-theoretic" scheme can be improved to yield better parameters, and whether the "computationally-secure" scheme can be improved to run with strategies that are locally computable.

Acknowledgements

We thank Danny Harnik, Amir Herzberg, Moni Naor and Muli Safra for helpful discussions, and the anonymous reviewers for their constructive comments.

References

[1] Y. Aumann, Y.Z. Ding, and M. O. Rabin. Everlasting security in the bounded storage model. *IEEE Transactions on Information Theory*, 48:1668–1680, 2002.

[2] Y. Aumann and M. O. Rabin. Information theoretically secure communication in the limited storage space model. In *Advances in Cryptology, Proceedings of the 19th Annual International Cryptology Conference (CRYPTO)*, volume 1666, pages 65–79, 1999.

[3] D. Bayer, S. Haber, and W. S. Stornetta. Improving the efficiency and reliability of digital time-stamping. In R. M. Capocelli et al., editor, *Sequences II: Methods in Communication, Security and Computer Science*, pages 329–334. Springer-Verlag, 1992.

[4] J. Benaloh and M. de Mare. Efficient broadcast time-stamping. Technical Report 1, Clarkson University Department of Mathematics and Computer Science, August 1991.

[5] J. C. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In *Advances in Cryptology, Proceedings of the 12th Annual International Conference on the Theory and Application of Cryptographic Techniques*, pages 274–285, 1993.

[8] In the "computationally-secure" scheme , both stamper and verifier read blocks of the string r online, and need to "hash them" quickly before reading the next incoming blocks. Thus, to implement this scheme, one needs to use hash functions which can be computed very efficiently.

[6] A. Buldas and P. Laud. New linking schemes for digital time-stamping. In *Information Security and Cryptology*, pages 3–13, 1998.

[7] A. Buldas, P. Laud, H. Lipmaa, and J. Villemson. Time-stamping with Binary Linking Schemes. In *Advances in Cryptology, Proceedings of the 18th Annual International Cryptology Conference (CRYPTO)*, pages 486–501, 1998.

[8] A. Buldas, H. Lipmaa, and B. Schoenmakers. Optimally efficient accountable time-stamping. In *Public Key Cryptography*, pages 293–305, 2000.

[9] C. Cachin, C. Crepeau, and J. Marcil. Oblivious transfer with a memory-bounded receiver. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 493–502, 1998.

[10] C. Cachin and U. Maurer. Unconditional security against memory-bounded adversaries. In *Advances in Cryptology, Proceedings of the 17th Annual International Cryptology Conference (CRYPTO)*, pages 292–306, 1997.

[11] I. Damgård. Collision Free Hash Functions and Public Key Signature Schemes. EUROCRYPT, pages 203–216, 2007.

[12] Y. Z. Ding. Oblivious transfer in the bounded storage model. In *Advances in Cryptology, Proceedings of the 21st Annual International Cryptology Conference (CRYPTO)*, pages 155–170, 2001.

[13] Y. Zong Ding, D. Harnik, A. Rosen, and R. Shaltiel. Constant-round oblivious transfer in the bounded storage model. In *Theory of Cryptography, First Theory of Cryptography Conference (TCC)*, pages 446–472, 2004.

[14] Y.Z. Ding and M.O. Rabin. Hyper-encryption and everlasting security. In *Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 1–26, 2002.

[15] S. Dziembowski and U. Maurer. Tight security proofs for the bounded-storage model. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 341–350, 2002.

[16] S. Goldwasser, S. Micali, and R. Rivest. A "Paradoxical" Solution to the Signature Problem. In *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science*, pages 441–448, 1984.

[17] V. Guruswami, C. Umans, and S. P. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. In *IEEE Conference on Computational Complexity*, pages 96–108, 2007.

[18] S. Haber and W. S. Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, 1991.

[19] S. Haber and W. S. Stornetta. Secure names for bit-strings. In *ACM Conference on Computer and Communications Security*, pages 28–35, 1997.

[20] J. Kilian. A Note on Efficient Zero-Knowledge Proofs and Arguments. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 723–732, 1992.

[21] C. Lu. Hyper-encryption against space-bounded adversaries from on-line strong extractors. In *Advances in Cryptology, Proceedings of the 22nd Annual International Cryptology Conference (CRYPTO)*, pages 257–271, 2002.

[22] U. Maurer. Conditionally-perfect secrecy and a provably-secure randomized cipher. *Journal of Cryptology*, 5(1):53–66, 1992.

[23] R. C. Merkle. A certified digital signature. In *Advances in Cryptology, Proceedings of the 18th Annual International Cryptology Conference (CRYPTO)*, pages 218–238, 1989.

[24] T. Moran, R. Shaltiel, and A. Ta-Shma. Non-interactive timestamping in the bounded storage model. In *Advances in Cryptology, Proceedings of the 24th Annual International Cryptology Conference (CRYPTO)*, volume 1666, pages 460–476, 2004.

[25] Nisan and Ta-Shma. Extracting randomness: A survey and new constructions. *Journal of Computer and System Sciences*, 58:148–173, 1999.

[26] R. Raz and O. Reingold. On recycling the randomness of states in space bounded computation. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 159–168, 1999.

[27] R. Raz, O. Reingold, and S. Vadhan. Extracting all the randomness and reducing the error in Trevisan's extractors. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 149–158, 1999.

[28] R. Raz, O. Reingold, and S. Vadhan. Extracting all the randomness and reducing the error in Trevisan's extractors. *Journal of Computer and System Sciences*, 65(1):97–128, 2002.

[29] Ronen Shaltiel. Recent developments in explicit constructions of extractors. *Bulletin of the EATCS*, 77:67–95, 2002.

[30] A. Srinivasan and D. Zuckerman. Computing with very weak random sources. *SIAM Journal on Computing*, 28:1433–1459, 1999.

[31] A. Ta-Shma. Storing information with extractors. *Information Processing Letters*, 83(5):267–274, 2002.

[32] A. Ta-Shma, C. Umans, and D. Zuckerman. Loss-less condensers, unbalanced expanders, and extractors. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 143–152, 2001.

[33] L. Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4):860–879, 2001.

[34] S. P. Vadhan. Constructing locally computable extractors and cryptosystems in the bounded storage model. *Journal of Cryptology*, 17(1):43–77, 2004.