Proof of Freshness: How to efficiently use an online single secure clock to secure shared untrusted memory.

Marten van Dijk, Luis F. G. Sarmenta, Charles W. O'Donnell, and Srinivas Devadas

MIT Computer Science and Artificial Intelligence Laboratory (CSAIL)
[*]
{marten,lfgs,cwo,devadas}@mit.edu

Abstract

We address the problem of using an untrusted server with a small trusted module to provide trusted storage for a large number of clients, where each client may own and use several different devices that may be offline at different times and may not be able to communicate with each other except through the untrusted server. We introduce a new cryptographic primitive: *freshness schemes*. We show and prove how the primitive can be used to implement tamper-evident trusted storage for a large number of users using a single constant-sized trusted *secure clock* and constant communication cost per operation, regardless of the number of clients and devices per client.

Keywords: untrusted storage, freshness, integrity checking, authenticated search tree, certificate list, secure clock, TPM

1 Introduction

In this paper, we address the problem of using an untrusted server with a small trusted module to provide trusted storage for a large number of clients, where each client may own and use several different devices that may be offline at different times and may not be able to communicate with each other except through the untrusted server. The challenge here is not in guaranteeing the privacy or integrity of a client's data, but in guaranteeing the data's *freshness*. The privacy of a client's data can easily be achieved through encryption, while integrity can be protected by use of digital signatures or message authentication codes (MACs). The freshness of data, however, is more difficult to guarantee. If a client's data is stored in untrusted memory or disk space, then even if the client encrypts the data and uses a signature or MAC, there is no way to prevent an adversary with access to the untrusted memory or disk space from performing a *replay attack*. That is, while the client is offline, the adversary can replace the current version of the data with an older version copied by the adversary at a previous time, effectively allowing it to rewind the client's data to a previous state.

In this paper we focus on providing *tamper-evident* storage, where clients are guaranteed to at least *detect* illegitimate modifications to their data, including replay attacks. Providing *tamper-tolerant* storage, which guarantees that a client can continue to retrieve its original data even after a malicious attack, is a harder problem, but can generally be solved by using some form of data replication on top of a tamper-evident storage system.

Replay attacks are not a problem if there exists a secure communication channel between all of a client's devices all the time. If this is the case, a global counter to be used for timestamping can be synchronized among each of the devices. Every time a client device updates data stored in untrusted memory, it keeps track of when that update occurred by reading and increasing the global counter. It computes and saves in untrusted memory a small constant-sized digest or root hash of the data timestamped with the increased global counter (through

[*]This work was done under the MIT-Quanta T-Party project, funded by Quanta Corporation.

well-known techniques such as Merkle hash trees [37]). Through the secure channel, the device broadcasts the increased global counter to all other devices such that the devices' copies of the global counter get synchronized. If all of the devices always receive the global counter at the end of each update, then any device retrieving the data from untrusted storage can verify freshness by simply recomputing a root hash from the retrieved data timestamped with the current value of the global counter, and comparing it against the stored root hash which corresponds to the most recent timestamp.

Replay attacks may become a problem, if a client's different devices cannot communicate directly or *if they are not all online at the same time*. In this case, it is not immediately clear how to construct a secure communication channel for synchronizing a global counter. Of course, if there exists a trusted server, which

- stores the global counters of each of the clients, and which
- is online all the time,

then this trusted server can be used as a secure communication channel. That is, at the end of each update the corresponding client's global counter is stored at the trusted server such that any device who wishes to check the freshness of retrieved data can request the current value of the global counter and proceed as before.

In this paper we show that it is not necessary to assume a fully trusted server. We only need to assume an untrusted server with access to a small trusted module with limited computational capabilities, which

- can be accessed with a few commands, and which
- has small constant sized trusted space.

In order to prevent replay attacks in which the trusted module is reset to one of its previous states, the trusted module must have an *irreversible* clock counter in its trusted space. We will introduce different schemes which use a *single* irreversible clock counter to maintain the *multiple* global counters for each of the clients. Since the trusted module has only a constant sized trusted space, it must be able to certify the multiple global counters such that these can be stored in untrusted memory at the server. In order to be able to certify global counters, the trusted module keeps a public-private key pair in its trusted space and has at least one command for generating digital signatures.

We propose to use the trusted module as follows. At the end of an update by a client's device, the device increments the client's global counter. Since the server itself cannot be trusted, the device asks the trusted module to sign the global counter together with the current value of its clock counter. In other words, the trusted module timestamps the client's global counter. Any device retrieving the client's global counter from the untrusted server, asks the server for the timestamped global counter and asks for a proof that no more recent values of the trusted module's clock counter have been used to timestamp the client's global counter. This proof is constructed in interaction with the trusted module. How to efficiently construct this *proof of freshness* is the topic of this paper.

Our main contribution is the definition of a *freshness scheme* which can be used to maintain global counters for multiple clients. We will introduce new techniques to construct freshness schemes which can be implemented with a small trusted module. There are many applications where our solution is of practical relevance, not only in file or message storage, but also in mobile payment and e-wallet applications. Furthermore, we are focusing on solutions with a small trusted module. First, it means that it is easier and cheaper for a manufacturer to produce a smaller trusted module than a larger one. Second, existing trusted platforms may be suitable for implementing our trusted module with its clock. In particular, some of the presented schemes in Section 4 can be implemented using the TPM 1.2 chip [39, **?**, 48], which is increasingly being integrated into end-user machines today. Finally, our techniques can also be of interest to designers of future versions of such trusted platforms.

The paper is organized as follows. Section 2 describes related work and summarizes our contributions. The small trusted module, freshness schemes, and a protocol which maintains global counters are defined in Section 3. We then present and analyze the performance and security of different schemes in Section 4. We conclude in Section 5.

2 Related Work and Our Contributions

Protecting and validating the integrity of data storage has been a well studied topic due mainly to its high importance to such a wide range of applications. For this, cryptographic one-way hash functions [17] are often used by a client to create a small local checksum of large remote data. Merkle proposed hash trees as a means to update and validate data hashes efficiently by maintaining a tree of hash values over the objects [37]. Recent systems [16, 19, 13, 31, 35] make a more distinct separation between untrusted storage and a trusted compute base (TCB), which can be a trusted machine or a trusted coprocessor. These systems run a trusted program on the TCB that uses hash trees to maintain the integrity of data stored on an untrusted storage. The untrusted storage is typically some arbitrarily large, easily accessible, bulk store in which the program regularly stores and loads data which does not fit in a cache in the TCB.

Byzantine-fault-tolerant file systems [11, 12] consist of storage state machines that are replicated across different nodes in a distributed system. The system provides recovery from tampering, but it relies on the assumption that at least two-thirds of the replicas will be honest. The expectation is that replicas are weakly protected but not hostile, so the difficulty of an adversary taking over k hosts increases significantly with k. Byzantine-fault-tolerant file systems distribute trust but assume a threshold fraction of honest servers.

SUNDR [36, 29] is another general-purpose, multi-user network file system that uses untrusted storage servers. SUNDR protects against *forking attacks* which is a form of attack where a server uses a replay attack to give different clients a different view of the current state of the system. SUNDR does not prevent forking attacks, but guarantees *fork consistency*, which essentially ensures that the system server either behaves correctly, or that its failure or malicious behavior will be detected later when users are able to communicate with each other. This is achieved by basing the authority to write a file on the public keys of each client. Plutus [25] is another efficient storage system for untrusted servers that cannot handle these forking attacks.

In our system, we place a small TCB at a server for maintaining certificates of global counters for timestamping to allow clients to immediately detect misbehavior, including replay attacks. This allows us to prevent forking attacks and guarantee the freshness, integrity, and consistency of data.

Our main contribution is a new primitive which we we call a *freshness scheme*. Freshness schemes are concerned with providing proof that something did *not* happen at a particular time. This is complementary to digital notarization developed in works such as Haber and Stornetta on time-stamping [3, 24]. Time-stamping provides proof that something existed or happened at a particular time.

In accountable time-stamping systems [9, 10] all forgeries can be explicitly proven and all false accusations explicitly disproven; it is intractable for anybody to create a pair of contradictory attestations. Buldas et al. [8] introduced a primitive called *undeniable attester*. Informally, the attester is such that it is intractable to generate both a positive and a negative attestation based on an element x and a set S such that the attester concludes $x \in S$ for the positive attestation and $x \notin S$ for the negative attestation. Their intention is to prove the existence or non-existence of objects in a database. Freshness schemes prove the existence or non-existence of events in a time interval. As opposed to future objects in a database, future time values can be predicted. In freshness schemes this property can be exploited and leads to significant performance improvements.

