A Security Analysis of Techniques for Long-Term Integrity Protection*

Matthias Geihs, Denise Demirel, and Johannes Buchmann

Darmstadt University of Technology, Darmstadt, Germany

Abstract. The amount of security critical information that is only available in digital form is increasing constantly. Some of these data, such as medical or tax data, need to be preserved for long periods of time. Thus, several schemes for long-term integrity protection of long-lived and archived data were developed. However, a comprehensive security analysis is still missing. In this paper we discuss existing security models for long lived systems and show to what extend they allow to prove the security of those schemes. Then, we introduce a new model that overcomes the shortcomings of the state of the art and allows to formally analyze timestamp-based long-term integrity schemes. Finally, we show how the security level of the long-term integrity scheme can be determined for concrete instantiations.

1 Introduction

1.1 Motivation

The amount of data that is only available in digital form is increasing constantly. Examples include scientific data, medical records, and land registries. Medical records, for instance, have to be preserved as long as the respective persons are alive and even beyond this time. Thus, the protection period of such data easily may become 100 years. Digital archives are needed that efficiently and securely preserve this information for a long period of time.

Obviously, an important protection goal for archived data is *long-term integrity*. This means that some evidence is needed that data has been archived at a certain point in time and has not been changed since. Various schemes for long-term integrity protection have been developed (e.g., [14, 13, 5, 9, 10, 16, 23]). However, if the security of such schemes is not analyzed, they may be prone to security failures and attacks. For example, an insecure technique for long-term integrity protection was described in [15]. Later, the security issue was noticed and the description was updated [3]. This shows that when developing schemes for long-term integrity protection their security has to be analyzed and proven.

* This work has been published in the proceedings of PST 2016 [11]. It has been cofunded by the DFG as part of project S6 within the CRC 1119 CROSSING and it has received funding from the European Union's Horizon 2020 research and innovation program under Grant Agreement No 644962.

1.2 Contribution

In this paper, we study the security of long-term integrity schemes, focusing on schemes that use timestamps for long-term integrity protection. We start by describing a widely accepted technique for timestamp-based long-term integrity protection and also give a brief overview of other existing methods. Then, we discuss the security of long-term integrity schemes. In particular, we discuss trust assumptions, cryptographic requirements, and the capabilities of adversaries to such schemes.

Finally, we formalize our observations in a novel security model for long-term integrity schemes. In particular, we formally define what a long-term integrity scheme is and which requirements the scheme has to fulfill to be secure. Then, we use this security model to analyze the security of the previously described scheme for timestamp-based long-term integrity protection. We give a reduction-based security proof, showing that the scheme suffices our security definition.

We also evaluate the results of our security analysis, showing which implication our result has for using timestamp-based long-term integrity schemes in practice. In particular, we show how the security level of such a scheme can be determined for concrete instantiations.

1.3 Related work

In [7], Canetti et al. propose a framework for modeling computational security in long-lived systems. Their motivation is that an adversary is usually considered as polynomial-time bounded, but long-lived systems may be running for exponential time. Thus, they propose a framework for such systems where they distinguish between computation time and real time. In their framework, an adversary may be active and interact with the scheme for an unbounded period of time, but has only limited computational power per unit of real time. Using their framework, they study the security of a concrete construction of a timestamp-based long-term integrity scheme. In [12], Geihs et al. use the framework of Canetti et al. to analyze the security of another construction.

However, both papers do not define a notion of security for long-term integrity schemes in general. In comparison, in this work we formalize the notion of security for long-term integrity schemes and provide a comprehensive analysis of their security.

Furthermore, the framework of Canetti et al. does not consider important aspects of long-lived systems. For example, the amount of computational work that can be done per unit of time is fixed. In our novel security model we allow more general types of adversaries. In particular, we consider adversaries whose computational power may increase and which may use different types of computing architectures over time (e.g., classical computers and quantum computers).

In [20], modeling real time for analyzing the security of authenticated key exchange protocols is discussed. Furthermore, the authors propose a model of real time based on a global clock mechanism. We use a similar model of real time in our paper, but in a slightly more general version.

1.4 Organization

The paper is organized as follows. In Section 2, we describe techniques for long-term integrity protection and discuss their security requirements in Section 3. In Section 4, we present a novel security model for long-term integrity schemes in general and analyze the security of timestamp-based long-term integrity schemes. In Section 5, we evaluate the obtained results and look at their implications for practical use of long-term integrity schemes. In Section 6, we draw conclusions and suggest open questions for further research.

2 Long-term integrity protection

In this work, we address long-term integrity protection of data objects (e.g., documents, images, etc.). By long-term integrity we mean that there is a proof that the data object existed at a certain point in time and has not been changed since. In the following, we describe a commonly used technique for generating long-term integrity proofs using timestamps.

2.1 Initiating integrity protection

To ensure integrity of documents it is common practice to use signatures. More precisely, assume a user has a data object d. Then, the integrity of d could, for instance, be ensured as follows. First, the user sends d to a trusted third party, e.g., a certified time-stamping authority (TSA) [1]. The TSA reads the current time t and signs $d\|t_0$, i.e., computes $s_0 = \mathsf{Sign}(d\|t_0)$. Finally, the TSA returns the signature s_0 to the user. This signature together with document d and time t_0 serves as a proof of integrity for d. To verify the integrity of d with respect to time t_0, a verifier simply needs to check that the digital signature s_0 is valid for $d\|t_0$.

