Non-Interactive Timestamping in the Bounded Storage Model

Tal Moran[1], Ronen Shaltiel[2⋆], and Amnon Ta-Shma[1]

[1] Department of Computer Science, Tel-Aviv University, Tel-Aviv, Israel
[2] Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel

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. Unfortunately, no such scheme exists against polynomial time adversaries that have *unbounded storage* at their disposal.
In this paper we show non-interactive timestamping is possible in the bounded storage model. In this model it is assumed that all parties participating in the protocol have small storage, and that in the beginning of the protocol a very long random string (which is too long to be stored by the players) is transmitted. 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 assuming 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.

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.

⋆ Research supported by the Koshland Scholarship.

Previous Work

Digital timestamping systems were first introduced in Haber and Stornetta [16], 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 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 [17, 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.

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 active 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 generate 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.

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.

The Bounded Storage Model

In contrast to the usual approach in modern Cryptography, Maurer's *bounded storage model* [19] 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 [19, 10, 2, 1, 14, 15, 18, 25], Key Agreement [10] and Oblivious Transfer [9, 12, 13]. In contrast, 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".

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.[3]

The Stamping Mechanism: To stamp a document *doc* at time t, the scheme specifies a function Stamp(doc, r) whose output is short. To stamp the document *doc*, the stamper stores 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 Sketch(r)) for every time t. At a later time the stamper can send the timestamp 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.

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 timestamp (which is of 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.

[3] 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 $t + 1$.

One can also consider a probabilistic notion of security: given a randomly chosen document, after the random string has passed, the adversary will not be able to stamp the document with more than negligible probability. We note that our notion of security implies this probabilistic notion as well.

Security of a Timestamping Scheme: Given a (T, V)-efficient timestamping scheme. Let M_{max} be the bound on the storage of the most powerful adversary. The scheme is α-optimal ($\alpha > 1$) if, for every $M \leq M_{max}$, no adversary with space M can successfully stamp more than $\alpha \frac{M}{T}$ documents (for a formal definition of "successful stamping" see definition 4).

Notice that the definition above requires α–optimality for every $M \leq M_{max}$. 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).

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).

Theorem 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 $R^{\Omega(1)}$ and allows $R^{\Omega(\eta)}$ rounds.*

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.

Theorem 2. *Assume that there exist collision resistant hash functions. 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 and allows R rounds.*

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 will appear in the full version of the paper.

Advantages of Our Non-interactive Timestamping Scheme

Non-interactive timestamp systems have some significant advantages over the interactive systems known to date. We summarize some of these below:

- The only communication made before the verification process is the transmission of the random string r. This allows the timestamp system to be used in situations where communication is infeasible or undesirable. E.g., communication may be asymmetric: one central agency can broadcast 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 timestamp systems. Even in distributed interactive systems, some measure of trust must be given to third parties. Our non-interactive timestamp 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: 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 has succeeded.

Overview of the "Information-Theoretic" Construction

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 $k < K$ vertices on the left have at least kc neighbors on the right.)[4]

[4] 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

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 $c \approx D$, therefore 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.

However, the argument above is not sufficient. It does not rule out the possibility that the adversary can stamp many documents such that the identity of these documents *depend* on the random string r. Our security definition requires that for every adversary, with high probability (over the choice of r) there do not exist k documents which the adversary can successfully stamp. To prove the security of our scheme we use a "reconstruction argument" and show that any adversary which 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.

Overview of the "Computationally-Bounded" 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

[24, 23]. However, we need graphs with somewhat different parameters. We construct such graphs by combining the constructions of [24] and a slight modification of [23, 21] (which in turn relies on explicit constructions of "randomness extractors" from [21, 22]).

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* [20]. 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.[5] 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.

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.

Online Space: For a family of functions F, we denote by $Space(F)$ the maximum space used by any function in F. 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*.

[5] 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.

Definition 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 $K_{max}D$. We use some existing constructions of unbalanced graphs [24] as well as a modification of [23] to prove the next theorem (the proof will appear in the full version of the paper).

Theorem 3. *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* $\log|V_1| + \log D$*.*

3 One Round Timestamping: The Model

In this section we formally define our model for timestamping in the bounded storage model. The definitions are only for a single round. Definitions for multiple rounds are straightforward generalizations and will appear in the full version.