In this paper, we show how authenticated search trees [26, 40, 8, 7], which can be used for the construction of an undeniable attester, can also be used as a building block to create freshness schemes. We propose two solutions; the “tree-based” scheme and the “log-based” scheme. Both solutions efficiently exploit the property that future time values can be predicted. The performance of the tree-based scheme depends mainly on the processing speed of the trusted module (there is only a small amount of communication between the server and the client devices). Since the trusted module is assumed to be small and cheap and therefore slow, we prefer the log-based scheme for which the performance mainly depends on the network bandwidth. We introduce two solutions in order to bound the amount of communication between a client and the server in the worst-case.

Our techniques reduce the trusted computing base to only a single secure clock with a small instruction set. This differs from many other systems that require complex secure processors [44, 30, 49, 46, 47, 45]. By using several tricks, we have shown that our best performing schemes can even be implemented by using a standard component on machines today, TPM 1.2 [39, **?**, 48] from the Trusted Computing Group. (We notice that in [43], Shapiro and Vingralek address the problem of managing persistent state in Digital Rights Management (DRM) systems. They include volatile memory within their security perimeter, which increases the TCB.)

Our secure clock is a secure counter which can be used to timestamp events according to their causal relations. An order in logical time can be extracted if we know which events logically cause other events. Lamport clocks [27, 28] are conceptual devices for reasoning about event ordering. Our scenario with a centralized untrusted server with access to the secure clock does not need Lamport clocks to reason about logical time. Our difficulty is how to reason about malicious behavior.

For completeness, we note that a certificate revocation list (CRL) is a list which is signed by a certification authority (CA) together with a timestamp. By obtaining the most recent CRL one can easily verify whether a certificate is still valid. This is a general principle which can also be used in the construction of freshness schemes. Based on CRLs, less communication intensive solutions are proposed in [38, 26, 40]. In [38] the idea is to sign a message for every certificate stating whether it is revoked or not and to use an off-line/on-line signature scheme [18] to improve the efficiency. The work on certificate authentication trees in [26] has led to the introduction of authenticated dictionaries [40] and authenticated search trees [8, 7]. A persistent authenticated dictionary [1, 22, 32, 33] maintains multiple versions of its contents as it is modified. Timeline entanglement [34] creates a tamper-evident historic record of different persistent authenticated dictionaries maintained by mutually distrusting servers.

Authenticated dictionaries are mainly implemented by using authenticated search trees or skip lists [22]. Another approach is described by Goodrich et al. [23] which uses a one-way accumulator [6, 2, 20, 15]. An insertion takes $O(1)$ time and a deletion takes $O(n)$ time, where n is the size of the dictionary. The computation performed to answer a single query takes $O(n)$ time. By using precomputed accumulations, the computation performed to answer a single query can be reduced to $O(1)$ time at the cost of increasing the cost of an insertion to $O(n)$ time.

One technique also worth noting is that described by Schneier and Kelsey for securing audit logs on untrusted machines [41, 42]. Each log entry contains an element in a linear hash chain that serves to check the integrity of the values of all previous log entries. It is this element that is actually kept in trusted storage, which makes it possible to verify all previous log entries by trusting a single hash value. The technique is suitable for securing append-only data that is read sequentially by a verifying trusted computer.

3 Freshness Schemes

3.1 Definitions

Even though devices can be offline with respect to one another, each device is able to contact an untrusted server with a small trusted module. The small trusted module has a secret key and maintains an irreversible clock counter. Devices use the server to reserve clock counter values and provide "proofs of freshness". By using the trusted module with its counter, the server maintains a data structure with, for each client i, the counter value of the most recent time one of i's devices reserved a counter value. A "proof of freshness" for client i proves the correct value of the most recent reserved counter value of i. This means that the most recent reserved clock counter value of client i can be used as a global counter which can be accessed by each of i's devices. As explained in the introduction, the global counter can be used to timestamp data and provide tamper-evident storage.

The next definition formally defines the different algorithms for generating key pairs (G), updating the server's data structure (U), reserving clock counter values (R), proving freshness (P), and verifying reservations and proofs of freshness (V_R and V_P). The trusted module implements U, the server executes R and P, and the client devices use V_R and V_P. We require that if a sequence of updated data structures is generated by U, then

1) this sequence can be mapped to a time line, 2) the updates correspond to (shared) reservations of time values, the corresponding reservation certificates generated by R can be verified by V_R, and 3) each update can be used by P to generate a proof of freshness for a client i proving the correct value of the time value of the most recent reservation of i (this can be verified by V_P).

These requirements are detailed in the next definition, which follows [21, 4, 5]; U^O denotes a probabilistic oracle algorithm which makes calls to a specific oracle O, when talking about U^O we may omit the superscript O, the notation $x \leftarrow U$ means that probabilistic algorithm U outputs x. Let $\mathcal{I}$ be the set of client identities. Let $\mathcal{C}_i$ index the set of devices of client $i \in \mathcal{I}$. Let $\mathcal{IC}$ be the product $\{(i,d) : i \in \mathcal{I}, d \in \mathcal{C}_i\}$. Let $\mathcal{D}$ describe the set of possible data structures. The definition of V_P relies on an additional input to V_P; a scheduling algorithm S, which is important in the construction of efficient freshness schemes, see Section 4.

Definition 1 *Let H be a hashing oracle and let $Sign^H(sk,.)$ be a signing oracle. A freshness scheme $\mathcal{F} = (G, U, R, V_R, P, V_P)$ is a set of probabilistic oracle algorithms, where*

1. *G is a key generation algorithm that, given the security parameter k, computes a key pair $(pk, sk) \leftarrow G^H(1^k)$ while possibly making calls to a hashing oracle H (pk is called public key and sk is called secret key or private key),*

2. *U is an updating algorithm: given a data structure $D \in \mathcal{D}$, a set of device identity pairs $I \subseteq \mathcal{IC}$, and a secret key sk, it computes an updated data structure D' while possibly making calls to a hashing oracle H and a signing oracle $Sign^H(sk,.)$;*

$$D' \leftarrow U^{H, Sign^H(sk,.)}(D, I),$$

3. *R is reservation algorithm: given a data structure D and an identity $i \in \mathcal{I}$, it returns a reservation certificate r; $r \leftarrow R(D, i)$,*

4. *V_R is a verification algorithm: given a reservation certificate r, identity $i \in \mathcal{I}$, $d \in \mathcal{C}_i$, and public key pk, it outputs a time value or the phrase "reject",*

5. *P is a proving algorithm: given a data structure $D \in \mathcal{D}$ and an identity $i \in \mathcal{I}$, it returns a freshness proof p; $p \leftarrow P(D, i)$,*

6. *V_P is a verification algorithm: given a freshness proof p, identity $i \in \mathcal{I}$, a scheduling algorithm S, and public key pk, it outputs a time value or the phrase "reject".*

Let $(pk, sk) \leftarrow G^H(1^k)$ for some k. We require that there exists

1. *a partial ordering $(\mathcal{T}, <)$ with the property if $a < b$ and $a < c$ then either $b < c$ or $c < b$ (hence, if there exists an element $> a$ then there exists a unique minimal solution $''a + 1''$ such that $a <'' a + 1''$), and*

2. *a class of scheduling algorithms $\mathcal{S} \subseteq \mathcal{T} \times \mathcal{T} \rightarrow \{''accept'','' reject''\}$ (notice that, for $S \in \mathcal{S}$ and $t \in \mathcal{T}$, there is either no solution or a unique minimal solution $t' > t$ such that $S(t, t') ='' accept''$)*

such that, for data structures D_j and sets of device identity pairs I_j, $j \geq 0$, satisfying the recurrence relation

$$D_{j+1} \leftarrow U^{H, Sign^H(sk,.)}(D_j, I_j), \;\; j \geq 0,$$

there exists a timing algorithm $T : \mathcal{D} \rightarrow \mathcal{T}$ such that, for $j > 0$:

1. *$T(D_j)$ is the result of a single increment of $T(D_{j-1})$ with respect to the ordering relation $<$ (that is, $T(D_j) ='' T(D_{j-1}) + 1''$).*

2. *For any $(i,d) \in \mathcal{IC}$, if $(i,d) \in I_{j-1}$ and $(i,d') \notin I_{j-1}$ for devices $d' \neq d$, then*

$$T(D_j) \leftarrow V_R(R(D_j,i),i,d,pk).$$

   If this condition does not hold, then $V_R(R(D_j,i),i,d,pk)$ returns $''reject''$ and we call the reservation certificate $R(D_j,i)$ not valid on (i,d) with respect to pk.