The security of this integrity protection relies on the security of the signature. If an adversary is not able to forge the signature of the TSA, it is not able to compromise the integrity of the document. We stress that TSAs must be trusted to always use the correct time in the signing process (for approaches how this trust assumption can be weakened see [22]).

2.2 Prolonging integrity protection

As described in the previous section, the security of the initial integrity protection relies on the security of a single digital signature. Note that digital signature schemes are only secure for a limited time for the parameters chosen [17]. Furthermore, it may happen that the private signing key of the TSA is leaked to an attacker over time. If this happens the attacker can forge signatures, compromising their security. To protect against such security failures, techniques have been developed that allow to prolong the validity of a digital signature using signature chains.

In more detail, the validity of a proof of integrity for d and time t_0 in form of a signature s_0 can be prolonged as follows. The user sends d, t_0, and s_0 to the TSA. The TSA reads the current time t_1 and signs $(d\|t_0\|s_0\|t_1)$, i.e., computes $s_1 = \mathsf{Sign}(d\|t_0\|s_0\|t_1)$. The new proof of integrity consists of d, t_0, s_0, t_1, and s_1 and shows that signature s_0 has been generated before the private signing key used for s_0 got revoked. In other words, the validity of a signature can be prolonged by protecting its integrity from a point in time when it was still valid.

The validity of an integrity proof can be prolonged multiple times. This is done by time-stamping the current integrity proof as described above. Whenever the validity of an integrity proof is prolonged, a new signature is added. Hence, the integrity proof can be described by a sequence of signatures and times, denoted as $t_0, s_0, \ldots, t_n, s_n$. We visualize the procedure in Figure 1.

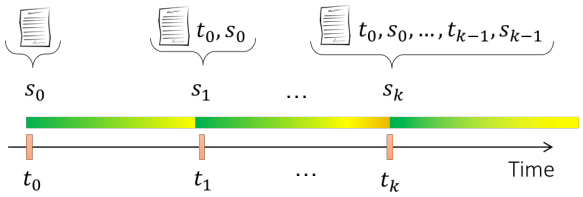

Fig. 1. Generation of a long-term integrity proof. The colored bars visualize the security of the signature schemes.

To verify the integrity of data object d with respect to time t_0 using sequence $t_0, s_0, \ldots, t_n, s_n$, the following needs to be checked. First, it needs to be checked that each signature s_i is a signature on $d\|t_0\|s_0\| \ldots \|t_{i-1}\|s_{i-1}\|t_i$. Second, it needs to be checked that the validity of the signatures was prolonged before the signature keys were revoked, i.e., signature s_i was secure from t_i to t_{i+1} and the latest signature s_n is still secure. Finally, it needs to be checked that s_0 is a signature on d and t_0.

2.3 Other approaches for long-term integrity protection

There are several approaches for long-term integrity protection based on timestamp sequences and notaries. In the following we will briefly summarize the most important techniques. An exhaustive overview can be found in [24].

Timestamps can be generated by using widely visible media (WVM) or digital signature schemes. The basic idea of WVM is that a hash of the data is published on a widely visible medium that has a time reference (e.g., a newspaper). Due to the public release of the hash value and the collision resistance of hash functions, the data cannot be altered afterwards. This approach assumes that the WVM will be available in the future. Furthermore, this approach requires the existence of witnesses, i.e., people that saw the original data and can testify that the data has not been changed since.

An alternative approach to long-term integrity protection uses notaries and digital signature schemes. In notary based schemes a notary generates an attestation by signing the hash of a document together with the current time. When the attestation is refreshed another notary first evaluates the correctness of the current attestation. Then, it generates a new attestation to the document using a fresh signature scheme and replaces the old attestation. Archiving schemes that are based on this idea are, for example, Cumulative Notarization [16] and Attested Certificates [23]. However, for a proof of existence and integrity all these solutions require that the verifier puts trust in the notaries that they verified the last timestamp properly. Since old attestations are deleted a verifier cannot check whether this process has been carried out correctly.

To reduce this trust assumption a sequence of timestamps can be generated and stored as introduced by Bayer et al. [3] and outlined in Section 2.2. This idea has been further developed and several schemes have been published, e.g., Advanced Electronic Signatures [9, 10], Content Integrity Service [14], and the Evidence Record Syntax [13, 5], that basically differ in how the sequence of timestamps to a document is generated and verified. Evidence Record Syntax, for instance, is a format which uses Merkle Trees [18] and allows to timestamp a set of documents. Therefore, this scheme is also efficient for large volumes of data.

3 Security

Several schemes for long-term integrity protection have been developed. For such a scheme to be secure, if a data object d did not exist at some point in time t, then, intuitively, it must be infeasible for anybody to present a proof of integrity for d and t which passes verification. If a proof of integrity scheme has this property, then we say it provides *unforgeability security*. In the following we present the assumptions regarding trust, cryptographic primitives, and attacker capabilities under which the long-term integrity schemes presented here provide unforgeability.

3.1 Trust assumptions

Trusted third parties need to be trusted that they provide a correct reference to real time. For example, TSAs need to be trusted that they read and sign the correct time.

3.2 Cryptographic requirements

For generating proofs of integrity, cryptographic primitives, such as digital signature schemes, are used. Each cryptographic primitive is associated with a validity period, which depends on the scheme and the parameters chosen. We require that the cryptographic primitive is secure within its validity period, e.g., that a signature scheme is unforgeable and that its security does not break unexpectedly.