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 2 (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 (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 4 (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* $\mathrm{Store}^*(r)$.[6] *An adversary γ-*successfully stamps *a document doc at (some fixed) r if*

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

Note that 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 define security as:

Definition 5 (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^*,*

$$\Pr_r\left[A \textit{ can } \gamma\textit{-successfully stamp } \alpha\frac{M^*}{T} \textit{ documents at } r\right] \leq \rho$$

Definition 5 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.

3.1 Security Against Feasibly Generated Documents

Until now, we have allowed the adversary to run in arbitrary time. When the adversary is time-bounded, we can imagine scenarios where Definition 5 does not hold, yet the system is secure because the adversary does not have the computational power to find the documents he can illegally stamp. It makes sense to require security only against "feasibly generated documents". We model feasibly generated documents by a probabilistic polynomial time machine $\mathrm{Generate}^*_{\mathrm{c}}$ which, on input r and an integer k, outputs k documents (all different).

Definition 6 (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 $M^* \leq$*

[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.

M^*_{max}, *every adversary* A *with space* M^*, *and every polynomial time machine* Generate^*_c*:*

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

where the probability is over the choice of r *and the random coins of* Generate^*_c *and* A*.*

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:

- $\text{Stamp}(doc, r) = r_{|\Gamma(doc)}$.
- $\text{Sketch}(r) = \mathcal{H}, r_{|\mathcal{H}}$ where $\mathcal{H}$ has $|\mathcal{H}|$ elements selected at random from $[N]$.
- $\text{Verify}(\text{Sketch}(r), doc, y) = \begin{cases} Accept & \text{Sketch}(r)_{|\mathcal{H}\cap\Gamma(doc)} = y_{|\mathcal{H}\cap\Gamma(doc)} \\ Reject & otherwise \end{cases}$

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 $\mathcal{H} \cap \Gamma(doc)$.

Theorem 4. *Let* G *be a* $(K_{max}, \beta D)$*-expanding graph,* $\gamma > 0$ *and* $g = \frac{1}{5}$*. Fix* n *large enough such that* $n > \log N \ge \frac{1}{g\cdot\beta}$ *and assume that* $\log|\mathcal{DOC}| \le g\beta Dn$*. If* $D|\mathcal{H}| \ge \frac{N}{g\beta}\ln(\frac{1}{\gamma})$ *then the scheme is* $(T = Dn, V = |\mathcal{H}|(n + \log N))$*-efficient and* α*-optimal for* $\alpha = \frac{1}{(1-4g)\cdot\beta}$*,* $\rho = 2^{-g\beta n D}$*,* γ *and* $M^*_{max} = (1-4g)\beta D n K_{max}$*.*

Plugging in parameters, a corollary of this is:

Corollary 1. *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* $\rho = 2^{-R^{\Omega(1)}}$*,* $\gamma = 2^{-R^{\Omega(\eta)}}$ *and* $M^*_{max} = \Omega(R)$*. The timestamping scheme allows stamping documents of length* $R^{\Omega(1)}$*.*

We prove the corollary in the full version of the paper. We remark that a probabilistic argument shows that there exist bipartite graphs of degree D which have expansion $(1 - o(1))D$ and using such non-explicit graphs in our construction (and setting $g = o(1)$) gives $\alpha = (1 + o(1))$ optimality (whereas the theorem below only achieves $\alpha = O(1)$). In the remainder of the section we prove Theorem 4.

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 $V = |\mathcal{H}|(n + \log N)$. 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

In Definition 4 we defined "successful stamping". Without loss of generality, we assume the adversary is deterministic. Let $\mathcal{R}_{\text{successful}}(k) = \mathcal{R}^{\gamma}_{\text{successful}}(k)$ denote the set of random 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 similar notion of "correct stamping":

Definition 7. *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.*

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

The security proof has two parts.

Lemma 1. *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* $M^* \leq (1 - 4g)\beta k Dn$ *we have* $\Pr_r[r \in \mathcal{R}_{\text{correct}}(k)] \leq 2^{-g\beta nDk}$.