3. *For any $i \in \mathcal{I}$ and scheduling algorithm $S \in \mathcal{S}$, there exists an index m such that*

$$(T(D_m),T(D_j)) \leftarrow V_P(P(D_j,i),i,S,pk)$$

   and m has the property that

$$[m > 0] \;\Rightarrow\; [\exists_{d \in \mathcal{C}_i}(i,d) \in I_{m-1}]$$

   and, for $m < h \leq j$,

$$\left[S(T(D_m),T(D_h)) ='' \, accept''\right] \;\Rightarrow\; \left[\forall_{d \in \mathcal{C}_i}(i,d) \notin I_{h-1}\right].$$

   (Given j, the implied condition may have multiple solutions m, hence, V_P's output does not necessarily need to correspond to the minimal one.) We call a freshness proof p not valid on (i,S) with respect to pk if $V_P(p,i,S,pk)$ returns $''reject''$.

We say that the freshness scheme is valid with respect to the partial ordering $(\mathcal{T},<)$ and class of scheduling algorithms $\mathcal{S}$.

The next definition formalizes the security requirement that 1) there is no practical algorithm that, with oracle access to U and without knowing sk, generates a valid freshness proof p with $V_P(p,i,\ldots) = (t_m,t_j)$ (which certifies that, with respect to time t_j, t_m is the most recent time reserved by i) as well as a valid reservation certificate r with $V_R(r,i,\ldots) = t_h$ (which certifies that time t_h was reserved by i) with $t_m < t_h \leq t_j$, and 2) there is no practical algorithm that, with oracle access to U and without knowing sk, generates valid reservation certificates for the same time value but for two different devices of the same client. The main assumption is that the update algorithm U as implemented by the trusted module does not generate two different data structures for the same time. In the next subsection we explain how this can be enforced by using the trusted module's irreversible clock counter.

Definition 2 *Let $(pk,sk) \leftarrow G^H(1^k)$ for some k. We say a probabilistic oracle algorithm M with k and pk as input succeeds if*

$$(p,r,r') \leftarrow M^{H,Sign^H,U^{H,Sign^H(sk,.)}}(1^k,pk),$$

M does not query the update oracle $U^{H,Sign^H(sk,.)}$ such that it outputs two different data structures D and D' with $T(D) = T(D')$ (that is, there is a one to one mapping between updated data structures and time values), and one of the following conditions hold:

1. *There exists a scheduling algorithm S, a client i with a device d, times t_m, t_h, and t_j, such that $t_m < t_h \leq t_j$,*

$$(t_m,t_j) \leftarrow V_P(p,i,S,pk),$$

$$t_h \leftarrow V_R(r,i,d,pk),$$

   and $S(t_m,t_h) ='' \, accept''$.

2. *There exists a client i with devices d and d', and a time t such that*

$$t \leftarrow V_R(r, i, d, pk),$$

$$t \leftarrow V_R(r', i, d', pk),$$

and $d \neq d'$.

We say M $(\tau, q_h, q_s, q_u, \epsilon, \delta)$-breaks the freshness scheme if, for security parameter k, its running time (plus size of description) does not exceed $\tau(k)$, the number of queries to the hashing oracle does not exceed $q_h(k)$, the number of queries to the signing oracle does not exceed $q_s(k)$, the number of its queries to the update oracle does not exceed $q_u(k)$, and with probability at least $\delta(k)$, $G(1^k)$ generates a key (pk, sk) such that, for any i, the probability of M's success on input $(1^k, pk)$ is at least $\epsilon(k)$. We say the freshness scheme is $(\tau, q_h, q_s, q_r, q_p, \epsilon, \delta)$-secure if no M $(\tau, q_h, q_s, q_r, q_p, \epsilon, \delta)$-breaks it.

In practice, the update algorithm U is implemented by the untrusted server as a protocol with the trusted module in which the trusted module executes a sequence of atomic instructions. In this paper, we analyze the performance of freshness schemes for which the server starts the first half of the execution of U after which the trusted module is asked to execute a series of atomic instructions. In these protocols, we assume that the trusted module has some small constant trusted state which can be used to implement a finite state machine which enforces sequences of atomic instructions to correspond to the second half of the execution of U. Hence,

$$U^{H, Sign^H(sk,.)}(D, I) = U_2^{H, Sign^H(sk,.)}(U_1(D, I)),$$

where U_1 is executed by the untrusted server and U_2 is executed by the trusted module. To capture this scenario in our definition, we should allow M to query the oracle $U_2^{H, Sign^H(sk,.)}$. The irreversible clock counter of the trusted module can be used to enforce that there is a one to one mapping between updated data structures and time values. Therefore, it is reasonable to assume that M cannot query $U_2^{H, Sign^H(sk,.)}$ such that it outputs two different data structures D and D' with $T(D) = T(D')$. In forthcoming proofs, we will implicitly use this slightly stronger definition of security.

Besides the atomic instruction which implement the update algorithm U, the trusted module may allow the execution of many more other instructions for other purposes. Formally, we should extend our definition such that M has oracle access to these other instructions as well.

3.2 Application

Let $\mathcal{F} = (G, U, R, V_R, P, V_P)$ be a valid freshness scheme implemented by an untrusted server with a trusted module with a single secure clock. We will explain how devices may use $\mathcal{F}$ to reserve clock counter values and generate freshness proofs of the most recent reserved values.

We assume that the clock counter is irreversible (that is, no command will reset the clock counter to an older value). For example, the clock counter can be implemented as a monotonic counter or as a *hash-clock* wherein the next counter value is determined by extending the current clock value (by using a one-way and collision-resistant hash function on the concatenation of the current value and some default "increment" value). We define $t' > t$ if t' can be computed as the result of multiple increments of a clock counter value t. In the case of a hash clock, the preimage resistance of the hash function implies that it is intractable to compute counter values t, t', and t'' such that $t < t' < t'' < t$. For this reason, we will assume that $<$ is a partial ordering as in Definition 1.

If the trusted module generates $D' \leftarrow U^{H, Sign^H(sk,.)}(D, I)$, then we require that $T(D') ='' T(D) + 1''$, where $T(D)$ is the value of the clock counter before the update and $T(D')$ is a single increment of the clock counter $T(D)$ by the trusted module. This means that the trusted module cannot produce two different data structures D' and D'', $D' \neq D''$, with the same time value $T(D') = T(D'')$. So, Definition 2 can be used to describe the security of the freshness scheme if the trusted module is used as an oracle by an adversary.

Devices will use the freshness scheme to reserve clock counter values and to generate freshness proofs of the most recent reserved values. A freshness proof, which verifies to a pair of time values (t_m, t_j), can be used to conclude that the most recent reservation happened at t_m if t_j corresponds to the current clock counter value. If not, then more recent reservations may have happened after the moment the clock counter was equal to t_j. In order to recover and verify the value of the current clock counter value, we assume that the trusted module can be asked to produce a *current time certificate*,

$$[t_c, Sign(sk, H(Nonce||t_c))],$$

where the signature is of the hash of an input nonce $Nonce$ and the clock's current value t_c with the secret key sk. By using a random input nonce, old certificates cannot be used to fake a new one.

Verification and Reservation Protocol: Suppose that device d of client i wishes to obtain and verify the most recent time value reserved by a device of client i. Then it proceeds as follows:

1. Device d sends a nonce $Nonce$ together with a (signed) request to the server.

2. The server collects and transmits to d a current time certificate, proof of freshness, and a confirmation of the most recent reservation:

   - The server asks the trusted module to compute a current time certificate

     $$[t_c, Sign(sk, H(Nonce||t_c))].$$

   - The server computes the freshness proof $p \leftarrow P(D, i)$, where D is the most recent data structure maintained by the server (hence, $T(D) = t_c$).
   - The server retrieves the confirmation of i's most recent reservation of a time value: $[S, t, Sign(sk_i, H(S||t))]$ (we assume that client i has its own key pair (pk_i, sk_i)).

3. The device verifies the current time certificate and confirmation and accepts t as the time value of i's most recent reservation if $(t, t_c) \leftarrow V_P(p, i, S, pk)$.

In order to reserve a new time value, device d and the server continue the protocol as follows:

1. Both device d and the server compute t_n as the minimal solution $t_n > t_c$ such that $S(t, t_n) ='' accept''$ (we assume that S is such that a minimal solution exists). If one of the other devices of i already requested to reserve t_n, then the server tells device d to try again later.

2. The server waits for the arrival of timeslot t_n. The server computes the set $I \subseteq \mathcal{IC}$ as the collection of all the device identity pairs who have outstanding reservation requests for t_n. It uses I to update the data structure D to $U(D, I)$. By using the new D, the server computes a reservation request for each pair in I. In particular, it computes and transmits $r \leftarrow R(D, i)$ to device d.

3. Device d agrees that it reserved t_n if $t_n \leftarrow V_R(r, i, d, pk)$. If t_n is accepted, then it chooses a new scheduling algorithm S' and transmits a confirmation

   $$[S', t_n, Sign(sk_i, H(S'||t_n))]$$

   to the server.