3.3 Security model and adversary capabilities

To analyze the security of a cryptographic solution a rigorous approach is to formulate a security model and prove the security of the solution within that model. One way to provide long-term security is to require that the scheme is secure against adversaries that have unlimited computational resources. Such a scheme is not prone to brute force attacks, e.g., where an attacker tries to identify the signing key by trying out all possible keys. However, for many cryptographic tasks, such as creating digital signatures, it is hard or even impossible to construct such cryptographic solutions. The good news is that many tasks become achievable if adversaries are modeled as computationally bounded, which is the case in practice.

In a security model, it is common practice to use the Turing Machine Model to model entities which are computationally bounded. Typically, a cryptographic system is associated with an integer security parameter. A computationally bounded adversary is only allowed to perform a number of operations polynomial in the security parameter of the system. For long-lived systems, the Turing Machine based model of computational security is not adequate. The adversary may run for a very long time and therefore it is not appropriate to have a bound on its overall computational power.

In [7], Canetti et al. formulate a security model that considers adversaries which may be active for an unbounded time, but are computationally bounded per unit of real time. This is an important step for modeling computational security in long-lived systems. However, they do not consider other important aspects of long-lived systems, for example, that new computational architectures or new algorithms may become available and the computational bound per unit of time may increase.

In the next section, we present a new framework for modeling computational security in long-lived systems which considers all of the mentioned aspects. Within this framework we consider entities with the following capabilities.

Like in the framework proposed by Canetti et al., adversaries may be active for an unbounded amount of time, and hence, over their indefinite lifetime may perform an unlimited amount of computations. At the same time, they are computationally bounded per unit of time, that is, the amount of computational work that can be done by an adversary per time interval is limited. In addition, we extend this notion by considering that computational power increases, for example, as predicted by Moore's Law. Note that not only the number of operations computable per unit of time may increase, but also new and more efficient algorithms may be discovered. Finally, we also take into account that in the long-term the computational architecture that is available to an adversary may change. For example, an adversary may use quantum computers once they are available. It is conjectured that specific operations can be performed much more efficiently on quantum computers compared to using classical computers [21].

4 Formalization

In this section, we formalize the functionalities of long-term integrity schemes and their security given in Section 2 and Section 3. We start by formally defining long-term integrity schemes and give an example construction TS-LTIS of a long-term integrity scheme based on the scheme described in Section 2. The construction TS-LTIS is timestamp-based and its core functionality is common to many proposed schemes for long-term integrity protection. Afterwards, we define which functionality a long-term integrity scheme must provide in order to provide *correctness* and show that our timestamp-based construction suffices this definition.

Then, we formally define security of long-term integrity schemes. We start by defining the adversary model. Afterwards, we define *unforgeability* for long-term integrity schemes via a game in which the adversary wins if he successfully forges an integrity proof for a document that did not exist at the claimed point in time. Note that in our model we assume that TSAs are trusted to read and sign the correct time and that cryptographic primitives are replaced before they become insecure. We show that our example construction TS-LTIS provides unforgeability security under these assumptions.

4.1 Long-term integrity scheme

Informally, a long-term integrity scheme consists of three algorithms, one for initially generating integrity proofs, one for prolonging the validity of integrity proofs, and one for verifying integrity proofs. The following definition captures this formally.

Definition 1 (Long-term integrity scheme). *A long-term integrity scheme is a tuple of algorithms* (Protect, Prolong, Verify) *with the following properties.*

- $\mathsf{Protect}(d) \to (P, t)$*: On input of data object* d*, the algorithm generates an integrity proof* P*. Additionally, it outputs the time* t *that the proof refers to.*
- $\mathsf{Prolong}(d, P) \to P'$*: When the validity of the existing integrity proof* P *is about to become insecure, this algorithm is called. Then, it generates on input of data object* d *and integrity proof* P *a new integrity proof* P'*.*
- $\mathsf{Verify}(\mathsf{VD}, t_{\mathsf{ver}}, d, t, P) \to b$*: On input of verification data* VD*, current time* t_{ver}*, data object* d*, time* t*, and integrity proof* P*, the algorithm checks the integrity of* d *with respect to time* t*. It outputs* $b = 1$ *if the proof is valid, and* $b = 0$ *otherwise.*

Note that, in contrast to many cryptographic schemes in the literature, a long-term integrity scheme is not associated with a single security parameter $\kappa \in \mathbb{N}$. Typically, this security parameter is used to describe a computational bound on the adversary with respect to a polynomial in κ. However, in a long-lived protocol the computational power of an adversary cannot be considered fixed and cryptographic components may have to be replaced to maintain security.

Hence, imposing a fixed polynomial computational bound on the adversary is not adequate. Instead, we express computational bounds with respect to real time. A more detailed description can be found in Section 4.3.

Next, we give an example construction of a long-term integrity scheme that uses timestamps for generating integrity proofs. It formalizes the description of the long-term integrity scheme from Section 2. The construction is fundamental to many existing schemes, such as [13], [5], or [14].

