Design and Implementation of a RFC3161-Enhanced Time-Stamping Service

Chung-Huang Yang,[1] Chih-Ching Yeh,[2] and Fang-Dar Chu[3]

[1] Institute of Information and Computer Education, National Kaohsiung Normal University, 116, Ho Ping First Road, Kaohsiung 802, TAIWAN, chyang@computer.org
[2] Institute of Information and Computer Education, National Kaohsiung Normal University, 116, Ho Ping First Road, Kaohsiung 802, TAIWAN, dexteryeh@mail3.cy.edu.tw
[3] Telecommunication Laboratories, Chunghua Telecom Co., Ltd., 12 , Lane 551, Min-Tsu Road, Sec.5, Yang-Mei, Taoyuan, 326, TAIWAN, cfonda@cht.com.tw

Abstract. As the Internet grows in scale almost every year, security measures are expected to become all the more important on the Internet. A time-stamping authority (TSA) is a trusted authority which provides a proof that a datum existed before a particular time. In this research, we designed and implemented a RFC3161-compliant trusted time-stamping service (TTS) over the Internet. Software for the TTS server (TSA) was implemented on the Linux platform using Gnu's C compiler while the TSA client software was implemented on the Windows platform using Borland's C++Builder tool. Both TSA server and TTS client is equipped with an IC card reader and an off-the-shelf smart card to strengthen security protection of TTS. For our TTS server, an IC card is used to store private key of the server. On the other hand, IC cards of TTS clients is used to provide user authentication during TTS request and provides TTS server protection against possible denial of service (DoS) attacks that might be deployed by opponents.

1 Introduction

A time-stamping authority (TSA) [1-2] is usually operated as a trusted third party that provides a proof that a datum existed before a particular time. The trusted time-stamping service (TTS) is a crucial part of many secure network applications such as online stock trading, digital notarization service, certificate revocation lists,…, etc. Public-key infrastructures (PKI) [3] are comprised of supporting services that are needed for using public-key technologies on a large scale and TTS is a part of PKI. A public-key certification system works by having a certification authority (CA) for the generation and management (application, storage, renewal, revocation, and inquiry) public-key certificates. In this research, we built a simple root CA whose public-key has already been held in each TTS client and this root CA

issues certificates of TTS servers so that TTS clients could authenticate TTS servers and assure the response contents signed by TTS servers.

As documents or hash values of documents are sent to TTS server, or TSA, for time-stamping, the TSA will response with a time-stamping token in order to indicate that the (hashed) datum existed at a particular point in time. There are many situations where we need to certify the date and time that some data was created or modified. The TSA will link time-stamping token in a tamper-evident chain and it is computationally impossible for anyone to insert a document previously unseen in the middle of the chain [4-5]. Moreover, TTS relies on digital signature to ensure that time-stamping tokens were not tampered and are genuine. Each TTS server would have a pair of keys, a private key and a public key. The private key of TTS server is used to sign the (hashed) submitted request from the TTS client while the public key is used to verify the integrity of digital signature signed by the TTS server. The signature verification process could be performed by any party and cannot be repudiated.

IC cards are much more difficult to duplicate than magnetic strip cards and cryptographic functions can be implemented inside these cards. With substantial cost reductions (at present, price of a reliable IC card reader is less than US$15 and IC card itself is less than US$ 5 per card) and the ability to handle multiple applications on a single card, the IC cards are about to enter a period of the rapid growth. An individual bearing a single IC card will be able to securely interact with several servers or service providers.

In this paper, we present our efforts in the design and implementation of a client/server TTS. Borland C++Builder 6 [6] is used as a software tool to develop an Internet-based TTS client. The open source software provided by the OpenTSA project [7] is used as a basis to build a secure TTS server on the Linux environment. To safeguard private key, our TTS server is equipped with an off-the-shelf IC card with a low-cost IC-card reader and TTS response certificates are signed by private key stored on IC cards. The communication protocol between our TTS clients and TTS server follows the Internet Engineering Task Force (IETF) RFC 3161 standard [8].

2 RFC 3161 Time Stamp Protocol (TSP)

The RFC 3161 time stamp protocol (TSP) [8] defines the format of TTS request and the TTS response. It also establishes several security-relevant requirements for TTS operation, with regards to processing requests to generate response. RFC 3161 protocol is dedicated to the TTS and provides better security functions than the RFC 3029 [9] protocol which describes a general data validation and certification server (DVCS).