Notice that the requirements in Definition 1 show that if the protocol is executed without malicious behavior, then in the protocol device d of client i accepts t as the time value of i's most recent reservation and accepts t_n as a new reserved time value which is not reserved by any of the other devices of i.

The next theorem proves the security of the verification and reservation protocol. A sketch of its proof is in Appendix A.

Theorem 1 *Suppose that the verification and reservation protocol uses a secure and valid freshness scheme. a) If device d of client i accepts t_c as the current time value and t as the time value of the most recent reservation, then none of the devices of client i reserved any of the time values t_r in the range $t < t_r \leq t_c$. b) If in addition t_n is accepted as the time value of the new reservation, then devices of client i did not reserve any of the time values t_r in the range $t_c < t_r < t_n$. Moreover, only device d and no other device of client i reserved t_n.*

In order to be precise, we should state the theorem as follows: if the used freshness scheme is $(\tau, q_h, q_s, q_r, q_p, \epsilon, \delta)$-secure, then there does not exist a probabilistic oracle algorithm M with k and pk as input with the properties that: It cannot query the update oracle $U^{H, Sign^H(sk,.)}$ such that it outputs two different data structures D and D' with $T(D) = T(D')$. Its running time (plus size of description) does not exceed $\tau(k)$, the number of queries to the hashing oracle does not exceed $q_h(k)$, the number of queries to the signing oracle does not exceed $q_s(k)$, and the number of its queries to the update oracle does not exceed $q_u(k)$. Finally, with probability at least $\delta(k)$, $G(1^k)$ generates a key (pk, sk) such that M can be used (by the untrusted server) to force the verification and reservation protocol to violate either a) or b) with probability at least $\epsilon(k)$.

Remark. The presented verification and reservation protocol is just one example. If the trusted module has two secure clocks, then a more complex protocol based on two freshness schemes is possible. We may start with the verification protocol for the first freshness scheme. If, for the first scheduling algorithm, a minimal solution $t_n > t_c$ with $S(t, t_n) ='' accept''$ exists, then the reservation protocol for the first freshness scheme is executed. If such a minimal solution does not exist (because $S(t, t_n) ='' reject''$ for all $t_n > t$), then the verification and reservation protocol for the second freshness scheme is executed. After its completion, if V_P of the second freshness scheme returned a small enough time interval, then the current time value corresponding to the first freshness scheme is reserved as well. In the complex protocol, all confirmations should sign the most recent reserved time values of both freshness schemes.

The complex protocol is secure and can be generalized to multiple secure clocks and freshness schemes. The scheduling algorithms for the different freshness schemes can be designed in such a way that clients who reserve often mainly use the first freshness scheme, clients who do not reserve often mainly use the second freshness scheme, clients who reserve even less often mainly use the third freshness scheme, etc. The protocol adaptively moves clients from one freshness scheme to another depending on the frequency at which they have been requesting reservations. The secure clocks can be incremented at different relative speeds. The speeds can be designed to optimize the performance of the protocol.

Remark. Replication of the reservation service over multiple distrusting servers can be exploited as follows. At each server, each client maintains a copy of its global counter signed together with the identity of the server and the most recent reserved time value at the server. If a device of client i wishes to read and increment i's global counter, then the device will connect to as many servers as possible and run the verification and reservation protocol with each. The device retrieves the most recent value of i's global counter if it connects to one of the honest servers who executed and received the most recent reservation request of client i. Notice that this condition is implied by the stronger condition that at all times a majority of the servers are honest and online.

4 Basic Constructions

We present two basic constructions of freshness schemes and we show how the use of scheduling algorithms improves their performance.

4.1 Tree-Based Scheme

A freshness scheme can be constructed by means of an authenticated search tree [8, 7] as follows. We remind the reader that a directed binary tree is a search tree [14] if every node n in the tree is associated to a unique search key $k[n]$ such that if n_L is the left child of n then $k[n_L] < k_n$ and if n_R is the right child of n then $k[n] < k[n_R]$ (here, $<$ is an ordering over keys). A tree is an authenticated search tree if the labels $l[n]$ of nodes n in the tree are recursively defined by

$$l[n] = H(v[n]) \text{ with } v[n] = (l_L, k[n], l_R),$$

where $l_R = l[n_R]$, if n's right child n_R exists, and $l_R = nil$, otherwise, and where $l_L = l[n_L]$, if n's left child n_L exists, and $l_L = nil$, otherwise. We call $v[n]$ the value of node n. As explained in [7], a sequence

$$s = (h_L, k, h_R; k_1, h_1; k_2, h_2; \dots; k_m, h_m) \tag{1}$$

corresponds to a path from the node with value $v = (h_L, k, h_R)$ to the root if the root's value is equal to the value v_m which is recursively defined by $l_0 = H(v)$ and, for $0 < j \leq m$,

$$l_j = H(v_j) \text{ with } v_j = \begin{cases} (l_{j-1}, k_j, h_j), & \text{if } k < k_j, \\ (h_j, k_j, l_{j-1}), & \text{if } k_j < k. \end{cases} \tag{2}$$

Let data structure D be an authenticated search tree with root $Root$ together with a time value t_c and a signature $Sign(sk, H(t_c||Root))$. Each node in the authenticated search tree corresponds to a unique identity i in $\mathcal{I}$. If the most recent reservation of client i was made by device d at time t, then the search key of the node corresponding to i is equal to the triple (i, d, t). The keys are ordered with respect to their identities, that is, $(i, d, t) < (i', d', t')$ if and only if $i < i'$.

The authenticated search tree of the updated data structure $D' \leftarrow U(D, I)$ is defined as the authenticated search tree of D, where, for $(i, d) \in \mathcal{I}$, the node corresponding to i has its key updated to $(i, d, t_c + 1)$. In order to update a key, the corresponding label needs to be updated, and this affects the labels of the nodes along the path to the root as explained by recursion (2). The time value of D' is equal to $t_c + 1$ and the signature of D' signs the hash of the new root together with $t_c + 1$.

A trusted module may implement the update algorithm $U(D, I)$ as a sequence of atomic instructions. First, the server loads $[Root, t_c, Sign(sk, H(t_c||Root))]$ into the trusted module. The trusted module knows that an update is about to start and verifies the signature (such that it cannot be used as an oracle to produce updates of arbitrary structures). Second, for each $(i, d) \in I$, the server inputs the entries of the sequence (1) that corresponds to the path from the node which corresponds to i to the root. The server also feeds d into the trusted module. For each node along the path, the trusted module uses the recurrence relation (2) to verify whether the the sequence corresponds to the path from the node which corresponds to i to the root and to compute the new root of the updated path. The trusted module stores the new root instead of the current one and updates the next path corresponding to the next device index pair in I. As soon as all paths are updated, the trusted module signs the new root with the time value $t_c + 1$.

A reservation certificate on i and a proof of freshness on i have the same form. Both are equal to the sequence (1) corresponding to the path from the node which corresponds to i to the root together with the signature of the root and the time value t_c of the data structure. This is verified by using (2) in order to compute the root's value and by using the signature to verify its correctness. If the check passes, then, regardless of the inputted scheduling algorithm, V_P returns (t, t_c) for t defined by $k = (i, d, t)$ in (1). For a reservation certificate, V_R returns t if $t = t_c$.

Security: Appendix B presents a sketch of a proof that the resulting freshness scheme is secure if H is a collision resistant hash function and $Sign$ corresponds to a secure signature scheme.

Performance: Let N be the total number of clients. Since reservation certificates and proofs of freshness have size $O(\log N)$, the advantage of the tree-based scheme is the limited communication between the server and client[1]. The disadvantage is the amount of computation by the trusted module; $U(D; I)$ costs $O(|I| \log N)$ atomic operations.

Instead of one authenticated search tree, we propose to use q authenticated search trees, each corresponding to N/q clients. The trusted module updates the trees in turn. We use a single secure clock and assign periodic time values to each tree for updates[2]. Each client uses a scheduling algorithm in the verification and reservation protocol which corresponds to the assigned time values of the tree to which the client belongs. This means that a proof of freshness on i only consists of a proof of freshness associated to the tree to which i belongs. In other words, the communication costs stay the same, while the computation of $U(D, I)$ is reduced to $O(1+|I| \log N/q)$ atomic operations (we need to count the signing of the root, which is the major contribution if $q \approx N$). Notice that the scheduling algorithm is used for time-multiplexing with period q. In order to allow different periods for the different trees and to adaptively insert and delete clients from trees, one should use balanced authenticated search trees [34] (a balanced binary tree is for example a red-black tree [14]).

Appendix B presents a precise performance analysis leading to the following result.