Construction 1 (Timestamp-based long-term integrity scheme) *The long-term integrity scheme* TS-LTIS *consists of the algorithms* (Protect, Prolong, Verify) *as defined below. The scheme is associated with a constant* $c \in \mathbb{N}$*, which affects the security level as we will see in the security proof.*

- Protect(d)*: The algorithm selects a secure TSA and a hash function* H*. It computes the hash* h *of* d *using* H *and requests a timestamp* τ *on* h *from the TSA. It outputs the timestamp* τ*, which is the initial integrity proof, and reference time* t *included in* τ*.*
- Prolong(d, P)*: When the hash function or the signature scheme used to generate the latest timestamp included in* P *is about to become insecure this algorithm is called. Then, it selects a secure TSA and a hash function* H*. It computes the hash* h *of* (d, P) *using* H *and requests a timestamp* τ *on* h *from the TSA. It outputs the pair* (P, τ)*, which is the new integrity proof.*
- Verify$(\mathsf{VD}, t_{\mathsf{ver}}, d, t, P)$*: The algorithm checks that* P *is of the form* $(\tau_0, \ldots, \tau_n)$ *and that* $n \leq c$*. For* $i = 0, \ldots, n-1$*, it checks that the hash function of* τ_i *and the signatures generated by the TSAs were valid at the time when timestamp* τ_{i+1} *was generated using the verification data* VD*. This verification data consists of the data needed to verify the signature, i.e., the certificate chain binding the public key to the TSA, the revocation list (showing that the signature key pair of the TSA has not been revoked yet), and the trust anchor in form of the root certificates for the certificate chains. Note that this also introduces an additional party, the certification authority (CA) generating the certificates to a private-public key pair of a TSA [6]. Additionally, the algorithm checks that the certificate chain and the hash function of the last timestamp* τ_n *are valid at time* t_{ver}*. For* $i = 0, \ldots, n$*, it computes the hash* h *of* $(d, \tau_1, \ldots, \tau_{i-1})$ *using the hash function of* τ_i *and checks that the signature of* τ_i *is valid for* h*. Finally, it checks that the time of* τ_0 *equals* t*. If all checks succeed, it outputs* 1*. Otherwise, it outputs* 0*.*

4.2 Correctness

Any long-term integrity scheme must fulfill the following correctness requirements. They define what is the intended functionality of a long-term integrity scheme when no adversary is considered and all cryptographic services are assumed to be secure. Afterwards we show that TS-LTIS satisfies this correctness definition.

Definition 2 (Correctness). *Let* $(\mathsf{Protect}, \mathsf{Prolong}, \mathsf{Verify})$ *be a long-term integrity scheme and* n *be an integer. A proof of integrity* P_n *for a data object* d *is generated as follows.*

At time t_0*, the algorithm* Protect *is executed with input* d*. Let integrity proof* P_0 *and reference time* t *be the output of that execution. Then, for* $i = 1, \ldots, n$*, at time* $t_i > t_{i-1}$*, the algorithm* Prolong *is executed with input data object* d *and integrity proof* P_{i-1} *at a point in time when the cryptographic primitives used to generate* P_{i-1} *are still valid. Let integrity proof* P_i *be the output of that execution. Finally, at time* t_{ver}*, the verification algorithm* Verify *is executed. Let* VD *denote the verification data provided by the environment at that time.*

For a long-term integrity scheme $(\mathsf{Protect}, \mathsf{Prolong}, \mathsf{Verify})$ *to be correct we require that for any data object* d *if proofs* $P_1, \ldots, P_n$ *were generated as described above, then the equation* $\mathsf{Verify}(\mathsf{TA}, t_{\mathsf{ver}}; d, t_0, P_n) = 1$ *must hold.*

The next theorem states that our construction TS-LTIS presented in Section 4.1 provides correctness.

Theorem 1 (Correctness of TS-LTIS). *The long-term integrity scheme TS-LTIS is correct.*

Proof. Assume that P_n is an integrity proof generated as follows. First, algorithm Protect is executed at time t_0 with data object d as input returning integrity proof P_0. Then, for $i = 1, \ldots, n$, algorithm Prolong is executed with P_{i-1} as input returning a new proof P_i. For correctness it is required that assuming a series of valid cryptographic services (i.e., TSAs and CAs) is used, then it must hold that $\mathsf{Verify}(\mathsf{VD}, d, t_0, P_n) = 1$, where VD is the verification data. Indeed, proof P_n generated as described above verifies for d and t_0, as can be seen as follows. We observe that the certificate and hash function validity checks in Verify succeed because we assume that the cryptographic services were chosen correctly. Furthermore, we observe that the signature checks during verification also succeed because P_n is constructed by executing algorithm Protect followed by multiple executions of algorithm Prolong, and thus, the data structure of the proof has the correct format.

4.3 Adversary model

In this section, we describe the adversary model for adversaries of long-term integrity schemes. First, we describe the computational model, that is, the computational capabilities of an adversary. Second, we describe the model of real time. Third, we describe how we impose computational bounds on the adversary with respect to real time.

Computational model. We model adversaries as universal computing machines [8]. A *computing machine* models the behavior of any computational device. That is, it has an internal state, and given an input, it performs a series of operations based on its internal program, and after the program is finished,

it outputs a result. A computing machine has a certain set of supported states and operations. For example, a Turing Machine is a computing machine which operates on classical bits using boolean arithmetic. Another example is a Quantum Turing Machine [8] which operates on quantum states using quantum gates. The computational effort performed by a computing machine can be measured based on the number of operations that it performs in order to obtain an output by evaluating its program on a given input.

Real time. In our formal model we use a notion of real time. This allows us to express computational bounds on the adversary with respect to time as discussed in Section 3.3. Various approaches for modeling real time exist [7, 20]. In [20], real time is modeled using a global clock which is incremented whenever the adversary makes a query. This imposes a strict order on any sequence of interactions of the adversary with its environment. Similar to [20], we also use a global clock which is incremented each time the adversary makes a query to advance time. We stress that our model is more general than [20] as the order of events imposed by our clock mechanism is non-strict. For example, this allows the adversary to request and obtain multiple timestamps with the same time reference.

