

Implementation of an Evidentiary Record Validation Utility and Security Analysis for Surety's AbsoluteProofSM

Michael Thimblin, NagaSree Chandu Kamisetty, Padmanabhan Raman, Anupama Paila

Abstract—Surety LLC provides digital time-stamping services with AbsoluteProofSM that allow customers to digitally notarize electronic files. Registration provides a customer with the ability to prove that an electronic file was created at a specific time and has not been altered. This service is provided by registering the electronic file fingerprint, a hash value, in Surety's Universal Registry. The integrity of the notarization process requires the need for third-party verification of the entire process. To accomplish this Surety publishes in *The New York Times* a hash value on a weekly basis that is calculated using each record that has been registered. Each registered electronic file has an associated Evidentiary Record that contains the data required to calculate the published value that is associated with the file. An independent utility does not currently exist to perform this verification and we demonstrate that such a utility can be built. In addition we provide a security analysis of the Surety service as its integrity is key to providing a trust-worthy time-stamping solution.

Index Terms—Digital Notarization, Digital Time-stamping, Surety Evidentiary Record

I. INTRODUCTION

TIME-stamping is a process that ensures the integrity of all types of electronic records by establishing the fact that its actual contents existed at a certain point in time. The growth of the Internet has paved the way to using computers for confidential transactions and it would be highly advantageous if one could digitally notarize and validate vital records. Common applications that could make use of digital notarization include banking transactions, patent applications, wills, and research diaries. The use of time-stamping dates back to the 1600's when Galileo published one of his inventions in anagram form and later its translation making way for the evolution of cryptographic time-stamping. Currently the most popular way to notarize a document is to use a Public Notary. These public officials serve as impartial witnesses. Notarizing digital documents has advantages over the paper process. They don't deteriorate while in storage over a long period of time and have a greater flexibility when it comes to storage and distribution. In the online world of today there is an increasing desire to perform most functions in an automated fashion and the notarization of electronic documents is no different. Several companies offer services that provide customers the means to digitally timestamp an electronic file. These services provide customers with means to prove that it existed at a specific point in time.

A notarized digital document should answer two questions; the identity of who created it and what specific time it was created. Digitally signing a document only answers the first and gives no clue about the second question. A major concern with using digital signatures is the management of the private signing keys, which if stolen from its owner can lead to forgery. An important aspect of time-stamping becomes clear when there is a need to verify the integrity of a long-lived document in a legal proceeding. Without electronic time-stamping one can neither trust signed electronic documents nor prove their integrity.

Over the last few years the organizational and legal use of digital signatures and time-stamping has been scrutinized. As such, one will frequently depend on time-stamping to resolve the status of documents. This leads to defining responsibilities of the owner of the signature in addition to the duties and responsibilities of the third party Time Stamping Authority (TSA). Current research in time-stamping techniques is looking to find ways to completely remove trust from the time-stamping authorities making the users ultimately liable for any of their own mistakes.

One of the most primitive protocols to timestamp a document was the Arbitrated Solution or "digital safety-deposit box". The user transmits the document to a Time Stamping Authority (TSA) which concatenates a time and date to the document, stores them in a database, then sending a response. Some of the major contributions for designing new protocols were proposed by Haber and Stornetta [6, 7] of Bellcore labs who proposed the linking and distributed trust scheme. They suggested a much-improved version of the Arbitrated Solution where the TSA receives the document (D) from the user adds time (t) to it and signs it using its private key and sends back $SGN_{TSA}(D||t)$ and t. Their proposal [7, Sect. 5.1, variant 1] also included the using of hash functions to transmit message digest of documents instead of the actual documents to protect alteration of the document, which is the Improved Arbitrated Solution protocol. Due to limitations in the linear linking scheme Ahto Buldas, Peeter Laud, Helger Lipmaa, and Jan Villemson devised a method using binary linking schemes [2]. Surety currently uses the extended binary linking scheme in its time-stamping service and we will discuss this in the further sections.