There is no mandatory transport mechanism for time stamping message in the RFC 3161, that is, it does not set a rule on how messages are

transmitted between TTS clients and TTS servers. The RFC 3161 TSP merely describes the content of each message sent between the client and the server. TTS clients could have an on-line or off-line mechanism to send request to TTS servers and receive response from TTS servers. In this research, TTS messages are transported via the common HyperText Transfer Protocol (HTTP) and uses a simple browser-server mechanism.

The RFC 3161 protocol provides an effective means to bind time and hashed documents together. A RFC3161 time stamp request contains the hash value of the document, TSP version number, and other optional parameters. As illustrated in Fig. 1, hashed document will be packed and encoded with TSP version number and additional information (such as a nonce to insure the timeliness of the response), then delivers to the TTS server. For RFC 3161, the time stamp request will not identify the requester and this information is not validated by the TTS server.

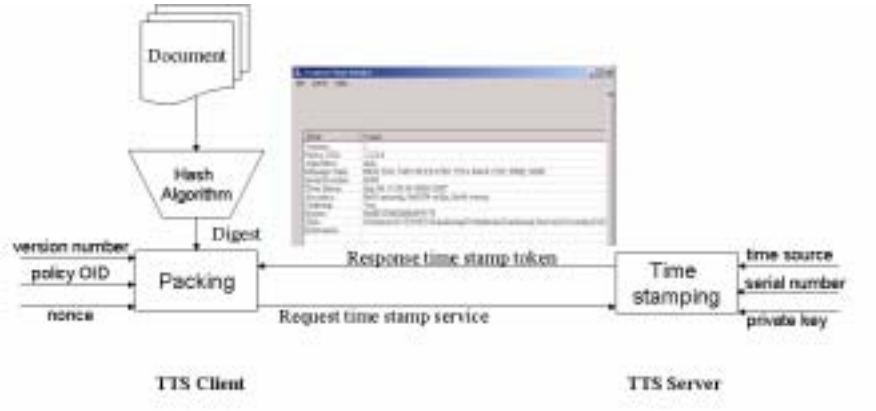

Fig. 1. RFC 3161 time stamping protocol

Upon receiving the request, TTS server will first decide whether or not to grant service to the incoming request. If a time stamp token will be granted, then server will need to prepare a response containing a status, version number, policy, UTC time, accuracy, serial number, message digest, and other optional parameters (such as the name of TTS server). This time stamp response token will be signed using TTS server's private key and the token could be verified again the original document using TTS server's public key.

3 Implementation of a Secure TTS Client

We developed a Windows-based TTS client software which interfaces with TTS server through HTTP protocol over the Internet. The client software,

developed using C++Builder version 6.0 [6] tool from Borland Software Corp. C++Builder is an object-oriented, visual programming environment, it provides tools to develop, test, debug, and deploy applications, including a large library of reusable components, a suite of design tools, application and form templates, and programming wizards.

Our TTS client will create a time-stamping request by selecting a file and choosing a message-digest algorithm (MD5 or SHA-1) to compute document's message digest value, then prepares and sends the time-stamping request via HTTP to the selected TTS server. Upon receiving time-stamping response from the TTS server, our TTS client software will parse and display the response in a meaningful manner. The TTS client software could also save the response and verify it against presentation of the original datum on which the time-stamping request had been sent to TTA.

Our client software could be equipped with an USB-interface IC card reader and an IC card issued by the TTS operator. This enhancement is use to protect operations against the possible denial of service (DoS) attack and our client software will automatically detect the present of IC card with reader and fetch the ID and password information from the card and send to TTS before the TTS request operation. TTS will have the option to decide whether such user authentication is required before providing time stamp service. At present, we use an IC card with the Infineon SLE 4418 memory chip to store TSA's private key. This IC card chip contains 1024 bytes of EEPROM memory and its contents are protected by 4 hexadecimal digits, the programmable security code (PSC).

4 Implementation of a Secure TTS Server

OpenTSA [7] is an open-source and an RFC3161 compliant implementation of TTS server, TSA. It uses OpenSSL [10] for providing basic cryptography functions, such as message encryption and digital signature. We installed and set up both OpenSSL and OpenTSA software on a Linux machine and add with extra C codes to provide additional functions including hardware time source and interface with IC card and reader.