Computational bounds with respect to time. In our model, the computational power of an adversary may be restricted with respect to time in various ways. First, we may restrict the number of operations performed by the adversary per unit of time. For example, an adversary in year 1980 may be capable of performing 1 Gigaflops while an adversary in year 2010 may be capable of performing 1 Petaflops. Second, we may restrict the set of programs known to an adversary at a certain point in time. For example, in year 1980 no polynomial-time deterministic algorithm for primality testing was known, while later such an algorithm was found [2]. Third, we may consider that the computational architecture available to an adversary may change. For example, today computers are only able to perform binary operations. However, in the near future quantum computers may be built that are able to perform operations on qubits. For certain problems, much more efficient algorithms are known for quantum computers compared to classical computers.

4.4 Adversary environment

In Section 4.5 we use a security game to define security of long-term integrity schemes. In such a game the adversary interacts with its environment. For example, it may request timestamps from TSAs or compromise TSAs after their validity period. In the following, we first describe several cryptographic primitives that are used in the context of long-term integrity schemes. Then, we define queries that the adversary can make to interact with cryptographic services in its environment.

Cryptographic primitives. We consider the following cryptographic primitives in the environment.

Digital signature schemes. A digital signature scheme consists of three algorithms KeyGen, Sign, and Verify, and is associated with a security parameter κ [19]. Algorithm KeyGen, on input κ outputs a secret signing key sk and a public verification key pk. Algorithm Sign, on input a secret key sk and a message m, outputs a signature σ. Algorithm Verify, on input a verification key pk, a message m, and a signature σ, outputs 1 if σ is valid for pk and m, and 0 otherwise. A digital signature scheme is considered secure, if it provides existential unforgeability against any polynomial-time adversary [19]. That is, the adversary, given only the public key pk and access to a signing oracle, must not be able to output with non-negligible probability a message-signature pair (m, σ) such that $\mathsf{Verify}(pk, m, \sigma) = 1$ if the signing oracle was not queried with m.

Hash functions. A hash function is a deterministic function mapping input strings of arbitrary length to output strings of fixed length. There are various security properties that can be desirable to be provided by a hash function. For example, a hash function H can be required to be collision resistant. That is, it must be infeasible to find a pair of strings (x, x') such that $\mathsf{H}(x) = \mathsf{H}(x')$. It is common practice to model hash functions as random oracles [4]. A random oracle is associated with a hash value space $\{0,1\}^\lambda$ and a table of assigned hash values T. For any string x, let $T[x]$ denote the entry of table T for string x. On input of a string x, the random oracle checks if x has been queried before and is already in table T. If this is the case, it returns $T[x]$. If not, a random string h is sampled from the hash value space $\{0,1\}^\lambda$. Then, h is inserted in table T at position x, i.e., $T[x] := h$, and returned.

Interaction with the environment. We describe the queries that the adversary uses to interact with the environment.

Set time. As described in Section 4.3, the adversary may only go forward in time. Formally, this is modeled as follows. The environment has a global variable $t_{\mathsf{cur}} \in \mathbb{N}_0$. At initialization, the time is set to zero, i.e., $t_{\mathsf{cur}} := 0$. The adversary may advance time via the following query. Note that because the computational power of the adversary is bounded with respect to time, the adversary is eventually forced to advance time via a $\mathsf{SetTime}$ query in order to continue its operation.

- $\mathsf{SetTime}(t)$: If time t is larger than the current time t_{cur} (i.e., $t > t_{\mathsf{cur}}$), t_{cur} is set to t (i.e., $t_{\mathsf{cur}} := t$).

Request timestamps. In the environment of the adversary TSAs are active. Each TSA is associated with a signature scheme S, security parameter κ, and a validity period, that is, a time interval $[t_1, t_2]$. At the beginning of the validity period, at time t_1, the TSA is initialized as follows. First, it generates a key pair, $S.\mathsf{KeyGen}(\kappa) \to (sk, pk)$. Second, it obtains a public key certificate from a

CA with validity until the end of its validity period (i.e., time t_2). The adversary interacts with TSAs in its environment. In particular, the adversary may request timestamps from them. In addition, after the validity period of a TSA has passed, it can compromise the TSA. In this case, the adversary obtains the private signing key. This is modeled by allowing the adversary to make the following queries.

- $\mathsf{Timestamp}(i, \mathsf{H}, h)$: The adversary makes a timestamp query $\mathsf{Timestamp}$ with input the identifier of a TSA i, a hash function identifier H, and a hash value h of the data object to be timestamped. If the current time t_{cur} lies within the validity period of TSA i, then a signature σ on the current time, the hash function identifier H, and the input hash value h is generated, i.e., $\mathsf{Sign}(t_{\mathsf{cur}}, \mathsf{H}, h) \rightarrow \sigma$. Afterwards, the tuple $(t_{\mathsf{cur}}, \sigma, \mathsf{cert})$ is returned to the adversary, where cert is the public key certificate chain of TSA i.
- $\mathsf{CompromiseTSA}(i)$: The adversary makes a query $\mathsf{CompromiseTSA}$ with input the identifier of a TSA i, to compromise a TSA. Upon such a query, it is checked whether the validity period of TSA i is over, and in this case, the private signing key of TSA i is given to the adversary.