II. HASH FUNCTIONS

A. Overview

Hashing is a process in which an arbitrary length of data is transformed into a fixed-length result. The original length can vary greatly but the result will always be of the same length, typically 128 or 160 bits in cryptographic use. Hashing is used widely for populating indexes that are used for rapid exact-match searching; the concept is no different in cryptographic use, although the techniques vary greatly. When hashing is used to construct index entries, a balance needs to be struck between the density of entries anticipated in a working system and the likelihood of collisions -- that is, of different entries returning the same hash value. Unless the index is made very large and sparsely filled, there will certainly be collisions, but in indexing these can easily be handled by chaining to empty values and then examining the original entries that share a hash value before returning a result. However, this option is not realistic when hashing is used in cryptosystems and the relevant algorithms need to eliminate collisions. As an example the number of unique 128-bit hash values is fixed at 3.4×10^{38} . However the number of possible messages is infinite and therefore collisions are certainly possible (and will, in fact, be infinite in number). In its practical application the chances of two different messages resulting in the same hash value are vanishingly small in any well-constructed cryptographic hash algorithm and will most likely not occur. [1]

A hash, also called a message digest or finger print, is a one way function used to produce a digital representation that uniquely identifies an electronic document. These one way functions make it impossible to reconstruct the content of the original message and provide a completely different output if the original file is changed in any way. Message digests, being small in size and virtually unique for a given message, are typically used as elements in digital signatures and in digital time-stamping.

Hash algorithms are fundamental to many cryptographic applications. Although widely associated with digital signature technology, the hash algorithms have a range of other uses. SHA-1 and MD5 are two of the most widely known and trusted for use in digital time-stamping.

B. MD5 and SHA-1 Hash Algorithms

Professor Ronald L. Rivest developed the MD5 algorithm in 1994. This algorithm takes as input a message of arbitrary length in 512 bit blocks and produces as output a 128-bit fingerprint or message digest of the input. The MD5 algorithm is an extension of the MD4 message-digest algorithm. MD5 is slightly slower than MD4, but is more conservative in design and is considered more secure. Its 128 bit (16 byte) message digest makes it a faster implementation than SHA-1.

The Secure Hash Algorithm (SHA-1) was developed by the National Institute of Standards and Technology (NIST) in 1994, a division of the US Department of Commerce, which issues standards for cryptographic routines. SHA-1 takes a

message length at most 2^{64} bits and produces a 160-bit "fingerprint" or "message digest" output. In contrast to MD5, SHA-1 is slower in execution and large digest size makes it stronger and more secure against brute force attack and inversion attack.

Research teams in China have published papers on methods for breaking the SHA-1 and MD5 algorithms using a collision search attack. [4] The collision attack involves finding two messages that produce the same hash value. This attack is computationally expensive, but did show a potential weakness of the algorithms. Now that the algorithms have been shown to have a weakness other research teams may quickly refine the process and produce even better results. Despite this fact the hash algorithms can be used in a secure manner using the chain linking scheme.

III. ABSOLUTEPROOFSM

A. Overview

Surety LLC, an IT security software and services company located in Herndon, VA, have a third-party digital-time-stamping service. This service is based on their patented Digital Notary[®] system, AbsoluteProofSM. It provides customers the means to guarantee the specific time the electronic file was created and that its contents have not been modified after that time. This service is provided through the implementation of a cryptographically secure system that is open to independent third party auditing at any time. Through the use of collision-free hash functions a unique representation of an electronic file is stored in a central repository. This repository is a long-lived storage system that currently contains the registration of every record that has been registered since it began running at Bellcore in 1992. [10]

The service is provided through the interaction of several different components. The client application, provided by Surety, is used to generate the hash of the electronic files and submit the records to Surety. The client application also manages the storage of the Notary Record that is returned from Surety. A SDK is available from Surety so that customers can integrate digital notarization and validation into their legacy and custom built applications. The client sends its notarization requests to the Digital Notary Server that timestamps the records. Once processed these records are added to the Universal Registry for permanent storage. The detailed process for notarizing an electronic file and performing a validation of the subsequent Notary Record is in [10].

B. Notarization Process

Here we provide a brief review of the notarization process performed when an electronic file is submitted to Surety (Fig. 1).

- The client application computes the cryptographic hash of the electronic record by computing two separate hashes, MD5 and SHA-1, and then concatenating them together.

- Once the hash has been computed the value is sent to the Surety Digital Notary Server where it is associated with a time-stamp.
- The hash value and its associated time-stamp are then stored in Surety’s Universal Registry.
- The Surety Digital Notary Server creates a Notary Record (document hash and time-stamp) and returns it to the client application.
- The document and its associated Notary Record are stored in the client application’s data store.