Theorem 2 *Suppose that, for each client, u_γ is the probability that one of its devices requests to start the verification and reservation protocol within the amount of time needed for the computation of one atomic instruction by the trusted module (so, we assume that u_γ is the same for each client). Then, the tree-based scheme with time-multiplexing can serve up to $1/(2u_\gamma)$ clients.*

4.2 Log-Based Scheme

The trusted module may be slow such that the bound in Theorem 2 on the total number of clients which can be served by the tree-based scheme may be too restrictive. For this reason we introduce a straightforward log-based scheme which performance relies on the network bandwidth.

Let data structure

$$D = (s[1], T[1], s[2], T[2], \ldots, s[t_c], T[t_c]; t_c)$$

be the current time value t_c together with a sequence of signatures and authenticated search trees such that $s[t] = Sign(sk, H(t||Root[t]))$ where $Root[t]$ denotes the root of the authenticated search tree $T[t]$. If the device identity pairs in $I \subseteq \mathcal{ID}$ reserved time value t, then $T[t]$ is an authenticated search tree with nodes corresponding to the identities in I. For $(i, d) \in I$, the search key of the node corresponding to i is equal to (i, d). The keys are ordered with respect to their identities.

The update algorithm $U(D, I)$ increments the current time value and appends to D an authenticated search tree for I together with a signature of its root. The first half of the execution of U in which the to be appended authenticated search tree is constructed is done by the server. The second half of the execution of U in which the current time value is incremented and the root of the tree is signed is done in one atomic operation by the trusted module.

The reservation certificate on i for an update at time value t is equal to the sequence (1) which corresponds to the path in $T[t]$ from the node which corresponds to i to the root together with the signature $s[t]$ and time value

[1]The communication between the server and clients can be reduced to single signatures if the trusted module implements V_R and V_P and signs their results together with a nonce.

[2]A separate clock for each tree does not improve the performance.

t. This is verified by using (2) in order to reconstruct the root's value and by using the signature to verify its correctness.

A proof of freshness on (i, S) is a log of sequences corresponding to paths in $T[t_1]$, $T[t_2]$, ..., $T[t_j]$ for some time values $t_1 < t_2 < \ldots < t_j$ together with the signatures $s[t_1]$, $s[t_2]$, ..., $s[t_j]$ and a time value $t_0 < t_1$ such that:

- For $1 \leq m \leq j$, the sequence $(h_L, k, h_R; k_1, h_1; k_2, h_2; \ldots)$ for time value t_m corresponds to a path in $T[t_m]$ from a node corresponding to some client i' to a node corresponding to some client i'' to the root of $T[t_m]$ with the property that either $i' < i < i''$ or $i'' < i < i'$. In order to guarantee the existence of such a path, the server adds two virtual clients $-\infty$ and $+\infty$ to each new tree.
- There exists a node in $T[t_0]$ which corresponds to i. Moreover, $S(t_0, t) ='' reject''$ for $t_0 < t$ with $t \notin \{t_1, t_2, \ldots, t_j\}$ and $t \leq$ the current time value.

A proof of freshness is verified by using the recursion (2) in order to compute each of the roots associated to the different paths and by using the signatures to verify their correctness. In addition the property regarding the start of each path (either $i' < i < i''$ or $i'' < i < i'$) and the property regarding the scheduling algorithm S are verified. We notice that the paths prove that i is not associated to any of the nodes in the trees $T[t_m]$, $1 \leq m \leq j$; we use the idea of undeniable attestation and the construction of the authenticated search tree attester of [8].

Security: Appendix C presents a sketch of a proof that the resulting freshness scheme is secure if H is a collision resistant hash function and $Sign$ corresponds to a secure signature scheme.

Performance: The main advantage of the log-based scheme is that, for each update, the trusted module only needs to perform one atomic operation. A second advantage is its simplicity. The disadvantage is the amount of communication between the server and a client; in a verification and reservation protocol the main transmission costs are due to the size of the log of sequences in a proof of freshness.

Let N be the total number of clients. Suppose that, as in the tree-based scheme with time multiplexing, each client uses a scheduling algorithm which assigns periodic time values with period q (in practice, clients may use different periods). So, the expected number of client who are scheduled for the same time value equals N/q.

Appendix C presents a precise performance analysis leading to the following result.

Theorem 3 *Suppose that, for each client, u_β is the probability that one of the client's devices requests to start the verification and reservation protocol within the amount of time needed for the transmission of one atomic piece of data (for example, the value of a node in a tree or a signature). Then, the log-based scheme with time-multiplexing can serve up to $1/(2u_\beta \log(u_\gamma/u_\beta))$ clients.*

Notice that the log-based scheme can serve many more clients than the tree-based scheme if $u_\beta \ll u_\gamma$. In the worst-case, the size of a proof of freshness in the log-based scheme is unbounded. In practice this may happen if a client is off-line for a long time. In order to avoid such a worst-case scenario, we may use the adaptive protocol of Section 3.2. In Appendix D we present a second technique; a hybrid scheme with time-multiplexing which allows about as many clients as in the log-based scheme with the advantage that the worst-case size of a proof of freshness is at most $\log(u_\gamma/u_\beta)$ times its expected size.

4.3 Implementation

Our techniques work on any system that can implement a secure clock as we have described it. In designing our secure clock and our freshness schemes, we minimized the requirements on the secure clock as much possible to make it as easy as possible to implement the secure clock's functionality, either as a stand-alone trusted module or as part of a larger general-purpose trusted module.

We have been looking into implementing a secure clock using existing components. In this regard, we have chosen to focus on the Trusted Platform Module (TPM) chip [39, **?**, 48] since it is rapidly growing in availability among end-user devices. We first considered version TPM 1.1 but found that it *cannot* be used to implement a secure clock as we have defined it. Although it is a powerful chip and is useful for certain applications, the main problem is that it has is been designed to trust the owner of the physical machine that it is installed on. Version TPM 1.2, however, offers significant new features. Among these are transport sessions, which can allow a remote user to issue commands with encrypted parameters to the TPM such that an adversary cannot decrypt the command even if he is the owner of the machine and is able to snoop on the bits going in and out of the TPM, and delegation, which enables you to limit the types of commands that users can execute on the TPM.

With these new features, we have developed a solution for implementing a secure clock using a TPM 1.2 chip for a freshness scheme similar to the hybrid scheme of Appendix D. We are in the process of implementing the adaptive protocol. We notice that due to the conditional statements in the atomic operation of the updating algorithm in the tree-based scheme, the tree-based scheme cannot be implemented using TPM 1.2.

5 Concluding Remarks

We introduced a new primitive called freshness scheme. We defined its security and explained multiple constructions based on well known techniques. We showed how freshness schemes can be used in protocols in which an untrusted server with a small trusted module provides trusted storage for a large number of clients, where each client may own and use several different devices that may be offline at different times and may not be able to communicate with each other except through the untrusted server. If the small trusted module is slow compared to the network bandwidth, then a log-based scheme is able to serve many more clients than a tree-based scheme. The simplicity of the log-based scheme is another advantage; it only requires a single secure clock with straightforward read-and-sign and increment-and-sign instructions. In combination with an adaptive protocol, the worst-case communication cost is bounded to a constant size.

A Appendix: Security of the Verifi cation and Reservation Protocol

In this appendix we sketch a proof of Theorem 1. We use induction on t_c with the following induction hypothesis. We assume that part a) of the theorem holds for executions of the protocol if they result in current time values $< t_c$ and we assume that part b) of the theorem holds for executions of the protocol if they result in reservations of times $\leq t_c$. Notice that in the theorem t_n corresponds to a reserved time $> t_c$.

a) Suppose that there is a more recent reservation at time t_r in the range $t < t_r \leq t_c$. Without loss of generality, we assume that t_r corresponds to the first reservation after time t. Since t_r was accepted by one of i's devices d', there exists a reservation certificate r' with $t_r \leftarrow V_R(r', i, d', pk)$ which was generated during a previous execution of the verification and reservation protocol. Since client i accepted the freshness proof p in the current execution of the protocol, $(t, t_c) \leftarrow V_P(p, i, S, pk)$. Since the freshness scheme is secure, the untrusted server cannot succeed in knowing both r and p if $S(t, t_r) ='' accept''$. Therefore, $S(t, t_r) \neq'' accept''$.

During the previous execution of the verification and reservation protocol that generated the reservation certificate t_r, client i verified a confirmation

$$[S', t', Sign(sk_i, H(S'||t'))]$$

of another reservation at some time $t' < t_r$ and verified $S'(t', t_r) ='' accept''$. Since $t_r \leq t_c$, our induction hypothesis states that no times t'_r in the range $t' < t'_r < t_r$ are reserved by i. Since the protocol could have been executed at time $t_r - 1 < t_c$, the server has been able to compute a proof of freshness p' such that $(t', t_r - 1) \leftarrow V_P(p', i, S', pk)$. We remind the reader that t_r corresponds to the first reservation after time t, hence, $t' \leq t < t_r$.