Random oracles. In our security model, the usage of hash functions is modeled by access to random oracles (cf. Section 4.4). A random oracle is associated with a security parameter $\lambda \in \mathbb{N}$, which specifies the length of the output hash values. Additionally, a random oracle has access to a table T that stores previously sampled hash values. Similar to TSAs, a random oracle is also assigned with a validity period $[t'_1, t'_2]$, which determines the validity period of the respective hash function. After this validity period, the adversary may program the oracle to values of its choice. However, as hash functions are deterministic, the adversary may not redefine previously determined hash values. The adversary interacts with random oracles via the following queries.

- $\mathsf{Hash}(j, x)$: If bit string x is not in the table T of random oracle j, i.e., has not been queried before, sample $\{0,1\}^\lambda \rightarrow_\$ h$ and set $T[x] := h$. Return $T[x]$.
- $\mathsf{ProgramHash}(j, s, h)$: If the validity period of random oracle j has passed (i.e., $t_{\mathsf{cur}} > t'_2$), and x is not in the table T of j, set $T[x] := h$.

4.5 Unforgeability

We formalize unforgeability security of long-term integrity schemes. We first describe the security game that an adversary has to win in order to compromise the security of a long-term integrity scheme. The game definition is followed by the unforgeability security definition of long-term integrity schemes. Finally, we prove that the scheme $\mathsf{TS\text{-}LTIS}$ is secure with respect to that definition.

Definition 3 (Unforgeability game). *Let* $\mathsf{LTIS} = (\mathsf{Protect}, \mathsf{Prolong}, \mathsf{Verify})$ *be a long-term integrity scheme,* $\mathcal{A}$ *be an adversary as defined in Section 4.3, and* Env *be an environment as defined in Section 4.4.*

The forgery game is defined as follows. The adversary interacts with its environment and, at some point in time t_{ver}, outputs (d, t, P). Let VD *denote the verification data provided by the environment at time t_{ver}. The adversary wins the game if* $\mathsf{Verify}(\mathsf{VD}, t_{\mathsf{ver}}, d, t, P) = 1$ *and it did not query a hash oracle with d until time t. That is, the adversary is able to output an integrity proof for data object d and time t, without knowing d at time t. We denote the winning probability of $\mathcal{A}$ in this game as* $\mathsf{Adv}^{\mathsf{integrity}}_{\mathsf{LTIS},\mathcal{A},\mathsf{Env}}$.

The game described above is used in the following definition of unforgeable security of long-term integrity schemes. Here, we use the following notation to refer to the security of cryptographic services, such as TSAs or CAs. Let i be a TSA or CA and $\mathcal{B}$ be an adversary. By $\mathsf{Adv}^{\mathsf{signature}}_{i,\mathcal{B}}$ we denote the probability that $\mathcal{B}$ wins the signature unforgeability game for the signature scheme and security parameter used by authority i (cf. Section 4.4). For a random oracle j we denote by $\mathsf{Adv}^{\mathsf{hash}}_{j}$ the probability that an adversary guesses a hash value for a string x without making a query for x. This probability is equal to $2^{-\lambda}$.

Furthermore, we use the following notation with respect to an environment Env. Let $\mathcal{I}$ denote the set of TSAs and CAs of Env. For any point in time t, we denote by $\mathcal{I}_t$ the TSAs and CAs that are available until time t. Likewise, by $\mathcal{H}_t$ we denote the random oracles that are available until time t. Let $\mathcal{B} = \{\mathcal{B}_i\}_{i\in\mathcal{I}}$ be a set of adversaries attacking signature schemes used by the TSAs and CAs of Env. We define $\mathsf{Adv}^{\mathsf{services}}_{\mathcal{B},\mathsf{Env},t}$ as the probability of a security failure of any of the cryptographic services (i.e., signature schemes and random oracles) that have been active until time t,

$$\mathsf{Adv}^{\mathsf{services}}_{\mathcal{B},\mathsf{Env},t} := \left(\sum_{i\in\mathcal{I}_t} \mathsf{Adv}^{\mathsf{signature}}_{i,\mathcal{B}_i}\right) + \left(\sum_{j\in\mathcal{H}_t} \mathsf{Adv}^{\mathsf{hash}}_{j}\right) \quad .$$

Definition 4 (Unforgeability). *Let* $\mathsf{LTIS} = (\mathsf{Protect}, \mathsf{Prolong}, \mathsf{Verify})$ *be a long-term integrity scheme and* Env *be an environment as defined in Section 4.4. We say* LTIS *is unforgeable secure if there exists a constant c such that for any adversary $\mathcal{A}$ as described in Section 4.3, there exist adversaries $\mathcal{B} = \{\mathcal{B}_i\}_{i\in\mathcal{I}}$, such that for any point in time t_{ver},*

$$\mathsf{Adv}^{\mathsf{integrity}}_{\mathsf{LTIS},\mathcal{A},\mathsf{Env},t_{\mathsf{ver}}} \leq c \cdot \mathsf{Adv}^{\mathit{services}}_{\mathcal{B},\mathsf{Env},t_{\mathsf{ver}}} \quad ,$$

where $\mathcal{B}_i$ has computational resources comparable to $\mathcal{A}$ until the end of the validity of service i.

The following theorem states that the long-term integrity scheme TS-LTIS provides unforgeable security.

Theorem 2. *The long-term integrity scheme TS-LTIS is unforgeable secure.*

Proof. We have to show that for every point in time t_{ver}, the probability of $\mathcal{A}$ forging a proof of integrity at time t_{ver} is smaller than the probability that

a similar adversary forges certificates or timestamps, or correctly guesses hash values for the random oracles.