We then relate $\mathcal{R}_{\text{correct}}$ and $\mathcal{R}_{\text{successful}}$:

Lemma 2. *Assume* $D|\mathcal{H}| \geq \frac{N}{g\beta} \ln(\frac{1}{\gamma})$. *For every k,* $\mathcal{R}_{\text{successful}}(k) \subseteq \mathcal{R}_{\text{correct}}(k)$.

Together,

Proof. (of Theorem 4) We need show that no adversary with space M^* can γ-successfully stamp more than $k = \alpha \frac{M^*}{Dn}$ documents. Notice that for $M^* \leq M^*_{max}$, $k = \frac{\alpha M^*}{Dn} \leq \frac{\alpha M^*_{max}}{Dn} = K_{max}$ 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 2 and the second inequality follows by Lemma 1. The third inequality is because $k \geq 1$.

4.4 The Proof of Lemma 1

We first define a compression function $\mathrm{Com}(r)$ for $r \in \mathcal{R}_{\mathrm{correct}}(k)$. Let $r \in \mathcal{R}_{\mathrm{correct}}(k)$. 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. Denote $\mathcal{BAD} = \cup_{1 \leq i \leq k} \mathcal{BAD}(doc_i)$, 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 $\mathrm{Com}(r)$ to be:

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

We define a "decompression" function $\mathrm{Dec}(a)$ that gets as input $\mathrm{Com}(r)$ and tries to recover r. Let r be a string from $\mathcal{R}_{\mathrm{correct}}$, 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 $\mathrm{Com}(r)$, we recover the set Γ, and from $\mathrm{Com}(r)$ 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 $\mathrm{Stamp}^*(doc_i, \mathrm{Store}^*(r))$ and take r_j from its output.

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

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. $|\mathrm{Store}^*(r)| \leq M^*$, by definition. As G is expanding and $k \leq K_{max}$, $|\Gamma| \geq \beta k D$ and therefore $r_{|\bar{\Gamma}} \leq R - \beta k D n$. We represent $\mathcal{BAD} \subseteq \Gamma$ by a binary vector of length $|\Gamma| \leq kD$ 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 $|\mathcal{BAD}_{doc_i}| \leq err$ and $|\mathcal{BAD}| \leq k \cdot err$. 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 $k \log |\mathcal{DOC}| + M^* + R - \beta k D n + kD + k \cdot err \cdot n$. We denote this quantity $R - \Delta$.

As every $r \in \mathcal{R}_{\mathrm{correct}}$ has a small description (of length $R - \Delta$) we have $|\mathcal{R}_{\mathrm{correct}}| \leq 2^{R-\Delta}$ and therefore $Pr[r \in \mathcal{R}_{\mathrm{correct}}] \leq 2^{-\Delta}$. We have $kD \leq g\beta k D n$ (for large enough n). We also have $err = g\beta D$ and by our assumption $\log |\mathcal{DOC}| \leq g\beta n D$. Altogether, $R - \Delta \leq R - \beta D k n[1 - 3g] + M^*$. We get that $\Delta \geq (1 - 3g)\beta D k n - M^*$. As $M^* \leq (1 - 4g)\beta k D n$ we get $\Delta \geq g\beta k D n$ as desired.

4.5 The Proof of Lemma 2

Claim. 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 $(1 - \frac{err^*}{N})^{|\mathcal{H}|} \leq e^{-\frac{err^*|\mathcal{H}|}{N}}$ (assuming the set $\mathcal{H}$ is chosen with repetition). 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.

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 very small then the sketch size V is almost all of the random string. Our second scheme has 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. In this section we briefly describe the scheme and give a sketch of the proof. Due to space constraints, the exact details will appear in the full version. We assume the reader has some familiarity with collision resistant hash functions[7] [11] (CRHFs) and Merkle trees [20].

5.1 The Stamping Scheme

Let $H = \{h \colon \{0,1\}^{2n} \mapsto \{0,1\}^n\}$ be a family of CRHFs[8] 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

[7] also called "collision intractable" or "collision free" hash functions

[8] Informally, this means that no computationally bounded adversary can find $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$ when given a random function h in the family. In this paper we require hash functions which are hard even for adversaries which run in time slightly super-polynomial in n. This is because the adversary runs in time polynomial in R, whereas n can be very small compared to R.