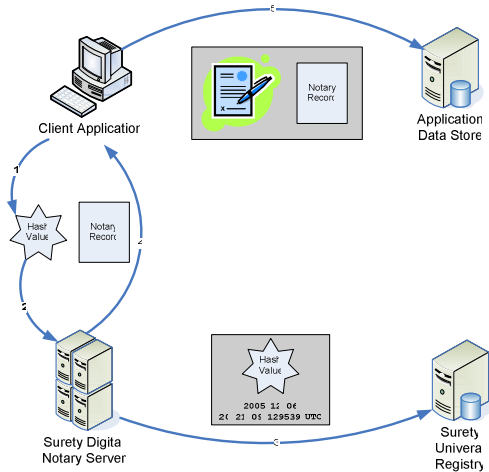


Fig. 1. Surety's Digital Notarization Process

C. Validation Process

Once the document has been stored in the Universal Registry validation of the Notary Record can be performed by the client application at any point in the future. (Fig. 2)

- The electronic file and its associated Notary Record are retrieved from the application’s data store. The hash of the electronic file is computed and compared to the value stored in the Notary Record. If they are the same then the electronic file has not been modified since it was originally notarized.
- The next step is to verify that the Notary Record was registered in Surety’s Universal Registry. Verification is performed by sending the Notary Record to the Surety Digital Notary Server.
- The Digital Notary Server verifies that the hash and time-stamp information match the data stored in the Universal Registry.
- The result of the validation is sent back to the client application.

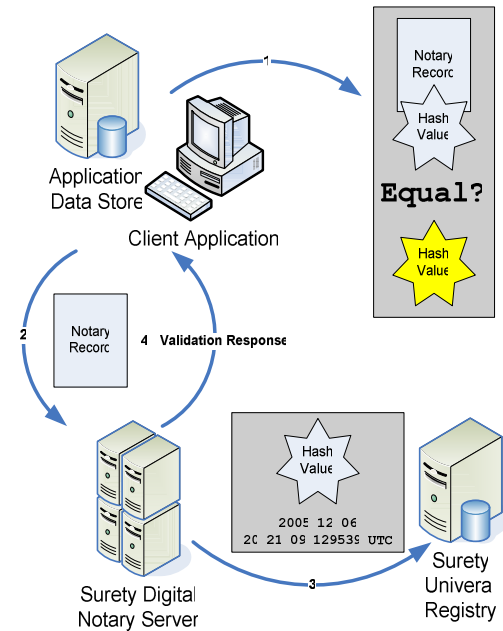


Fig. 2. Surety's Validation Process

D. Universal Registry and Hash Chain Linking

Now that the records are stored in the Universal Registry, the integrity of this data is of great importance. The integrity of the data stored in these records must be kept or else any related Notary Record is of no value in validating the integrity of the original electronic files to which it is related. Surety uses a combination of methods to ensure that the integrity of the registry is kept. The hash records that are entered into the registry are linked using the same hash functions that are used to produce the digest value for the individual electronic files. Not only are the individual records linked, but an aggregate hash value (AHV) is also published weekly in the Public Notices section of *The New York Times* newspaper. [5] This aggregate hash is known as a widely witnessed value. The AHV is calculated every Tuesday night and is the result of each record in the Universal Registry. Publication occurs on the following Sunday. The linking of each record in the registry is the key factor in the process that makes it impossible for anyone to modify any records that exist in the registry.

This linking is also used to combine each individual record that is submitted for entry and is referred to as a hash chain. This linking is accomplished by calculating a new end hash chain value when new records are submitted to the registry. Each link in the chain is known as a Summary Hash Value (SHV) and represents an algorithmically obtained value based upon each previous value in the registry. Upon registration of a new record into the Universal Registry the new hash value is concatenated with the last SHV in the chain and processed in the same manner as the original electronic file. That is two new hash values are created using the MD5 and SHA-1 digest algorithms and the results are concatenated together to produce the new SHV. This new SHV is then added to the end of the hash chain and associated with a specific time-stamp that is used to order the SHV links (Fig. 3).

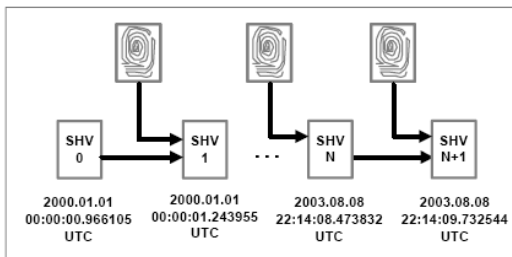


Fig. 3. Surety Hash Chain Linking [10]

During the submission of records to the Universal Registry it is possible that more than one will have the same timestamp. When this occurs these documents need to be linked in some manner before being combined with the latest SHV. Surety performs this task by combining the chronologically equivalent hash records in a binary tree structure known as a Merkle Tree (Fig. 4). In this example the document hash values present in the bottom nodes were submitted with the same timestamp. The paired values are processed via the hash functions in the same manner as the original electronic record. The computed intermediate hash values are then combined in the same manner until the root node is reached. The root node value is then combined with the latest SHV in the hash chain. In actuality the service takes all the hash values submitted within a fixed timeframe for processing. This provides for better system performance and significantly reduces the data storage requirements of the system.

