Cryptographic Authentication for Real-Time Network Protocols

David L. Mills

Abstract. This paper describes a new security model and authentication scheme for distributed, real-time network protocols used in time synchronization and event scheduling applications. It outlines the design requirements of these protocols and why these requirements cannot be met using conventional cryptography and algorithms. It proposes a new design called *autokey*, which uses a combination of public-key cryptography and a psuedo-random sequence of one-way hash functions. Autokey has been implemented for the Network Time Protocol (NTP), but it can be adapted to other similar protocols. The paper describes the protocol operations, data structures and resources required for autokey, as well as a preliminary vulnerability assessment.

1. Introduction

The Network Time Protocol (NTP) [5] is widely deployed in the Internet to synchronize computer time to national standards. The current NTP population includes over 230 primary servers and well over 100,000 secondary servers and clients. It provides comprehensive mechanisms to access national time and frequency dissemination services, organize the hierarchical network server-client topology and adjust the clock of each participant. It uses redundant servers, diverse network paths and crafted algorithms which cast out incorrect servers and minimize errors due to network latencies and clock frequency variations. The protocol can operate in peer-peer, client-server, and multicast modes, where the identity of the servers can be cryptographically authenticated. In most places of the Internet of today, NTP provides accuracies of 1-50 ms, depending on the characteristics of the synchronization source and network paths.

Sponsored by: DARPA Information Technology Office Contract DABT 63-95-C-0046, NSF Division of Network and Communications Research and Infrastructure Grant NCR 93-01002 and Northeastern Center for Electrical Engineering Education Contract A303 276-93.

1991 *Mathematics Subject Classification*. Primary 94A60; Secondary 68P25.

Most NTP servers and clients use the NTP Version 3 reference implementation for Unix, VMS and Windows, which is a relatively complex, real-time, distributed application. The architecture, protocol and algorithms are specified in RFC-1305 [6]. The NTP Version 4 specification is not yet complete, but transition documents are available [7] which describe the new features. The NTP Version 4 reference implementation now under test supports most of these features, including the authentication scheme described in this paper. Additional information can be found at the NTP home page http://www.eecis.udel.edu/~ntp and the author's home page http://www.eecis.udel.edu/~mills.

The NTP security model is common to many other ubiquitous, distributed applications, such as directory services, web servers and archive repositories. Conventional authentication models are based on public-key cryptography and digital signatures; however, the known public-key algorithms drastically degrade timekeeping accuracy and require significant computing resource commitments. Current key-agreement schemes based on Diffie-Hellman [11] do not scale well in a large network with thousands of servers and clients. Furthermore, a time-sensitive protocol such as NTP places severe requirements on latency and available resources which cannot be satisfied using conventional approaches. With NTP, cryptographic media lifetimes and time synchronization interact in complicated ways that can affect the security of all authenticated network services.

This paper presents a new security model and authentication scheme for NTP and similar real-time protocols. It begins with an overview of the NTP protocol operations, in order to provide a context for the remainder of the paper. Following this is a description of the new authentication scheme, called *autokey*, which provides public-key based authentication of servers where the cryptographic keys are randomly generated and used only once. The paper concludes with a discussion of possible weakness in the scheme and an assessment of its vulnerability to specific attacks.

2. NTP Version 3 Security Model and Authentication Scheme

The existing NTP security model and authentication scheme described in the NTP Version 3 specification RFC-1305 were developed some years ago in response to what were then considered likely directions in the evolution of generic Internet security models and authentication schemes. Since then, there have been significant developments in the field of cryptographic algorithms and security models. In order to understand how the current authentication scheme works and how it has evolved to the autokey scheme, it is necessary to understand the various modes of operation and how previously unknown servers are discovered.