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:

Sketch$_c$ The verifier stores r_0 and the root of a Merkle tree whose leaves are $r_1, \dots, r_N$, using the hash function specified by r_0.[9] Note that Sketch$_c$ is *deterministic* (unlike the case of the previous section where Sketch is probabilistic).

Stamp$_c$ Given a document $doc \in \mathcal{DOC}$ the stamper uses the function Stamp of the previous section, and for every $j \in \Gamma(doc)$ stores r_j along with the Merkle-path from r_j to the root of the tree.[10]

Verify$_c$ Given a document doc, a "root" a and a stamp y composed of D Merkle-paths, the function $\text{Verify}_c(a, doc, y)$ accepts iff all paths are valid (that is, the label of the tree root computed from the Merkle-paths is consistent with that stored by the verifier).

We note that both Sketch$_c$ and Stamp$_c$ can be computed online in small space, using the standard method for computing Merkle-trees online. For our choice of parameters, this gives the required efficiency. Using the expander construction of Theorem 3 for G, we obtain a scheme with efficiency $2^{(\log\log R)^{O(1)}}$ (and thus prove Theorem 2). 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 3. It is folklore that such graphs exist by a probabilistic argument. However, at this point no such explicit construction is known. In the remainder of the section we sketch the proof of security of the scheme (The complete proof of Theorem 2 appears in the full version of the paper).

5.2 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 polynomial in R. We use *coins* to denote the concatenation of the random coins used by $Stamp_c^*$ and Generate_c^*. We define $\mathcal{R}_{\text{correct}}^{\text{comp}}(k)$ to be the set of pairs $(r, coins)$ such that, for every $doc \in \text{Generate}_c^*(r, k)$, the Merkle paths output by $Stamp_c^*(Store_c^*(r), doc)$ are correct (i.e. that they are actual paths in the Merkle tree whose leaves are the blocks of r). In particular, this implies that the leaves of the paths are a

[9] Informally, a Merkle tree of $r_1, \dots, r_N$ using the hash function h is a labeled binary tree, where the leaves are labeled by $r_1, \dots, r_N$ and the label of each internal node is given by applying h to the concatenation of its children's labels.

[10] A Merkle-path from r_j consists of r_j along with the labels of the siblings of all nodes on the path from r_j to the root of the Merkle tree. Such as sequence contains sufficient information to compute the labels of all nodes on the path to the root node (by repeatedly applying the hash function).

"correct" timestamp for the k documents output by $\text{Generate}^*_{\text{c}}(r,k)$ in the sense of Section 4.

We now want to define the computational analogue of $\mathcal{R}_{\text{successful}}$ and relate it to $\mathcal{R}^{\text{comp}}_{\text{correct}}$. We define $\mathcal{R}^{\text{comp}}_{\text{successful}}(k)$ to be the set of all pairs $(r, coins)$ for which the adversary successfully stamps the k documents output by $\text{Generate}^*_{\text{c}}(r,k)$ (i.e. for all $doc \in \text{Generate}^*_{\text{c}}(r,k)$, the the Merkle paths output by $Stamp^*_c(Store^*_c(r), doc)$ are accepted by the verifier). This definition of success corresponds to the notion of security in Definition 6.

We prove that $Pr_{r,coins}[(r, coins) \in \mathcal{R}^{\text{comp}}_{\text{successful}}(k) \setminus \mathcal{R}^{\text{comp}}_{\text{correct}}(k)] \leq \text{neg}$ (where neg is a negligible function of n). This is because we can imagine a machine which, when given a random hash function r_0, uniformly selects the pair $(r, coins)$ and runs the adversary. The claim follows, as for every pair $(r, coins) \in \mathcal{R}^{\text{comp}}_{\text{successful}}(k) \setminus \mathcal{R}^{\text{comp}}_{\text{correct}}(k)$, this machine can find a collision for r_0. Thus we have a computational analogue of Lemma 2.

We then show (using Lemma 1) that every random string for which the adversary can correctly stamp many documents can be compressed, which gives a bound on the probability that this occurs.