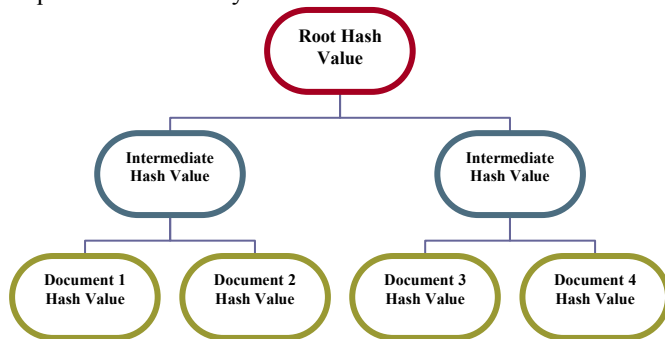


Fig. 4. Merkle Tree Example

The process for calculating the AHV for publication is also performed by creating a Merkle Tree using all the SHV records in the chain. Pairs of the records are combined in the same manner and so on until the final result is obtained. In order for this value to be of use in validation of any specific original electronic file the components of the path used in the calculation need to be known. As mentioned above the hash value for several electronic files may be submitted in the same window during the computation of the SHVs. These values are combined and incorporated into the hash chain. At the appointed time the binary tree of all registered SHVs is created and used to compute the AHV.

E. Surety's Evidentiary Record

The processes described above show that several additional values other than the original electronic file hash are required in the generation of the AHV. The verification process would

be impossible without knowing what values were combined with the original hash value. Once the AHV is published, Surety provides the customer with an XML file known as an Evidentiary Record (ER). This file contains all the data that is needed to compute the AHV from the original hash value. This means that the ER contains all the information required to validate the document and its associated timestamp. Fig. 5 illustrates which components (depicted in yellow) are used in the computation of the SHV and the subsequent published AHV. The fingerprint of the original electronic file is concatenated with the fingerprint of the previous document and processed with the hash algorithms. This value is in turn processed with the hash value created by the other two hash values that were submitted. This produces the Intermediate Hash Value (IHV) that is combined with the SHV+1 to form the new end of the chain (at the time of submission). Creation of the weekly AHV requires the creation of a binary tree using every entry in the chain, and then the computations are performed to obtain the AHV result.

Evidentiary Record Hash Chain Linking

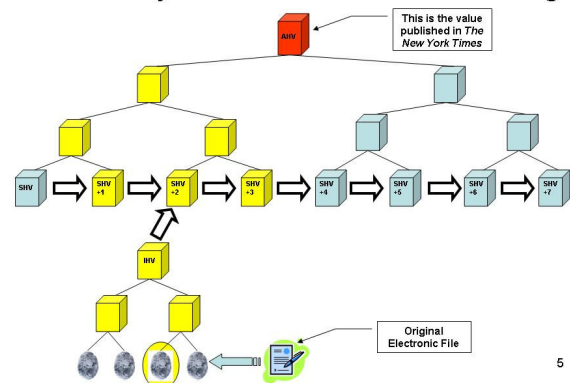


Fig. 5. Evidentiary Record Creation

The first portion of the Evidentiary Record contains the Notary Record that contains the hash algorithms, electronic file digest value, and timestamp.

```
<NotaryRecord>
  <DigestMethodSequence>
    <DigestMethod
      Algorithm="http://www.w3.org/2000/09/xmld
      sig-more#md5"/>
    <DigestMethod
      Algorithm="http://www.w3.org/2000/09/xmld
      sig#sha1"/>
    </DigestMethodSequence>

    <DigestValue>hQ/+mqfAt9HCQjDVQ6ST9+/mpXxo
    Mj4lf1tXv60GRx2dsMau</DigestValue>
    <SureId>01DA523-69F532D-9400030-
    0000000</SureId>
    <Timestamp>2005-11-
    10T22:09:11.559Z</Timestamp>
    <Zone>3</Zone>
  </NotaryRecord>
```

Fig. 6. Notary Record Portion of the ER

The next portion of the Evidentiary Record is the Evidentiary Path that contains the hash algorithms and links