In the current execution of the protocol, client i checks whether t in $(t, t_c) \leftarrow V_P(p, i, S, pk)$ is part of the confirmation

$$[S, t, Sign(sk_i, H(S||t))]$$

and corresponds to a reservation of time t. Since there are no reserved times in the range $t' < t'_r < t_r$, we conclude $t' = t$. By our induction hypothesis there exists only one confirmation for time $t' = t$. Hence, if the signature scheme is secure, then no second confirmation can be forged and we conclude $S = S'$. Hence, $S(t, t_r) = S'(t', t_r) =$ "$accept''$, a contradiction. We conclude that our initial assumption that there exists a more recent reservation at time t_r in the range $t < t_r \leq t_c$ is wrong.

b) Suppose that there is a more recent reservation at time t_r in the range $t_c < t_r < t_n$. Without loss of generality, we assume that t_r corresponds to the first reservation after time t_c. Since t_r was accepted by one of i's devices d', there exists a confirmation $[S', t', Sign(sk_i, H(S'||t'))]$ and there exists a reservation certificate r' such that $t' \leftarrow V_R(r', i, d', pk)$ and $S'(t', t_r) =$ "$accept''$. By the induction hypothesis, t' is i's most recent reserved time value before t_r. In the previous part of the proof we showed that time t was reserved by client i and that there does not exist a reservation at time t'_r in the range $t < t'_r \leq t_c$. Since t_r corresponds to the first reservation after time t_c, we conclude $t = t'$. By our induction hypothesis there exists only one confirmation for time $t' = t$. Hence, if the signature scheme is secure, then no second confirmation can be forged and we conclude $S = S'$. Hence, $S(t, t_r) = S'(t', t_r) =$ "$accept''$. This contradicts t_n being the minimal solution $t_n > t_c$ such that $S(t, t_n) ='' accept''$. This proves that there does not exist a more recent reservation at time t_r in the range $t_c < t_r < t_n$.

Finally, if, besides d, a second device d'' of client i reserved t_n, then the server knows reservation certificates r and r'' with $t_n \leftarrow V_R(r'', i, d', pk)$ and $t_n \leftarrow V_R(r, i, d, pk)$. This contradicts the security of the freshness scheme. Therefore, only device d and no other device of client i reserved t_n.

B Appendix: Analysis of the Tree-Based Scheme

Security: The following is a sketch of a proof that the tree-based scheme is secure if H is a collision resistant hash function and $Sign$ corresponds to a secure signature scheme: 1) Let r and r' be reservation certificates such that $t \leftarrow V_R(r, i, d, pk)$ and $t \leftarrow V_R(r', i, d', pk)$ with $d \neq d'$. Since the signature scheme is assumed to be secure and the update oracle (trusted module) is assumed to only return one signature per time value, the signatures in both certificates are equal to one another. This means that their associated roots must be equal to one another as well. This is only possible if a collision of the hash function has been forged. 2) Let p be a freshness proof and let r be a reservation certificate such that, for some scheduling algorithm S, $(t_m, t_j) \leftarrow V_P(p, i, S, pk)$ and $t_h \leftarrow V_R(r, i, d, pk)$ with $t_m < t_h \leq t_j$ and $S(t_m, t_h) ='' accept''$. Notice that p contains a signature of time t_j which cannot be forged. We conclude that p must correspond to a data structure outputted by the updating oracle, that is, $p = P(D, i)$ with $D \leftarrow U(D', I)$ and known D'. For $p' = P(D', i)$, $(t_m, t_j - 1) \leftarrow V_P(p', i, S, pk)$. By repeating this argument, there exists an older data structure from which a proof of freshness can be deduced which verifies to (t_m, t_h). Since $t_m < t_h$ and the hash function is collision resistant, this proof of freshness contains a signature different from the signature of the reservation certificate. Since both signatures sign the same time value t_h, at most one of them is returned by the updating oracle. Therefore, one of the signatures must have been forged.

Performance: Let N be the total number of clients. Instead of one authenticated search tree, we use q authenticated search trees, each corresponding to N/q clients. This means that a proof of freshness on i only consists of a proof of freshness associated to the tree to which i belongs. This reduces the computation of $U(D, I)$ to $O(1 + |I| \log N/q)$ atomic operations.

A more precise analysis leading to the result presented in Theorem 2 is as follows:

- We define u_i as the probability that one of the devices of client i starts the verification and reservation protocol within one unit of real time. We assume that $u_i = u$ equals the same constant for each client.
- We define γ as the number of time units needed for the computation of one atomic instruction by the trusted module.

Then, the major cost of a single verification and reservation protocol is the computation of an update $U(D, I)$ which costs $(1 + |I| \log N/q) \cdot \gamma$ time units. Let $\mathbf{I}$ be the expected number of clients who requested a reservation of the same time value. Then, the expected time $\mathbf{T}$ between two time values scheduled for the same client is equal to the period q times $(1 + \mathbf{I} \log N/q) \cdot \gamma$:

$$\mathbf{T} = q(1 + \mathbf{I} \log N/q)\gamma. \tag{3}$$

The probability that none of the devices of a client i start the protocol during time $\mathbf{T}$ is equal to $(1-u)^{\mathbf{T}}$. Hence, out of the N/q clients who are scheduled for the same time value only a fraction $(1 - (1-u)^{T})$ have started the verification and reservation protocol:

$$\mathbf{I} = \frac{N}{q}(1 - (1-u)^{\mathbf{T}}). \tag{4}$$

To avoid replay attacks, only one device of a client should use a certain time value to timestamp data. Therefore, if, during the time $\mathbf{T}$ between two time values scheduled for client i, multiple devices of i request to start the protocol, then the protocol tells all devices except for one to "try again later". The remaining devices will be served at the next scheduled time values. During the next time intervals, more devices who request to start the protocol will be disappointed. This cascading effect will force a device to wait for all other devices to finish their version of the protocol. Only if $\mathbf{T}$ is less than the expected time $(1-u)/u$ between two consecutive requests of devices of the same client, this situation is avoided; for example, we require

$$u\mathbf{T} \leq 1/2. \tag{5}$$

Given (5), (4) reduces to $\mathbf{I} \approx Nu\mathbf{T}/q$ and (3) reduces to $\mathbf{T} \approx q\gamma + Nu\mathbf{T}(\log N/q)\gamma$, hence,

$$\mathbf{T} \approx \frac{q\gamma}{1 - Nu\gamma \log N/q} \tag{6}$$

and

$$\frac{\mathrm{d}\mathbf{T}}{\mathrm{d}q} \approx \frac{\gamma(1 - Nu\gamma \log N/q) - Nu\gamma^2/\ln 2}{(1 - Nu\gamma \log N/q)^2}.$$

We conclude that $\mathbf{T}$ is minimized for q defined by $1 - Nu\gamma \log N/q = Nu\gamma/\ln 2$, that is ($\ln e = 1$),

$$q = \frac{Ne}{2^{1/Nu\gamma}}.$$

Substituting this value for q in (6) yields

$$\mathbf{T} \approx \frac{e \ln 2}{u 2^{1/Nu\gamma}}.$$

For this solution for $\mathbf{T}$, requirement (5) is equivalent to

$$N \leq 1/(u\gamma \log(2e \ln 2)) = 0.52/(u\gamma). \tag{7}$$

Notice that $(1 - (1-u)^{\gamma}) \approx u\gamma$ is equal to the probability that one of the devices of a given client requests to start the verification and reservation protocol within the amount of time needed for the computation of one atomic instruction by the trusted module.

C Appendix: Analysis of the Log-Based Scheme

Security: The following is a sketch of a proof that the log-based scheme is secure if H is a collision resistant hash function and $Sign$ corresponds to a secure signature scheme: 1) We can use the proof of the first part for the tree-based scheme to conclude that collisions of reservation certificates cannot be forged. 2) Suppose that p is a freshness proof and r is a reservation certificate such that, for some scheduling algorithm S, $(t_m, t_j) \leftarrow V_P(p, i, S, pk)$ and $t_h \leftarrow V_R(r, i, d, pk)$ with $t_m < t_h \leq t_j$ and $S(t_m, t_h) =''\ accept''$. Since $t_h \leftarrow V_R(r, i, d, pk)$, r contains a signature of the hash of t_h together with the root of a tree with a node corresponding to i. Since $(t_m, t_j) \leftarrow V_P(p, i, S, pk)$, $t_m < t_h \leq t_j$, and $S(t_m, t_h) =''\ accept''$, p contains a signature of the hash of t_h together with the root of a tree for which none of the nodes correspond to i. Since the signature scheme is assumed to be secure and the update oracle (trusted module) is assumed to only return one signature per time value, the two signatures are equal to one another. This means that the signed roots must be equal to one another as well. Since the two trees are different, this is only possible if a collision of the hash function has been forged.