6 Discussion and 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 the blocks in the intersection differ by a small amount. The proof of Lemma 1 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 timestamp 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 timestamp 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 [25]. 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-bounded" scheme (Section 5) does not.[11] Natural open problems are whether the "information-theoretic" scheme can be improved to yield better parameters, and whether the "computationally-bounded" scheme can be improved to run with strategies that are locally computable.

Acknowledgements

We thank Danny Harnik, 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, 2002.
2. Y. Aumann and M. O. Rabin. Information theoretically secure communication in the limited storage space model. In *Advances in Crypology — CRYPTO '99*, 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, Berlin Germany , New York, 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. Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital signatures. *Lecture Notes in Computer Science*, 765:274, 1994.
6. Ahto Buldas and Peeter Laud. New linking schemes for digital time-stamping. In *Information Security and Cryptology*, pages 3–13, 1998.
7. Ahto Buldas, Peeter Laud, Helger Lipmaa, and Jan Villemson. Time-stamping with Binary Linking Schemes. In Hugo Krawczyk, editor, *Advances on Cryptology — CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 486–501, Santa Barbara, USA, August 1998. Springer-Verlag.

[11] In the "computationally-bounded" 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.

8. Ahto Buldas, Helger Lipmaa, and Berry Schoenmakers. Optimally efficient accountable time-stamping. In *Public Key Cryptography*, pages 293–305, 2000.
9. Christian Cachin, Claude Crepeau, and Julien Marcil. Oblivious transfer with a memory-bounded receiver. In *IEEE Symposium on Foundations of Computer Science*, pages 493–502, 1998.
10. Christian Cachin and Ueli Maurer. Unconditional security against memory-bounded adversaries. In Burton S. Kaliski Jr., editor, *Advances in Cryptology — CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 292–306. Springer-Verlag, 1997.
11. I.B. Damgard. Collision free hash functions and public-key signature schemes. In *Advances in Cryptology — EUROCRYPT '87, Proceedings*, volume 304, pages 203–216. Springer-Verlag, 1987.
12. Yan Zong Ding. Oblivious transfer in the bounded storage model. *Lecture Notes in Computer Science*, 2139:155, 2001.
13. Yan Zong Ding, Danny Harnik, Alon Rosen, and Ronen Shaltiel. Constant-round oblivious transfer in the bounded storage model. In *Theory of Cryptography — TCC '04*, volume 2951, Cambridge, MA, USA, February 2004. Springer-Verlag. To appear.
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. Stefan Dziembowski and Ueli Maurer. Tight security proofs for the bounded-storage model. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 341–350. ACM, May 2002.
16. Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Lecture Notes in Computer Science*, 537:437, 1991.
17. Stuart Haber and W. Scott Stornetta. Secure names for bit-strings. In *ACM Conference on Computer and Communications Security*, pages 28–35, 1997.
18. C. Lu. Hyper-encryption against space-bounded adversaries from on-line strong extractors. In *Advances in Cryptology — CRYPTO '02*, volume 2442, pages 257–271. Springer, 2002.
19. U. Maurer. Conditionally-perfect secrecy and a provably-secure randomized cipher. *Journal of Cryptology*, 5(1):53–66, 1992.
20. Ralph C. Merkle. A certified digital signature. In *Proceedings on Advances in cryptology*, pages 218–238. Springer-Verlag New York, Inc., 1989.
21. R. Raz, O. Reingold, and S. Vadhan. Error reduction for extractor. In *40th IEEE Symposium on Foundations of Computer Science*, pages 191–201, 1999.
22. A. Srinivasan and D. Zuckerman. Computing with very weak random sources. *SIAM Journal on Computing*, 28:1433–1459, 1999.
23. Amnon Ta-Shma. Storing information with extractors. *Information Processing Letters*, 83(5):267–274, September 2002.
24. Amnon Ta-Shma, Christopher Umans, and David Zuckerman. Loss-less condensers, unbalanced expanders, and extractors. In ACM, editor, *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing: Hersonissos, Crete, Greece, July 6–8, 2001*, pages 143–152, New York, NY, USA, 2001. ACM Press.
25. S.P. Vadhan. On constructing locally computable extractors and cryptosystems in the bounded storage model. In *Advances in Cryptology — CRYPTO '03*. Springer, 2003.