It is important to point out at the outset that the NTP security model is intended only to verify that a server is in fact authentic and not an intruder attempting accidently or on purpose to impersonate a legitimate server. It is not necessary, nor would it be politically expedient, to encrypt the timestamps or otherwise hide the data in NTP messages, since these are public values. It is not the intent in the model to include access controls; other mechanisms based on network address and port filtering are available for that. It is not necessarily the case that the model includes protections from message loss, duplication or corruption, since these are provided by the NTP protocol itself.

A NTP client normally operates with several servers, in order to provide redundancy and insure correctness. It discovers these servers by indexing directory services or by intercepting multicast messages they send. The state variables for each server belong to a client process, or association, which can operate in various modes using point-to-point and point-to-multipoint communications paradigms. In peer-peer and client-server modes, a client sends a request to a designated server address and expects a reply from which it can determine the relative clock offset and roundtrip delay. Synchronization flow is strictly from server to client in client-server modes, but can flow either way in peer-peer modes. In multicast mode, a server sends a message to a designated multicast group address. Multicast clients normally listen on the designated address and send no requests. Some multicast servers can also respond to client-server requests, so that clients can calibrate the network propagation delay before continuing operation in listen-only mode.

The NTP security model recognizes that individual servers can fail or operate incorrectly or even attempt to disrupt other servers and clients in one way or another. In addition, network links can fail, routes can change or become congested, and cryptographic keys and even security policies can change while in regular, continuous operation. However, at any one time, synchronization flows from the primary time servers at the root of the NTP tree graph or *subnet* to all secondary servers and clients at increasing stratum levels toward the leaves. The model requires the authentication tree to follow the same subnet paths. Thus, if the primary servers are correctly synchronized and authenticated (by outside means) and, if each secondary server and client is synchronized and authenticated to each server at the next lower stratum level, the subnet is considered secure.

The authentication scheme described in RFC-1305 is designed to support this security model. This scheme has since been augmented to include provisions for the MD5 message digest algorithm [12], in addition to the DES-CBC [8], [9] algorithm. The scheme contains provisions to cryptographically authenticate individual servers operating in any mode using a symmetric-key cryptosystem and private keys.

In RFC-1305, an association is distinguished by source and destination network addresses and assigned a secret cryptographic key, which is stored in a secure database. The key is accessed by a unique key identifier, which serves a purpose similar to the security parameter identifier (SPI) described in recent internet drafts [2], [3]. The key is used to construct a message digest (one-way hash function) of the message using either keyed MD5 [4] or DES-CBC. The key identifier and message digest together form the message authentication code (MAC), which is transmitted following the NTP packet header. The recipient uses the key identifier included in the MAC to retrieve the secret key from its own secure database and verifies the authenticity of the message by computing the message digest and comparing it with the corresponding value included in the MAC.

The authentication function itself is reasonably fast, even with thousands of clients and in servers providing other functions, such as file sharing, name resolution and security services. The busiest Internet servers have well over 750 clients producing an average (input plus output) of over ten packets per second. For a Sun IPC workstation, which is slow by today's standards, the average time to service a non-authenticated packet is about 1.5 ms; authentication adds about 0.28 ms to this figure. However, even the busiest servers spend not more than about 1.6 percent of the available processing resources for all NTP operations.

It is possible to engineer some interesting and useful security topologies by sharing a single key among a set of servers and clients. For example, a closely cooperating clique of primary servers operating in peer-peer modes can share a single key, in order to provide backup for each other if a radio clock source fails. This avoids having to distribute a different key to every server in the clique. In another example, a set of servers can operate in multicast mode with a single key, so that a client population can synchronize to any of them without requiring separate keys for each one.

3. Autokey Authentication Scheme

While the current NTP security model and authentication scheme have been in use for well over a decade, there are several drawbacks, the most serious being the requirement that keys must be securely distributed in advance for all server-client pairs. There are no provisions in the NTP protocol specification for key distribution or management on the assumption these functions can be provided by standard network security services. Even if such services were available, the large number of associations, well over 250,000 in the current NTP subnet, would make the operations to securely manufacture and distribute keys and enforce their lifetimes very difficult to sustain. In addition, the recent addition of multicast modes raises new issues where the identity of servers is not known in advance and their credentials must be determined using only public values.