(digest value and position) that are used to compute the AHV. The links are the leaves of the Merkle tree that are used to compute the AHV starting with the digest value of the electronic file.

```

<EvidentiaryPath>
  <DigestMethodSequence>
    <DigestMethod
      Algorithm="http://www.w3.org/2000/09/xmld
      sig-more#md5" />
    <DigestMethod
      Algorithm="http://www.w3.org/2000/09/xmld
      sig#sha1" />
    </DigestMethodSequence>
  <Link>

  <DigestValue>dCzRP9bxCLiIc9gc8l6ey9nGbnpa
  Gcvs4wCWxacmTGNP2G/L</DigestValue>
  <Position>left</Position>
</Link>
<Link>

  <DigestValue>kjG2KRWgSaFYZOQCjZ4CEpkbiBiz
  vjV0js+PCJHzQzM+L9Di</DigestValue>
  <Position>right</Position>
</Link>
<Link>

  <DigestValue>AbmY5GFo4YQnJ1+rdR4CNvTyvZz3
  3LjhX16t774cwLmTqVec</DigestValue>
  <Position>left</Position>
</Link>
.
.
.
  <Link>

  <DigestValue>gfFVKEjPvUGVb0b7SRsF53ht6QhT
  0ElCdJvOcorr4Z4vLcqO</DigestValue>
  <Position>right</Position>
</Link>
</EvidentiaryPath>

```

Fig. 7. Evidentiary Path Portion of the ER

The final section of the Evidentiary Record is the Witness Record that contains the widely witnessed hash value and the information on how and where it was published. The example below shows that the AHV was published in the New York Times on Sunday November 20, 2005.

```

<WitnessRecord>
  <DigestValue>GKV8iuIHUQLn/k0GDy+hd6ogbWWa
  JBgAlY/jg3CjDU+B1ctM</DigestValue>
  <WitnessMethod>
    <Publication>
      <Name>New York Times</Name>
      <Date>2005-11-20</Date>
    </Publication>
  </WitnessMethod>
</WitnessRecord>

```

Fig. 8. Witness Record Portion of the ER

IV. EVIDENTIARY RECORD VALIDATION UTILITY

A. Overview

Surety issues an Evidentiary Records for electronic files that are submitted to AbsoluteProofSM. As shown above this file links the hash value of the electronic file to the widely witnessed value that is published in *The New York Times*.

There is currently no facility available to independently verify the electronic file and its associated Evidentiary Record. As part of this research project we created a utility to perform this verification. The validation performed by the utility not only verifies that the document is the same as the original submission, but also that it existed at the related point in time before the widely witnessed value was published.

B. Applet Details

The utility is a Java Applet that obtains the path to the Evidentiary Record and electronic file from the user. The files are then used to verify if the document was the one associated to the Evidentiary Record specified. The results of the process are displayed on the screen. The applet was built using J2SE version 1.4.2. The MD5 and SHA-1 hash functions, originally released as the Java Cryptography Extension (JCE) are included in the java platform used. The complexity of building this validation utility did not reside with the implementation of the cryptographic algorithms, but in manipulating the byte array results. Care has to be used to ensure that no bits are lost in concatenation otherwise subsequent calculations using the values will produce erroneous results. The security package also provides support for more current hash functions so any updates Surety makes to the digest algorithms used can easily be incorporated into the utility.

The first step is to verify that the electronic file being validated is the same as the one for which the Evidentiary Record is associated. This is performed by calculating the MD5 and SHA-1 hash values for the electronic file. Once calculated the two byte arrays are concatenated, MD5 || SHA-1, and then encoded to BASE64. Surety performs the BASE64 conversion so that the digest values will consist of printable characters for use in XML. Comparison of the calculated value with the digest value in the Notary Record section of the Evidentiary Record will determine if the electronic file is the same as the original submission. If these values are equal validation of the file is successful.

The next step in the verification process is to validate the widely published value that is contained in the Evidentiary Record. In order to perform this validation the AHV must be calculated using the digest value of the document and the digest values for each link in the Evidentiary Path. To perform this step in the process the links in the Evidentiary Path section of the Evidentiary Record are used in the order that they are listed in the XML file. These values need to be decoded from their BASE64 format into byte arrays before being used in the process. To begin, the digest value of the electronic file is concatenated with the digest value of the first link in the Evidentiary Path. The order in which they are concatenated is determined by the position value of the link. A position value of 'RIGHT' indicates that the concatenation will be DOCUMENT DIGEST VALUE then LINK DIGEST VALUE. A value of 'LEFT' indicates LINK DIGEST VALUE then DOCUMENT DIGEST VALUE. The result of the concatenation is stored in a temporary variable. This value

is then hashed using the MD5 and SHA-1 algorithms and then concatenated together (MD5 || SHA-1). This value is stored and used in the next round of processing with the subsequent link digest value following the same process. This continues until all the link values have been used. The final hash value is then encoded using BASE64 and compared to the reported value in the Evidentiary Record. If the calculated value is the same as that reported in the Evidentiary Record then validation was successful and the electronic file is proven to be identical to the original and created at the specified point in time.