Let $\mathcal{I}$ be the TSAs and CAs, and $\mathcal{H}$ be the random oracles in the environment Env of an adversary $\mathcal{A}$. Furthermore, let (d, t, P) be the output of $\mathcal{A}$ at time t_{ver} such that $\mathsf{Verify}(\mathsf{VD}, d, t, P) = 1$, where VD is the verification data at that time. Write $P = (\tau_0, \ldots, \tau_n)$. We know that the signature of the last timestamp τ_n is a signature that is secure at time t_{ver}, i.e., refers to a certificate chain where each CA is valid at time t_{ver}.

We start with bounding the probability that the adversary $\mathcal{A}$ forges the certificate chain of τ_n. Let $i_1, \ldots, i_m$ be the identities of the CAs of that chain. For each authority i, we construct an adversary $\mathcal{B}_i$ against authority i, as follows. The adversary $\mathcal{B}_i$ runs $\mathcal{A}$, simulating the environment of $\mathcal{A}$. In particular, when $\mathcal{A}$ asks for the signature verification key of authority i, $\mathcal{B}_i$ provides the public verification key of its forgery game, and when $\mathcal{A}$ asks for a certificate of that authority, then $\mathcal{B}_i$ creates that certificate using its signature oracle. When $\mathcal{A}$ makes a query that contains a forged certificate of authority i, then $\mathcal{B}_i$ outputs that forgery. It is obvious that the computational resources required by $\mathcal{B}_i$ are comparable to that of $\mathcal{A}$ until time t_{ver}. We conclude that the probability of the certificate chain being forged is at most $\sum_{j=1}^{m} \mathsf{Adv}^{\mathsf{signature}}_{i_j, \mathcal{B}_{i_j}}$.

Assuming that the certificate chain is genuine, we observe that the probability of the timestamp τ_n being forged is bounded by $\mathsf{Adv}^{\mathsf{signature}}_{i, \mathcal{B}}$, where i is the identity of the corresponding TSA and $\mathcal{B}$ is an adversary constructed similar to the CA adversary described in the previous paragraph.

Let H denote the random oracle of τ_n. We observe that given τ_n is genuine and H is not compromised, the probability that H has not received query $(d, \tau_0, \ldots, \tau_{n-1})$ before time t_n is $\mathsf{Adv}^{\mathsf{hash}}_{\mathsf{H}}$, where t_n is the time assigned to τ_n.

In summary, we obtain that $\mathcal{A}$ did not request a hash value for $(d, \tau_0, \ldots, \tau_{n-1})$ from random oracle H with probability at most

$$\mathsf{Adv}^{\mathsf{hash}}_{\mathsf{H}} + \mathsf{Adv}^{\mathsf{signature}}_{i, \mathcal{B}} + \sum_{j=1}^{m} \mathsf{Adv}^{\mathsf{signature}}_{i_j, \mathcal{B}_{i_j}} \quad .$$

Assuming that the adversary output the data object d and timestamps $\tau_0, \ldots, \tau_{n-1}$ at time t_n, we apply the previous reasoning recursively for $i = n-1, \ldots, 0$.

In summary, we obtain that the probability that the adversary has queried a random oracle with data object d before time t_0 is smaller than

$$\left(\sum_{i \in \mathcal{I}_P} \mathsf{Adv}^{\mathsf{signature}}_{i, \mathcal{B}_i} \right) + \left(\sum_{j \in \mathcal{H}_P} \mathsf{Adv}^{\mathsf{hash}}_{j} \right) \quad ,$$

where $\mathcal{H}_P$ denotes the set of random oracles, $\mathcal{I}_P$ denotes the set of TSAs and CAs which integrity proof P refers to, and for $i \in \mathcal{I}$, $\mathcal{B}_i$ denotes an adversary attacking authority i as described above.

For now, we only considered the case where the adversary outputs an integrity proof that refers to a fixed subset of cryptographic services. If we allow the

adversary to present integrity proofs referring to any sequence of TSAs and random oracles, and take into account that the length of an integrity proof is bounded by the constant parameter c, then we obtain the desired result.

We remark that this reduction-based security proof is substantially different from typical reduction-based security proofs. Typically, the security of a scheme is based on the hardness of a specific computational problem. In contrast, the security of TS-LTIS is based on the security of a set of cryptographic services. Another important difference is that the reduction algorithm is required to be successful within the validity period of the respective service which is in most cases before the long-term adversary has finished its computation.

5 Evaluation

Given the security analysis in Section 4 and the result of Theorem 2, we evaluate the security level L_{int} of the timestamp-based long-term integrity scheme TS-LTIS.

We observe that this value mainly depends on three parameters. The first parameter L_{serv} is the common minimum on the security level of cryptographic services. For example, $L_{\mathsf{serv}} = 80$ means that any service has at least 80 bit security against the considered adversary. The second parameter n corresponds to the total number of services. The third parameter c corresponds to the number of hash values used in the integrity proof.

In Table 1 we show the security level L_{int} of TS-LTIS for different values of L_{serv}, $n = 2^{16}$, and $c = 2^8$. Note that there is a constant factor Δ of security loss between the common security level of the cryptographic services L_{serv} and the security level of the long-term integrity scheme L_{int}.