The autokey scheme described in following sections is specifically tailored to the needs of time-sensitive protocols where the keys are randomly generated and used only once, as in the S/KEY scheme [1]. It involves three stages of operation for each potential server. In the first stage, the server network address is determined, either directly from directory services or from an intercepted multicast message, and an association mobilized to communicate with the server. In the second stage, time synchronization and server authentication functions proceed independently.

As each function proceeds, potential servers that fail the authentication test are discarded and keys that fail the lifetime test are discarded. The third stage begins when the time-sensitive server credentials are verified and the server joins the population used to discipline the system clock. Subsequently, the authenticity of each NTP message is verified using mechanisms described in the next section.

3.1 Conventional Approaches

In a perfect world with inexhaustible processor and memory resources, a public-key cryptosystem such as RSA public-key infrastructure (PKI) [10] would be a good foundation on which to build the authentication scheme. In a typical scheme such as RSA or El Gamal digital signatures, each server or clique of servers is assigned a public/private key pair along with public values such as the algorithm modulus, server name, and network addresses. The private key is held by the server and never divulged. The public values are stored in directory service databases, together with one or more digitally signed certificates which bind the public values to a trusted agent serving as a notary.

The power of public-key cryptography lies in the fact that a message encrypted with a given private key can be correctly decrypted only using the corresponding public values. In order to minimize the vulnerability to cryptographic attack, every message must be individually signed using the server private key. In order to minimize the processor requirements, the usual practice is to construct the message digest using a one-way hash function such as MD5, then encrypt it using RSA and the server private key. The result is stored in the MAC and transmitted with the message.

To verify the signature, the client decrypts the MAC using the server public values, then compares the result with its own message hash. If the two values agree, the message must be authentic, since only the designated server has the corresponding private key. In practice, the public values must be cryptographically bound to the server name and address, as verified by a trusted certificate authority. A certificate including these values is then installed in a public directory service such as the DNS.

Constructing the MD5 message digest is a relatively fast operation involving only standard arithmetic and logical functions. For instance, the time to construct the NTP message digest on a Sun SPARC 20 is 54 us and 291 us on a SPARC IPC. However, even when the encrypted data are a relatively small 16-octet MD5 hash, the RSA encryption operation is expensive. For instance, the processor time to generate a NTP MAC using MD5 and RSA ranges from 80.4 ms on a Digital Alpha to 2.1 s on a Sun SPARC 1.

In principle, the encryption times can be measured in advance and used to correct time values. However, there is a large variance in running times, depending on the population of one bits in the key and other factors. For example, with random bit strings as keys, the Alpha requires a mean time of 80.4 ms; however the actual times range from 53.3 ms to 104.4 ms. Since time values must be obtained before encryption, these variations translate directly to timekeeping errors. While there are other schemes based on public-key cryptography, all are based on computation-intense algorithms and are likely to behave in a way similar to RSA. For these reasons, the autokey scheme does not require each message to be individually signed.

An alternative to authenticating each message is a key distribution scheme, in which the server generates a session key and transmits it to the client using public-key cryptography, or a key agreement scheme, in which the server and client jointly agree on a joint session key using a variant of Diffie-Hellman. Either way requires servers to hold a distinct key for each client and for clients to hold a distinct key for each server. Either of these schemes requires the server and client to maintain persistent state variables, but this may not be possible for a server with hundreds or thousands of clients. For this reason, the autokey scheme must not depend on pairwise private keys.

3.2 Interactions Between Synchronization and Key Lifetimes

A basic rule in all key distribution and management schemes is that cryptographic key and certificate media must have enforced lifetimes. Specific keys must be destroyed and replaced from time to time, in order to frustrate potential cryptanalysis. New keys must not work with old data and old keys must never be used again. Obviously, reliable lifetime enforcement requires reliable time synchronization. If secure timekeeping is dependent on lifetime-enforced cryptographic media, an interesting circularity results. In principle, this circularity can be resolved through the use of special timekeeping hardware present in some drop-in security devices, such as the Fortezza card; however, this hardware is politically unacceptable in some contexts and does not work for all computers manufactured today.