TTS server performs the generation of time-stamping tokens that are digitally signed by the TTS server's private key. In order to protect the private key of TSA, each TSA in our system is equipped with an IC card and reader to store and retrieve private key. The operation of a TTS server will be required to have accurate time source. Our TTS server is equipped with a PCI-interface Time Processor from TrueTime, Inc. This hardware unit is synchronized with national standard time and provides a trusted time source for our TTS server.

Certification authority (CA) fundamentally performs the generation and management (application, storage, renewal, revocation, and inquiry) public-

key certificates that are digitally signed by the CA's private key. With OpenSSL, it is very easy to build a simple root CA that issues public-key certificates, including certificate for TTS servers. Each TTS client software is programmed with the root CA's public key so that it could verify the integrity of TTS server's certificate which is sent with time stamp token by the TTS server. To provide HTTP transport mechanism, we set up the apache web server to listen for time stamp requests from TTS clients, and then pass these request data to the TTS server. Table 1 shows the difference between our TTS server with the original OpenTSA server.

Table 1. Improvements of the implemented TTS server over OpenTSA

System / Feature	TTS server	OpenTSA server
Transport mechanism	on-line via HTTP	off-line
Time source	hardware	software
Protection of private key	IC card	none
User authentication	Optional ID/Password	none
Protection of DoS attack	IC card	none

Table 2. Performance of time-stamping of TTS server on a Linux machine with Intel Celeron 2.40 GHz, signings per second

Algorithm / Key length	SHA1-RSA	MD5-RSA	SHA1-DSA
512 bits	270.0	276.6	298.1
1024 bits	118.7	121.0	197.7
2048 bits	28.4	28.7	91.0

The TTS server provides two choices of one-way hash algorithms, MD5 and SHA-1. It also provides two choices of digital signature algorithms, RSA and DSA. Table 2 gives the performance of our TTS server for different algorithms with 512 to 2048-bit public-key length on a Linux machine with Celeron 2.40 GHz CPU. Since public-key length of 2048 bits are recommended for most secure applications, our TTS is able to provide up to 91 time signings/s. However, without some kind of user authentication, an opponent could easily send more than 100 RFC 3161 requests per second and let our server run out of service. Therefore, our client software has an optional feature to equip with an IC card reader and IC cards issued by our TTS provider to provide a simple user authentication.

5 Conclusion

Public-key infrastructure (PKI) is essential for large-scale secure network applications, where communication confidentiality, authentication, and non-repudiation services are made an integrated part of the systems. PKIs will become more and more prevalent in the near future while time-stamping service (TTS) is a part of PKI. A secure, certifiable, and auditable TTS solution would be useful in many e-government and e-commerce applications. In this paper, we have described our preliminary results on implementing RFC3161-compatible TTS client and server software. We hope to further enhance client's functions to integrate with secure email and/or secure ftp functions on the same software.

Acknowledgement

This research was partially supported by a grant (NSC 93-2213-E-017-001) from the National Science Council of Taiwan.

References

1. S. Haber and W.S. Stornetta, "How to Time-Stamp a Digital Document," *Journal of Cryptology*, Vol. 3, No. 2, 1991, pp. 99-111.
2. S. Haber, B. Kaliski, and W. Stornetta, "How Do Digital Timestamps Support Digital Signatures?," *Cryptobytes*, Vol. 1, No. 3, pp 14-15, RSA Laboratories, Autumn 1995.
3. C. Adams and S. Lloyd, *Understanding Public-Key Infrastructure*, Macmillan Technical Publishing, 1999.
4. Surety.com, "Digital Notary Service Technical Overview," http://www.surety.com/
5. M. Une and T. Matsumoto, "An Evaluation Method of Time Stamping Schemes from Viewpoints of Integrity, Cost and Availability," *IEICE Trans. Fundamentals*, Vol.E86-A, No.1, Jan. 2003, pp.151-164.
6. Borland C++Builder version 6.0, http://www.borland.com/cbuilder/
7. OpenTSA Project, http://www.opentsa.org/
8. C. Adams, et al, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol," *IETF RFC 3161*, August 2001.
9. C. Adams, et al, "Internet X.509 Public Key Infrastructure Data Validation and Certification Server Protocols," *IETF RFC 3029*, February 2001.
10. OpenSSL Project, http://www.openssl.org