Table 1. Security of long-term integrity protection as a function of the security of cryptographic services.

n	c	L_{serv}	L_{int}	$\Delta = L_{\mathsf{serv}} - L_{\mathsf{int}}$
2^{16}	2^8	80	56	24
		128	104	
		192	168	

Now we show the dependency between L_{serv} and the number of hash values c contained in the integrity proof. The scheme TS-LTIS only requires one hash value per element of the timestamp chain. However, we remark that some existing schemes for long-term integrity protection use a much higher number of hash values per timestamp. Table 2 shows that for a fixed common security level of the cryptographic services and a fixed number of services, the long-term security level decreases as c increases.

Table 2. Security of long-term integrity protection as a function of the length parameter.

L_{serv}	n	c	L_{int}
128	2^{16}	2^8	104
		2^{16}	96
		2^{32}	80

6 Conclusions

Long-term integrity protection is an important aspect of long-term storage. In this paper, we have studied the security of timestamp-based techniques for long-term integrity protection. We have formalized a notion of security for such schemes and analyzed their security with respect to that notion. This is the first comprehensive security analysis of such schemes, providing a basis for the trustworthy usage of them. We have also shown some practical consequences of our analysis, such as, how the security level of the involved cryptographic services affects the security level of the long-term integrity scheme.

For future work, we plan to provide a security model for long-term integrity schemes similar to ours, but without making use of the random oracle methodology. The random oracle model is widely used, but the criticism remains that random oracles cannot be instantiated in the real world.

Furthermore, it would be interesting to analyze the security of other techniques that promise long-term security. For example, one can chain multiple unconditionally hiding commitments, similar to chaining multiple timestamps, in order to obtain a long-term binding and unconditionally hiding commitment chain. The results from our paper could be used as a basis to formulate a security model for such a construction and analyze its security.

References

1. Adams, C., Cain, P., Pinkas, D., Zuccherato, R.: Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP). RFC 3161 (Proposed Standard) (Aug 2001), updated by RFC 5816
2. Agrawal, M., Kayal, N., Saxena, N.: Primes is in p. Annals of mathematics pp. 781–793 (2004)
3. Bayer, D., Haber, S., Stornetta, W.S.: Improving the efficiency and reliability of digital time-stamping. In: Sequences II: Methods in Communication, Security and Computer Science. pp. 329–334. Springer-Verlag (1993)
4. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993. pp. 62–73 (1993)
5. Blazic, A.J., Saljic, S., Gondrom, T.: Extensible Markup Language Evidence Record Syntax (XMLERS). RFC 6283 (Proposed Standard) (Jul 2011)
6. Buchmann, J.A., Karatsiolis, E.G., Wiesmaier, A.: Introduction to Public Key Infrastructures. Springer (2013)

7. Canetti, R., Cheung, L., Kaynar, D.K., Lynch, N.A., Pereira, O.: Modeling computational security in long-lived systems, version 2. IACR Cryptology ePrint Archive 2008, 492 (2008)
8. Deutsch, D.: Quantum theory, the church-turing principle and the universal quantum computer. Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 400(1818), 97–117 (1985)
9. European Telecommunications Standards Institute (ETSI): TS 101 903 (v1.4.2): XML advanced electronic signatures (XAdES) (2010)
10. European Telecommunications Standards Institute (ETSI): TS 101 733 (v2.2.1): CMS advanced electronic signatures (CAdES) (2013)
11. Geihs, M., Demirel, D., Buchmann, J.: A security analysis of techniques for long-term integrity protection. In: 2016 14th Annual Conference on Privacy, Security and Trust (PST). pp. 449–456 (Dec 2016)
12. Geihs, M., Demirel, D., Buchmann, J.A.: On the security of long-lived archiving systems based on the evidence record syntax. In: Codes, Cryptology, and Information Security - First International Conference, C2SI 2015, Rabat, Morocco, May 26-28, 2015, Proceedings - In Honor of Thierry Berger. pp. 27–44 (2015)
13. Gondrom, T., Brandner, R., Pordesch, U.: Evidence Record Syntax (ERS) (2007)
14. Haber, S.: Content Integrity Service for Long-Term Digital Archives. In: Archiving 2006. pp. 159–164. IS&T, Ottawa, Canada (2006)
15. Haber, S., Stornetta, W.S.: How to time-stamp a digital document. J. Cryptology 3(2), 99–111 (1991)
16. Lekkas, D., Gritzalis, D.: Cumulative notarization for long-term preservation of digital signatures. Computers & Security 23(5), 413–424 (2004)
17. Lenstra, A.K.: Key lengths. In: The Handbook of Information Security. Wiley (2004)
18. Merkle, R.C.: Protocols for public key cryptosystems. In: IEEE Symposium on Security and Privacy. pp. 122–134 (1980)
19. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21(2), 120–126 (1978)
20. Schwenk, J.: Modelling time for authenticated key exchange protocols. In: Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II. pp. 277–294 (2014)
21. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review 41(2), 303–332 (1999)
22. Vigil, M., Demirel, D., Habib, S.M., Hauke, S., Buchmann, J., Mühlhäuser, M.: Lot: a reputation-based trust system for long-term archiving. In: SECURWARE 2016, July 24-28, Nice, France. pp. 1–9 (2016)
23. Vigil, M.A.G., Cabarcas, D., Buchmann, J., Huang, J.: Assessing trust in the long-term protection of documents. In: ISCC. pp. 185–191 (2013)
24. Vigil, M.A.G., Cabarcas, D., Buchmann, J., Huang, J.: Assessing trust in the long-term protection of documents. In: ISCC. pp. 185–191 (2013)