If trusted hardware is not available, the autokey scheme treats clock synchronization and server authentication as separate functions. These considerations require that these functions may have to operate when reliable network timekeeping has not yet been established or when the keys have not yet been certificated. The most common case occurs when a client is first started after reboot or when the server configuration is changed. In these cases, the key distribution and management functions must operate even when key lifetimes are not enforced and before the local clock has been reliably set. Thus, any protocols used by NTP itself to initiate cryptographic associations must not depend on prior key exchanges that are themselves dependent on synchronized clocks.

A public-key cryptosystem requires reliable directory services to obtain the server public values, including the server name, network address, public key, modulus and optional certificates. In principle, these services are required to be synchronized to trusted sources only if they support encryption or decryption operations, since these operations require keys with enforced lifetimes. Presumably, the availability and authenticity of the public values depend on databases accessible via inband or outband mechanisms; however, the ultimate decision on whether the data are authentic rests with the clients of these services, not the server itself.

In general, server public values are obtained during the initial configuration process when the client is first started. This is done for each server separately while at the same time provisional time values are collected. Since reliable encryption and decryption operations cannot be done unless the clock is synchronized and the lifetimes verified, this requires the client to fetch all cryptographic values first, then perform the decryptions on a tentative basis. When sufficient time values have accumulated to reliably synchronize the clock, the lifetimes can be checked against the tentative time. If all checks agree, the server is considered authentic. Thereafter, the authenticity of its messages must be determined by other means as described below.

3.3 Cryptographic Operations

Certain extensions to the existing NTP security model and authentication scheme are required for autokey to support additional cryptographic data in the packet header and to enforce key lifetime. The packet format has been revised to include an optional extension field, which is inserted between the end of the NTP header and the beginning of the MAC. The cryptographic message digest is constructed using all the data between the beginning of the NTP header to the beginning of the MAC, including the extension field, if present.

A cryptographic key consists of a 4-octet key identifier, a variable length key, a key type indicator (DES or MD5), a validity indicator, and a remaining lifetime counter. In the NTP reference implementation, these values are stored in a key cache, with the most recently used key saved in a special location for quick access. A miss on the key identifier causes the new key to replace the old key, which is stored in the key cache. The key cache itself uses a hash table for quick lookup. Routines are provided to create an entry, mark it valid and specify its lifetime. The validity indicator provides a mechanism to create a block of keys in advance and enable them as a block at a given time.

In order to preserve backwards compatibility, there are two ranges of key identifier values. Private keys have identifiers less than 65,536 and are assigned specific values with indefinite lifetimes. Random keys have identifiers greater than 65,536 and are assigned random values or hashes of a random values with designated lifetimes. During normal operation, the remaining lifetime of each random key is decremented once each second. When this value decrements to zero, the entry is automatically expunged from the cache. While the association of key identifier and key is arbitrary in this design, an important consequence is that each key identifier must be unique and never replicated in the cache during its lifetime.

3.4 Protocol Operations

As each autokey server is registered with directory services, it generates a RSA public/private key pair and constructs a certificate trail which clients can use to verify the authenticity of the public values. Registering the server can be a lengthy process involving interactions with many other services, so this is done only on rare occasions during the server lifetime. Each server maintains a private random value used as a master key-generating key. This value is refreshed at intervals of about one day using a good random generator, which is a relatively expensive operation taking up to several seconds on modern workstations. The random keys in the key cache must be expunged when the RSA key pair or master key are changed.