Performance: Let N be the total number of clients. Suppose that, as in the tree-based scheme with time multiplexing, each client uses a scheduling algorithm which assigns periodic time values with period q. So, the expected number of client who are scheduled for the same time value equals N/q.

A precise performance analysis leading to the result presented in Theorem 3 is as follows:

- We define β as the number of time units needed for the transmission of one atomic piece of data (for example, the value of a node in a tree). We assume that the processing speed of the trusted module is slow compared to the network bandwidth, that is, $\beta \ll \gamma$.

The costs of the computation of an update for a given time value in a verification and reservation protocol is a single atomic operation by the trusted module which takes γ time. In (4) we computed the expected number of clients $\mathbf{I}$ who requested a reservation of the same time value (as a function of the expected time $\mathbf{T}$ between two time values scheduled for the same client). Notice that the total number of clients who start a *complete* execution of the verification and reservation protocol equals the total number of clients who reserve a time value. This argument shows that $\mathbf{I}$ also equals the expected number of clients who start a complete execution of the verification and reservation protocol at a given time value. This means that at a given time value we expect the transmission of $\mathbf{I}$ proofs of freshness as well as $\mathbf{I}$ reservation certificates.

Let $\mathbf{L}$ be the expected number of sequences that constitute the log of a freshness proof. Then, the expected size of the log of a freshness proof is equal to $\mathbf{L} \cdot \log \mathbf{I}$ atomic pieces of data which costs $\beta \mathbf{L} \cdot \log \mathbf{I}$ time to transmit. The expected size of a reservation certificate is equal to $\log \mathbf{I}$ atomic pieces of data which costs $\beta \cdot \log \mathbf{I}$ time to transmit. Hence, in the log-based scheme $\mathbf{T}$ is equal to the period q times $\gamma + \mathbf{I}(\beta \mathbf{L} \log \mathbf{I}) + \mathbf{I}(\beta \log \mathbf{I})$:

$$\mathbf{T} = q(\gamma + \beta(\mathbf{LI} + \mathbf{I}) \log \mathbf{I})$$

The probability that, for a given client, h scheduled non-reserved time values occur between two scheduled reserved time values is equal to

$$(1 - (1-u)^{\mathbf{T}})(1-u)^{h\mathbf{T}}(1 - (1-u)^{\mathbf{T}}).$$

Hence,

$$\mathbf{LI} = \frac{N}{q} \sum_{h \geq 0} (1 - (1-u)^{\mathbf{T}})^2 (1-u)^{h\mathbf{T}} h = \frac{N}{q}(1-u)^{\mathbf{T}}. \tag{8}$$

Notice $\mathbf{LI} + \mathbf{I} = N/q$ such that

$$\mathbf{T} = q\gamma + \beta N \log \mathbf{I} \tag{9}$$

For the same reason as in the tree-based scheme we require the bound (5). This reduces (4) to $\mathbf{I} \approx Nu\mathbf{T}/q$ (and reduces (8) to $\mathbf{LI} \approx N/q$, hence, $\mathbf{L} \approx 1/(u\mathbf{T})$ which is consistent with our intuition that $1/(u\mathbf{T})$ is approximately equal to the expected time $(1-u)/u$ between two consecutive requests of devices of the same client divided by $\mathbf{T}$). These approximations combined with (9) yield

$$\mathbf{T} \approx q\gamma + \beta N \log(Nu\mathbf{T}/q) \tag{10}$$

and

$$\frac{\mathrm{d}\mathbf{T}}{\mathrm{d}q} \approx \gamma + \frac{\beta N q}{\ln 2 \mathbf{T}} \left\{ \frac{1}{q}\frac{\mathrm{d}\mathbf{T}}{\mathrm{d}q} - \frac{\mathbf{T}}{q^2} \right\},$$

hence,

$$\frac{\mathrm{d}\mathbf{T}}{\mathrm{d}q} \approx \frac{\gamma - \beta N/(q \ln 2)}{1 - \beta N/(\mathbf{T} \ln 2)}.$$

This shows that, if $\beta N/(T \ln 2) < 1$, then $\mathbf{T}$ is minimized for q defined by $\gamma = \beta N/(q \ln 2)$, that is, $q = \beta N/(\gamma \ln 2)$. Substituting this value for q in (10) yields

$$\mathbf{T} \approx \beta N/\ln 2 + \beta N \log(u\gamma\mathbf{T}(\ln 2)/\beta) = \beta N \log(eu\gamma\mathbf{T}(\ln 2)/\beta). \tag{11}$$

Notice that this solution for $\mathbf{T}$ is an increasing function in N. So, the restriction $\mathbf{T} \leq 1/(2u)$ of (5) is equivalent to the requirement that $N \leq N^*$ for N^* computed as the solution of equation (11) with $\mathbf{T} = 1/(2u)$. That is,

$$N \leq 1/(2u\beta \log(e(\ln 2)\gamma/(2\beta))) = 1/(2u\beta \log(0.94\gamma/\beta). \tag{12}$$

Notice that $(1-(1-u)^\beta) \approx u\beta$ is equal to the probability that one of the devices of a given client requests to start the verification and reservation protocol within the amount of time needed for the transmission of one atomic piece of data. From (7) and (12) we infer that the log-based scheme can serve many more clients than the tree-based scheme if $\beta \ll \gamma$.

D Appendix: Analysis of a Hybrid Scheme

In order to bound the size of freshness proofs to a constant size in the log-based scheme, we propose a hybrid solution between the log-based and tree-based scheme. The idea is to move nodes into the tree-based scheme if they correspond to clients for which the size of a proof of freshness in the log-based scheme gets too large. As soon as a client reserves a new time value, then the client is moved back to the log-based scheme.

A data structure D in the hybrid solution contains both the log-based data structure as well as the tree-based data structure:

$$D = (s[1], T[1], s[2], T[2], \ldots, s[t_c], T[t_c]; t_c, T'),$$

where T' is a balanced authenticated search tree which not only allows modifications but also insertions and deletions and where $(s[1], T[1], s[2], T[2], \ldots, s[t_c], T[t_c]; t_c)$ is exactly the data structure of the log-based scheme in which each signature $s[t] = Sign(sk, H(t||Root[t]||Root'))$ not only signs the root $Root[t]$ of $T[t]$ but also another root $Root'$. In $s[t]$ the value $Root'$ represents the root of T' at the moment of its appearance at time value t. In particular, $Root'$ in $s[t_c]$ is equal to the current value of the root of T'.

Let K be some constant which indicates the worst-case allowable number of atomic pieces of data constituting a single proof of freshness. As in the log-based scheme, the update algorithm $U(D, I)$ increments the current time value and appends to D a new authenticated search tree for I together with a signature of its root. Let t_c be the incremented current time value and let k be the largest integer such that

$$\log N + \sum_{t=t_c-(k-1)}^{t_c} \log \#T[t] \leq K, \tag{13}$$

where $\#T[t]$ indicates the number of nodes in $T[t]$. We delete the nodes in T' which correspond to clients in I (these are the clients who reserve time value t_c) and we insert the nodes of $T[t_c - k]$ which correspond to clients for which no node in each of the k trees $T[t_c]$, $T[t_c - 1]$, $\ldots$, $T[t_c - (k-1)]$ exists (the size of the proofs of freshness for these clients gets too large).

A reservation certificate on i is as in the log-based scheme with an additional proof that no node in T' corresponds to i. A proof of freshness is as in the log-based scheme where the log is truncated up to the first k sequences. If the log is truncated, then a proof of freshness of the tree-based scheme based on T' is included. From the definition of k in (13), we conclude that a proof of freshness contains at most K atomic pieces of data.