Table 1. Complexity of the Validation Utility Code

Utility Class	Lines of Code	Description
ValiationApplet	577	General applet code for the utility interface.
EvidentiaryRecord	69	Reads the XML file and creates a temporary entity to store the data for validation.
NotaryRecord	70	Reads the XML file and stores file specific information such as the digest value and timestamp.
WitnessRecord	158	Reads the XML file and stores the digest values in the ER hash chain.
GenerateHash	134	Performs the hash calculations using the java.security class. This also performs the concatenations specific to the Surety process.

V. SECURITY ANALYSIS

A. Overview

Analyzing the security of time-stamping schemes is important because the documents that are time-stamped are archived and verified at a later time. This requires that the storage of these records are secure against any key compromise, certificate expiry, or weakening of algorithms. Once the documents have been time-stamped, we have to make sure that the time-stamps have not been altered or destroyed from the very time they are stamped. Thus security would be compromised when we issue, renew, validate or store timestamps.

We have tried to identify the various security issues and then sorted out the possible ones, which can exploit an

extended binary linking scheme like Surety. This way we were able to come up with a possible attack scenario.

B. Security and Time-Stamping

The idea of using extended binary linking scheme for time-stamping was not a brainstorm; rather it was the end result of trying out a method and rejecting it over a set of security issues encountered. Based on this we analyze some of the issues of different protocols used for time-stamping.

One of the first protocols available for time-stamping was the Arbitrated Solution discussed earlier that had some serious flaws in its design namely;

- Lack of privacy: Since the TSA receives the original document anyone who intercepts the communication could attempt to modify the contents before sending it to the TSA. Also the adversary can submit the original document independently and get it notarized. This is a form of Replay attack in which a valid data transmission is maliciously or fraudulently repeated or delayed. In this attack the adversary intercepts the data and retransmits it at a later stage. Also the document along with the timestamp would be stored in the TSA database and its security cannot be guaranteed for the future.
- High storage capacity: Files sent to the TSA vary in size from less than few kb to as big as a few gigabytes. Thus storing all files in the database would mean continuously increasing storage requirement for TSA that adds to overall costs.
- Bandwidth Requirements: Sending the large size documents over the Internet would mean the client needs to have high speed and extremely reliable connection.
- Transmission errors: Any errors in the transmission would possibly corrupt the file and junk characters whose notarization would not be accepted later would replace any meaningful pre-existing data.
- Trust issue: The issues of trust remains and it is left to the clients whether they can trust the timestamps issued by a time-stamping authority.

Most of the above-mentioned issues are resolved in the Improved Arbitrated protocol since they use a one-way small hash of a document for transmission thus taking care of lack of privacy and high storage capacity and bandwidth requirement. Also instantaneous response from the TSA enables client to immediately examine the signed time-stamped hash thus any transmission error could be noticed immediately.

But again this protocol does not remove the trust from the time-stamping authority since the TSA uses its private key when signing the concatenation of fingerprint with time thereby an adversary who can break their keys would be able to generate malicious timestamps for clients. These attacks in general are termed as “Backdating/Forward dating” where an adversary would be able to alter the time when it is added to the document. If he timestamps it for an earlier/later dates it would be a Backward/Forward dating attack respectively. The

type of attack an adversary may launch would depend on the purpose of the document. For example if the adversary wants to deal with the Will he will have to use the forward dating attack and in case of a patent he would be benefited he can prove that his document was time-stamped earlier.

A secure hardware based timestamp generator can be still be used to cover its weakness and is currently used by Digistamp Inc. in their application “E-timestamp” which combines hashing techniques with Public key infrastructure PKI. As shown in the figure below the client uses SHA-256 hashing algorithm to create a fingerprint and sends it to the TSA. Digistamp uses a secure timestamp generator to add time and sign the record with its private key. This scheme can definitely answer “who” and “what” of a notarization process but their answer to “WHEN” cannot be accepted since it would depend on how secure their keys are.