Clients may operate several simultaneous server associations, each identified by a unique source and destination address. At intervals on the order of a thousand seconds, the server generates a 4-octet random seed using a fast algorithm and computes the 16-octet MD5 hash of this value concatenated with the current master key. This is the first entry in a pseudo-random sequence or key list of 16-octet session keys The least significant four octets of the hash is used as the key identifier for the second entry, which is constructed as the MD5 hash of the concatenated source and destination network addresses and the key identifier of the first entry. Note that the autokey scheme in effect includes all significant fields of the NTP message, not just the NTP header as in the original scheme, and thus provides additional security.

Continuing in this way, the server completes the key list, which may have from a few to several hundred entries, depending on the interval between master key regenerations and the interval between NTP messages. Finally, the server encrypts the last session key using its RSA private key, and saves the result for later.

The server uses the key list in inverse order; that is, the last entry is used first and is assigned sequence number zero, then the next before that, which is assigned sequence number one, and so on until all entries have been used. At this point, the server generates a new random seed and computes a new key list as before. Each time the server uses an entry, it stores the low order four octets of the next session key (not yet used) in the key identifier field of the packet and the sequence number and encrypted last session key in the extension field. Formulated in this way, the server does not need to recalculate session keys as they are needed and can expunge them immediately after use.

A client authenticates each message relative to the message that immediately precedes it. It computes the session key and message digest as described above, then extracts the low order four octets of the session key and compares them with the key identifier of the previous message, which has been saved for this purpose. If the values agree, the current message is considered valid. If not, a message might have been discarded in transit, so the client hashes again. This procedure may continue for a maximum number of hashes equal to the current sequence number. To complete the procedure, the client decrypts the last session key, which is included in the extension field, and verifies it matches the final hash.

4. Security Vulnerabilities

In the NTP security model, the first line of defense is access control based on an address value bounds check. This is followed by the authentication scheme, and a set of sanity checks which deflect old duplicates or messages with format errors or data range errors. The crafted NTP filtering, selection, clustering and combining algorithms are designed to distinguish between *truechimers* that are correctly synchronized to UTC, and *falsetickers* that are not. The bottom line is that an attack succeeds if the intruder can fool all of these algorithms into accepting a real falseticker as a truechimer or rejecting a real truechimer as a falseticker.

An intruder can try to break the various cryptographic keys used by the autokey scheme, including the RSA private key, master key, random seed used to generate the session key list, or an individual session key. For all practical purposes, the RSA private key and the master key are cryptographically unassailable. However, the attacker may choose as target an individual session key. Since the source and destination network addresses are known, only the 4-octet key identifier for a session key not yet used needs to be known in order to predict the remaining session keys on the list.

The attacker might start by intercepting a legitimate message, then try to guess a key identifier that results after one or more hashes to match the key identifier in the message. This might not be as hard as it seems. The work function for a successful attack is $O(2^{31})$ MD5 hashes. If each hash takes one microsecond, which might be possible with future hardware, the intruder succeeds on average after only 2000 seconds, so the scheme would be classed as cryptographically weak. While it would be simple to extend the key length by changing the packet header format, this would create an incompatibility with older versions of the protocol. The difficulties this would create outweigh the advantages of a longer key.

Depending on the network configuration and location of the intruder, it may be able to intercept, delay and retransmit messages (replay attack), modify these messages (modification attack), or prevent onward transmission and attack in either of these ways (middleman attack), or even clog the network, server or client with spurious traffic (clogging attack).

A middleman can intercept, delay and replay selected messages from the client to the server or from the server to the client. These messages will be properly authenticated so will be believed correct. The middleman can delay the message or pass on only certain messages while discarding all others. The effect would be to introduce a statistical bias in the timekeeping data or to cause the client to believe the server is unstable. Such attacks may be hard to detect, since much the same thing happens as the result of normal network behavior. There is no intrinsic protection against this attack, other than the NTP mitigation algorithms, which rely on the inherent redundancy and diversity of the synchronization subnet to resist attack.

The session key applies only to the current message and is not useful for any subsequent message. However, an middleman could intercept a legitimate server message and learn the current session key before clients have received the message. The intruder is then free to manufacture a bogus message which will be accepted by these clients. There appears to be no defense against this attack other than using public-key cryptographic signatures on every message.