Performance We use the hybrid scheme together with time-multiplexing with period q. Due to the law of large numbers, inequality (13) is approximately equivalent to

$$\log(N/q) + k \log \mathbf{I} \le K \tag{14}$$

Let $\mathbf{W}$ be the expected size of a tree T' at any moment. The probability that no device of a given client requests a reservation for at least $k+1$ consecutive scheduled time values is equal to $(1-u)^{\mathbf{T}(k+1)}$. Hence,

$$\mathbf{W} = \frac{N}{q}(1-u)^{\mathbf{T}(k+1)}. \tag{15}$$

If $\mathbf{W}$ is a small constant, then the performance of the hybrid scheme is dominated by the log-based part of the scheme. Up to a small constant this will lead to the bound (12) on the total number of clients N. The expected size $\mathbf{W}$ is small if and only if

$$k \approx \frac{1}{u\mathbf{T}} \log \frac{N}{q}.$$

Notice that $N/q = \gamma \ln 2/\beta$ and $1/(u\mathbf{T}) \approx \mathbf{LI}$ in the log based scheme. So, $\mathbf{W}$ is small if and only if

$$k \approx \mathbf{LI} \log(\gamma/\beta).$$

See (14), a proof of freshness will contain at most

$$\log(N/q) + K \ge k \log \mathbf{I} \approx (1 + \mathbf{LI} \log \mathbf{I}) \log(\gamma/\beta)$$

atomic pieces of data. The expected size of a proof of freshness in the log-based scheme is $1 + \mathbf{LI} \log \mathbf{I}$ atomic pieces of data. We conclude that the hybrid scheme allows as many clients as in the log-based scheme while restricting the worst-case size of a proof of freshness to $\log(\gamma/\beta)$ times the expected size of a proof of freshness:

Theorem 4 *The hybrid scheme with time-multiplexing can serve about as many clients as in the log-based scheme with the advantage that the worst-case size of a proof of freshness is at most $\log(u_\gamma/u_\beta)$ times the expected size of a proof of freshness.*

References

[1] A. Anagnostopoulos, M. Goodrich, and R. Tamassia. Persistent Authenticated Dictionaries and Their Applications. In *4th International Conference on Information Security (ISC)*, 2001.

[2] N. Baric and B. Pfi tzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Advanves of Cryptology: Proceedings of Eurocrypt '97*, pages 480–494, 1997.

[3] D. Bayer, S. Haber, and W. Stornetta. Improving the Effi ciency and Reliability of Digital Time-Stamping. In *Sequences II: Methods in Communication, Security, and Computer Science*, pages 329–334, 1993.

[4] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing effi cient protocols. In *Proceedings of the First Annual Conference on Computer and Communications Security*, 1993.

[5] M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In *Advances in Cryptology - Eurocrypt '96*, pages 399–416, 1996.

[6] J. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Advanves of Cryptology: Proceedings of Eurocrypt '93*, pages 274–285, 1993.

[7] A. Buldas, P. Laud, and H. Lipmaa. Accountable Certifi cate Management using Undeniable Attestations. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 9–17, 2002.

[8] A. Buldas, P. Laud, and H. Lipmaa. Eliminating Counterevidence with Applications to Accountable Certifi cate Management. *Journal of Computer Security*, 10:273–296, 2002.

[9] A. Buldas, P. Laud, H. Lipmaa, and J. Villemson. Time-stamping with binary linking schemes. In *Advances in Cryptology - CRYPTO '98*, pages 486–501, 1998.

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

[11] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Third Symposium on Operating Systems Design and Implementation*, February 1999.

[12] M. Castro and B. Liskov. Proactive recovery in a byzantine-fault-tolerant system. In *Fourth Symposium on Operating Systems Design and Implementation*, October 2000.

[13] D. Clarke, S. Devadas, M. van Dijk, B. Gassend, and G. E. Suh. Incremental Multiset Hash Functions and their Application to Memory Integrity Checking. In *Advances in Cryptology - Asiacrypt 2003 Proceedings*, volume 2894 of *LNCS*. Springer-Verlag, 2003.

[14] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. In *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.

[15] J.-S. Coron and D. Naccache. Security analysis of the gennaro-halevi-rabin signature scheme. In *Advanves of Cryptology: Proceedings of Eurocrypt 2000*, pages 91–101, 2000.

[16] P. T. Devanbu and S. G. Stubblebine. Stack and queue integrity on hostile platforms. *Software Engineering*, 28(1):100–108, 2002.

[17] D. Eastlake and P. Jones. RFC 3174: US secure hashing algorithm 1 (SHA1), Sept. 2001.

[18] S. Even, O. Goldreich, and S. Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, 1996.

[19] B. Gassend, G. E. Suh, D. Clarke, M. van Dijk, and S. Devadas. Caches and Merkle Trees for Effi cient Memory Integrity Verifi cation. In *Proceedings of Ninth International Symposium on High Performance Computer Architecture*, New-York, February 2003. IEEE.

[20] R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In *Advanves of Cryptology: Proceedings of Eurocrypt '99*, pages 123–139, 1999.

[21] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[22] M. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an Authenticated Dictionary with Skip Lists and Commutative Hashing. In *Proceedings of DARPA Information Survivability Conference and Exposition*, pages 68–82, 2001.

[23] M. Goodrich, R. Tamassia, and A. Schwerin. An effi cient dynamic and distributed cryptographic accumulator. In *Proceedings of Inf. Security Conference (ISC)*, pages 372–388, 2002.

[24] S. Haber and W. S. Stornetta. How to Time-Stamp a Digital Document. In *CRYPTO '90: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, pages 437–455, 1991.

[25] M. Kallahala, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable Secure File Sharing on Untrusted Storage. In *Proceedings of the Second Conference on File and Storage Technologies (FAST 2003)*, 2003.

[26] P. Kocher. On certifi cate revocation and validation. In *Proceedings of Financial Cryptography 1998*, pages 172–177, 1998.

[27] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, 1978.

[28] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, 28(9):241–248, 1979.

[29] J. Li, M. Krohn, D. Mazières, and D. Shasha. Secure untrusted data repository (SUNDR). In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, 2004.

[30] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz. Architectural Support for Copy and Tamper Resistant Software. In *Proceedings of the 9^{th} Int'l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pages 168–177, November 2000.

[31] U. Maheshwari, R. Vingralek, and W. Shapiro. How to Build a Trusted Database System on Untrusted Storage. In *Proceedings of OSDI 2000*, 2000.

[32] P. Maniatis. *Historic Integrity in Distributed Systems*. PhD thesis, Stanford University, Aug. 2003.

[33] P. Maniatis and M. Baker. Enabling the Archival Storage of Signed Documents. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST 2002)*, pages 31–45, 2002.

[34] P. Maniatis and M. Baker. Secure History Preservation through Timeline Entanglement. In *Proceedings of the 11th USENIX SEcurity Symposium*, 2002.

[35] D. Mazières and D. Shasha. Don't trust your fi le server. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, May 2001.

[36] D. Mazières and D. Shasha. Building Secure File Systems out of Byzantine Storage. In *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing*, pages 108–117, 2002.

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

[38] S. Micali. Effi cient certifi cate revocation. Technical Report MIT/LCS/TM-542b, 1996.

[39] C. Mitchell, editor. *Trusted Computing*. The Institution of Electrical Engineers, 2005.

[40] M. Naor and K. Nissim. Certifi cate revocation and certifi cate update. In *Proceedings 7th USENIX Security Symposium (San Antonio, Texas)*, 1998.

[41] B. Schneier and J. Kelsey. Cryptographic support for secure logs on untrusted machines. In *The Seventh USENIX Security Symposium Proceedings, USENIX Press*, pages 53–62, January 1998.

[42] B. Schneier and J. Kelsey. Secure Audit Logs to Support Computer Forensics. *ACM Transactions on Information and System Security (TISSEC)*, 2(2):159–176, 1998.

[43] W. Shapiro and R. Vingralek. How to manage persistent state in DRM systems. In *Digital Rights Management Workshop*, pages 176–191, 2001.

[44] S. W. Smith and S. H. Weingart. Building a High-Performance, Programmable Secure Coprocessor. *Computer Networks (Special Issue on Computer Network Security)*, 31(8):831–860, April 1999.

[45] G. E. Suh. *AEGIS: A Single-Chip Secure Processor*. PhD thesis, Massachusetts Institute of Technology, Aug. 2005.

[46] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing. In *Proceedings of the* 17^{th} *Int'l Conference on Supercomputing (MIT-CSAIL-CSG-Memo-474 is an updated version)*, New-York, June 2003. ACM.

[47] G. E. Suh, C. W. O'Donnell, I. Sachdev, and S. Devadas. Design and Implementation of the AEGIS Single-Chip Secure Processor Using Physical Random Functions. In *Proceedings of the* 32^{nd} *Annual International Symposium on Computer Architecture (MIT-CSAIL-CSG-Memo-483 is an updated version available at http://csg.csail.mit.edu/pubs/memos/ Memo-483/Memo-483.pdf)*, New-York, June 2005. ACM.

[48] Trusted Computing Group. Trusted Computing Platform Alliance (TCPA) Main Specifi cation Version 1.1b. https://www.trustedcomputinggroup.org/specs/TPM/TCPA_Main_TCG_Architecture_v1_1b.pdf, 2003.

[49] B. S. Yee. *Using Secure Coprocessors*. PhD thesis, Carnegie Mellon University, 1994.