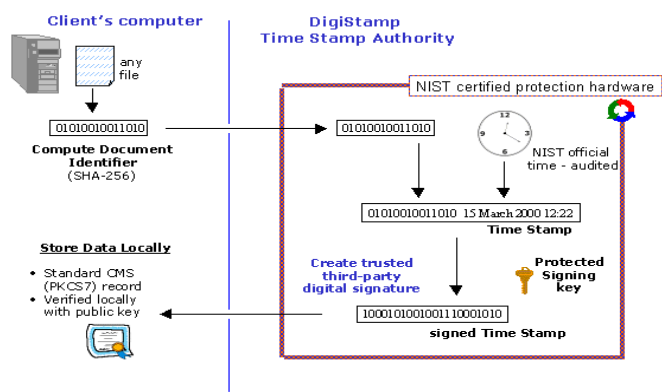


Fig. 9. Digistamp Porcess (<http://www.digistamp.com/timestamp.htm>)

Linking protocol solves this problem by making sure that a timestamp request is linked to the others that came before and after it thus the request or rather the fingerprint sent from the client is concatenated and hashed with his neighbors in the order their requests came in.

These protocols completely eliminate backdating and up to an extent the forward dating on documents. Forward dating attack is still possible by creating a “Fake-chain” which is discussed later. But the ultimate solution to remove trust for implementing time-stamping protocols would be to timestamp without using keys. Now we look into some issues that linking protocol can face if not for the additional features of Surety’s improvised version.

C. Possible Attacks on Surety’s Service

Denial of Service (DoS) attacks has a primary goal, which is denying the victim access to a particular resource. Here the victim could be either the client or the TSA. It is characterized by attempts made by an adversary to inhibit the legitimate users of a service from using or providing that service. One common factor of this D0S attacks is that they all result in service outages. The DoS attacks can be conducted by physical destruction of network components, changing the configuration information, etc.

Time-stamping needs high availability of services since client would have to get document time stamped

instantaneously from the TSA and receive a notary record file.

Surety’s Digital Notary System was designed for high availability. Critical data has been replicated between data centers so each data center can serve as a backup to the other. DN Client applications are configured with a primary server and a secondary server (e.g. using the AbsoluteProofSM Data Integrity Service Deployment as an example, ns1.surety.com and ns2.surety.com are the production notary servers each representing one data center. Likewise, vs1.surety.com and vs2.surety.com are the validation servers). If a client cannot reach the primary, it automatically fails over based on the set connection timeout value. Also a casual enquiry into history of unavailability of service with the employees in Surety reveals that it is a non-occurrence there.

Surety’s Absolute proof generates hash values of 288-bits in length. Finding collisions for such bit lengths is practically infeasible and even if it possible in the near future they cannot alter the validity since a timestamp of a particular document is added to Surety’s hash chain thereby freezing the existence of the contents of the document in time. Hence finding an alternate document that computes to the same hash value has to be done before the original document can be time-stamped, which is not possible since we need the hash value of the original document.

A Man in the middle attack occurs when the adversary gains access to the channel and thus alters the contents of fingerprint transmitted by the client before sending it to the TSA. A client can of course detect the change in the contents of the fingerprint when he tries to validate it. This is just one instance where the security of the time-stamping can be compromised by an insecure line. Surety uses state of the art SSL mechanisms to ensure privacy of communications.

Assumptions used for the theoretical based attacks are listed below.

- All underlying crypto algorithms used are widely published.
- All data transmitted among entities during issuing and verification phases assure confidentiality and integrity, i.e. transmission is over secure channel
- The servers at the TSA are secure.
- Irrespective of the hash function used collision or second pre-image attacks are infeasible to mount.

We now take a look at the possibility of a fake chain attack. Surety’s usage of a Secure Socket layer (SSL) connection prevents an adversary from committing any interception-based attack and so the TSA itself has to carry out a fake chain attack. This attack is possible but can be easily detected if the customer is vigilant enough to validate his time-stamp with the weekly-published Aggregate hash value (AHV). To realize this attack the person should have access to the time-stamping entity and also knowledge of the time-stamp linking mechanism.