In a clogging attack, the intruder attempts to deny service by overwhelming the resources of the client, server or network by generating large volumes of traffic, either replay or bogus. Clogging attacks to not require the intruder to pry open legitimate NTP messages, just the capability to clone a valid NTP message. By its very nature, NTP operating in client-server or multicast modes is designed as a ubiquitous service; that is, a server will ordinarily respond to any client request without necessarily authenticating the request or checking to see if it contains valid data. As the processor cycles necessary to reply to a client request are only modest and no additional state persists after the reply has been generated, the vulnerability of a server to a clogging attack is minimal. However, an intruder can manufacture an otherwise correct NTP message, but substitute a bogus key identifier with correct session key and MAC. The victim can discover the message is bogus, but only after repeated hashes and a significant resource investment to validate the RSA signature.

In the old authentication scheme, an intruder can disable a properly authenticated source by a contrived jamming attack. It can simply send a stream of bogus messages to the victim, which will cause it to expunge all time values, legitimate or not. This lesson having been learned, in the autokey scheme bogus messages are discarded before any damage can result.

5. Summary and Conclusions

The design of a robust security model and authentication scheme for a time-sensitive protocol such as NTP is not a straightforward application of current cryptographic technology. The scheme must be scalable, non-invasive of processor resources and continue support for legacy protocol versions. It must allow server authentication and time synchronization to proceed independently to avoid circular dependencies that might lead to deadlock. The autokey scheme described in this paper has been devised to comply with these requirements. It uses a combination of public-key cryptography with one-way hash algorithms where keys are randomly generated and used only once.

The alpha distribution of the reference implementation for NTP Version 4, which includes the autokey scheme, has been in regular operation for several months. While it includes all the protocol mechanisms described in this paper, the mechanisms to retrieve public values from secure directory services have yet to be implemented. It is anticipated that these mechanisms will be incorporated in the form of application libraries made available from other sources.

6. References

Note: Internet Drafts are perishable memoranda describing works in progress and may change in substantial ways before final publication as research reports. They are cited here only when no other source of the material is available.

1. Haller, N. The S/KEY one-time password system. Network Working Group Report RFC-1760. Bellcore, February 1995, 12 pp.

2. Karn, P., and W.A. Simpson. The Photuris session key management protocol. Network Working Group Internet Draft, Qualcomm, November 1995, 66 pp.

3. Maughan, D., M. Schertler. Internet security association and key management protocol (ISAKMP). Internet Draft, National Security Agency, November 1995, 59 pp.

4. Metzger, P., and W. Simpson. IP authentication using keyed MD5. Network Working Group Paper RFC-1828, Piermont and Daydreamer, August 1995, 5 pp.

5. Mills, D.L. Internet time synchronization: the Network Time Protocol. IEEE Trans. Communications COM-39, 10 (October 1991), 1482-1493.

6. Mills, D.L. Network Time Protocol (Version 3) specification, implementation and analysis. Network Working Group Paper RFC-1305, University of Delaware, March 1992, 113 pp.

7. Mills, D.L, and A. Thyagarajan. Network time protocol version 4 proposed changes. Electrical Engineering Department Paper 94-10-2, University of Delaware, October 1994, 32 pp.

8. DES modes of operation. FIPS Publication 81, National Bureau of Standards, December 1980.

9. Data encryption standard. FIPS Publication 46-1, National Bureau of Standards, January 1988.

10. PKCS #1: RSA encryption standard, Version 1.5. RSA Laboratories, November 1993.

11. PKCS #3: Diffie-Hellman key-agreement standard, version 1.4. RSA Laboratories, November 1993.

12. Rivest, R. The MD5 message-digest algorithm. Network Working Group Paper RFC-1321, MIT and RSA, April 1992, 21 pp.

ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT, UNIVERSITY OF DELAWARE, NEWARK, DE 19716