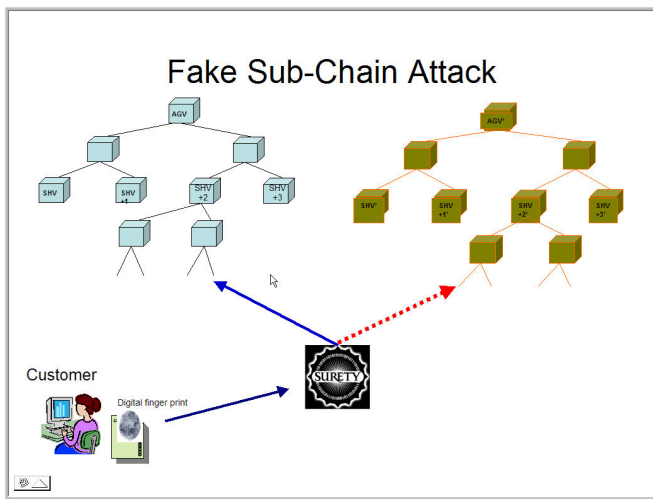


Fig. 10. Fake Hash Chain Attack

Creating a fake set of Summary hash values and other fake fingerprints and linking them completely independent of the TSA's actual hash link chains according to the figure above to construct an entire tree at the TSA can carry out this attack. When a client sends his fingerprint of the document it is added at to a fake fingerprint and hashes with other fake stamps and a fake SHV as shown by the dotted lines in the left side of the figure. The TSA as usual would return a notary record file with .snr extension. When the client tries to immediately validate the document his fingerprint is again added to the fake chain to calculate the SHV and validate. These attacks can be detected within a week's time if and only if the user would take up the task of calculating the hash values in the Merkle tree made from SHV generated over the week thereby it cannot be detected till the final aggregate hash value is made widely witnessed. Hence this attack is a short-lived attack. This can be ruled out by making regular validation.

Another attack we have analyzed is the forward dating attack. This attack is initiated from the customers end. Consider a customer who has time-stamped the hash of his will which initially favors the adversary. After some time the will is updated, writing the adversary out and then is time-stamped again. Assuming that the adversary has access to the hash of the original will and the will owner's login information he can once again re-register the hash of the original will with the Surety service. The TSA time-stamps the hash of the "first" will and it can be used to prove its authenticity. By doing this an adversary can successfully launch such an attack. We do not consider this to be a threat to Surety's service as they cannot be held responsible for securing a customer's information. Surety's responsibility is to preserve the integrity of the registered hash values that can be used to prove that an electronic file was created at a particular time and has not been modified since.

VI. CONCLUSION

The creation of the ER validation utility was successful using only the latest release of Java. By following the publicly available process that Surety uses for registering digital

fingerprint values we were able to validate ER files for documents that we submitted to their service. Another goal of the utility was to determine the complexity of its creation. A programmer with a working knowledge of Java and reading XML files should be able to produce a similar utility without much trouble.

Time-stamping schemes can be made secure only by non-usage of keys and removing complete trust from any third party like a time-stamping authority. Restating our initial statement that security would be compromised when we issue, renew, validate or store timestamps, Surety's Digital notary system seems to be free of any major issues so far. In general approach to designing a much secure time-stamping scheme. Surety has to include a method of conveying the information of summary hash values generated immediately after the client's own summary hash value so that the existence of the hashed document in the chain can be confirmed in less than a week and some suggestions include publishing Aggregate hash values everyday.

ACKNOWLEDGMENT

We would like to thank Surety and specifically Edward Quackenbush for their support in answering questions and providing access to the Surety service. We would also like to thank Dr. Kris Gaj for his invaluable comments in preparing this paper and our presentation.

REFERENCES

- [1] Charlie Kaufman, Radio Perlman, Mike Speciner. Network Security Private Communication in a Public World Edition, Second Edition ©2003
- [2] Ahto Buldas, Peeter Laud, Helger Lipmaa, and Jan Vilemson. Time-stamping with binary linking schemes. In Hugo Krawczyk, editor, Advances on Cryptology — CRYPTO '98, volume 1462 of Lecture Notes in Computer Science, pages 486–501, Santa Barbara, USA, August 1998. Springer-Verlag.
- [3] Bruce Schneier, Applied Cryptography – Protocols, Algorithms, and Source Code in C, 2nd ed., John Wiley & Sons, Inc., New York, 1996
- [4] Bruce Schneier, Schneier on Security – SHA-1 Broken, http://www.schneier.com/blog/archives/2005/02/sha1_broken.html
- [5] Edward Quackenbush, "Surety Technical Presentation", Surety, LLC, 2005.
- [6] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. In A. J. Menezes and S. A. Vanstone, editors, Advances in Cryptology—CRYPTO '90, volume 537 of Lecture Notes in Computer Science, pages 437–455. Springer-Verlag, 1991, 11–15 August 1990.
- [7] Stuart A. Haber and Wakefield Scott Stornetta. How to time-stamp a digital document. Journal of Cryptology, 3(2):99–111, 1991.
- [8] Helger Lipmaa. Secure and Efficient time-stamping systems. PhD dissertation from Department of Mathematics, University of Tartu, Estonia, April 1999.
- [9] The Security Evaluation of Time Stamping Schemes: The Present Situation and Studies IMES Discussion paper series 2001 – E18
- [10] "Ensuring Record Integrity with AbsoluteProofSM", Technical Whitepaper, Surety, LLC, 2003.