Internet Engineering Task Force David L. Mills
Internet Draft University of Delaware
Standards Track February 2002

Public key Cryptography for the Network Time Protocol
Version 2
< draft-ietf-stime-ntpauth-03.txt >

Status of this Memorandum

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [1].

1. Abstract

This memorandum describes a scheme for authenticating servers to clients using the Network Time Protocol. It extends prior schemes based on symmetric key cryptography to a new scheme based on public key cryptography. The new scheme, called Autokey, is based on the premiss that the IPSEC schemes proposed by the IETF cannot be adopted intact, since that would preclude stateless servers and severely compromise timekeeping accuracy. In addition, the IPSEC model presumes authenticated timestamps are always available; however, cryptographically verified timestamps require interaction between the timekeeping function and authentication function in ways not yet

considered in the IPSEC model.

The main body of this memorandum contains a description of the security model, approach rationale, protocol design and vulnerability analysis.

It obsoletes a previous report [11] primarily in the schemes for distributing public keys and related values. A detailed description of the protocol states, events and transition functions is included. Detailed packet formats and field descriptions are given in the appendix. A prototype of the Autokey design based on this memorandum has been implemented, tested and documented in the NTP Version 4 software distribution for Unix, Windows and VMS at www.ntp.org.

While not strictly a security function, the Autokey protocol also provides means to securely retrieve a table of historic leap seconds necessary to convert ordinary civil time (UTC) to atomic time (TAI) where needed. The tables can be retrieved either directly from national time servers operated by NIST or indirectly through NTP and intervening servers.

Changes Since the Preceding Draft

This is a major rewrite of the previous draft. There are numerous changes scattered through this memorandum to clarify the presentation and add a few new features. Among the most important:

1. The reference implementation now uses the OpenSSL cryptographic software library. Besides being somewhat faster than the older RSAref2.0 library, it supports several different message digest and signature encryption schemes.

2. The Autokey protocol and reference implementation support the Public Key Infrastructure (PKI), including X.500 certificates.

3. The Autokey protocol has been redesigned to be simpler, more uniform and more robust. There is only one generic message format and all requests can carry signed parameters.

4. Strong assertions are now possible about the authentication of timestamps and filestamps. This makes correctness modeling more robust and simplifies vulnerability assessment.

5. Certain security potholes have been filled in, in particular the cookie in client/server and symmetric modes is now encrypted.

6. The description of the protocol, its state variables, transition function, inputs and outputs are simpler, less wordy and more amenable to correctness modelling.

7. Provisions have been made to handle cases when the endpoint addresses are changed, as in mobile IP.

Introduction

A distributed network service requires reliable, ubiquitous and survivable provisions to prevent accidental or malicious attacks on the servers and clients in the network or the values they exchange. Reliability requires that clients can determine that received packets

are authentic; that is, were actually sent by the intended server and not manufactured or modified by an intruder. Ubiquity requires that any client can verify the authenticity of any server using only public information. Survivability requires protection from faulty implementations, improper operation and possibly malicious clogging and replay attacks with or without data modification. These requirements are especially stringent with widely distributed network services, since damage due to failures can propagate quickly throughout the network, devastating archives, routing databases and monitoring systems and even bring down major portions of the network.

The Network Time Protocol (NTP) contains provisions to cryptographically authenticate individual servers as described in the most recent protocol specification RFC-1305 [7]; however, that specification does not provide a scheme for the distribution of cryptographic keys, nor does it provide for the retrieval of cryptographic media that reliably bind the server identification credentials with the associated private keys and related public values. However, conventional key agreement and digital signatures with large client populations can cause significant performance degradations, especially in time critical applications such as NTP. In addition, there are problems unique to NTP in the interaction between the authentication and synchronization functions, since each requires the other.

This memorandum describes a cryptographically sound and efficient methodology for use in NTP and similar distributed protocols. As demonstrated in the reports and briefings cited in the references at the end of this memorandum, there is a place for PKI and related schemes, but none of these schemes alone satisfies the requirements of the NTP security model. The various key agreement schemes [2, 5, 12] proposed by the IETF require per-association state variables, which contradicts the principles of the remote procedure call (RPC) paradigm in which servers keep no state for a possibly large client population. An evaluation of the PKI model and algorithms as implemented in the RSAref2.0 package formerly distributed by RSA Laboratories leads to the conclusion that any scheme requiring every NTP packet to carry a PKI digital signature would result in unacceptably poor timekeeping performance.

A revised security model and authentication scheme called Autokey was proposed in earlier reports [5, 6, 8]. It has been evolved and refined since then and implemented in NTP Version 4 for Unix, Windows and VMS [11]. It is based on a combination of PKI and a pseudo-random sequence generated by repeated hashes of a cryptographic value involving both public and private components. This scheme has been tested and evaluated in a local environment and in the CAIRN experiment network funded by DARPA. A detailed description of the security model, design principles and implementation experience is presented in this memorandum. Additional information about NTP, including executive summaries, software documentation, briefings and bibliography can be found at www.eecis.udel.edu/~mills/ntp.htm. Additional information about the reference implementation can be found at www.eecis.udel.edu/~ntp/ntp_spool/html/authopt.htm.

Security Model

NTP security requirements are even more stringent than most other distributed services. First, the operation of the authentication mechanism and the time synchronization mechanism are inextricably intertwined. Reliable time synchronization requires cryptographic keys which are valid only over designated time intervals; but, time intervals can be enforced only when participating servers and clients are reliably synchronized to UTC. Second, the NTP subnet is hierarchical by nature, so time and trust flow from the primary servers at the root through secondary servers to the clients at the leaves.

A client can claim authentic to dependent applications only if all servers on the path to the primary servers are bone-fide authentic. In order to emphasize this requirement, in this memorandum the notion of "authentic" is replaced by "proventic", a noun new to English and derived from provenance, as in the provenance of a painting. Having abused the language this far, the suffixes fixable to the various noun and verb derivatives of authentic will be adopted for proventic as well.

In NTP each server authenticates the next lower stratum servers and proventicates the lowest stratum (primary) servers. Serious computer linguists would correctly interpret the proventic relation as the transitive closure of the authentic relation.

It is important to note that the notion of proventic does not necessarily imply the time is correct. A client considers a server proventic if it can validate its certificate and its apparent time is within the valid interval specified on the certificate. The statement "the client is synchronized to proventic sources" means that the system clock has been set using the time values of one or more proventic client associations and according to the NTP mitigation algorithms. While a certificate authority must satisfy this requirement when signing a certificate request, the certificate itself can be stored in public directories and retrieved over unsecured networks.

Over the last several years the IETF has defined and evolved the IPSEC infrastructure for privacy protection and source authentication in the Internet, The infrastructure includes the Encapsulating Security Payload (ESP) [4] and Authentication Header (AH) [3] for IPv4 and IPv6. Cryptographic algorithms that use these headers for various purposes include those developed for the PKI, including MD5 message digests, RSA digital signatures and several variations of Diffie-Hellman key agreements. The fundamental assumption in the security model is that packets transmitted over the Internet can be intercepted by other than the intended receiver, remanufactured in various ways and replayed in whole or part. These packets can cause the client to believe or produce incorrect information, cause protocol operations to fail, interrupt network service or consume precious processor resources.

In the case of NTP, the assumed goal of the intruder is to inject false time values, disrupt the protocol or clog the network or servers or clients with spurious packets that exhaust resources and deny service to legitimate applications. The mission of the algorithms and protocols

described in this memorandum is to detect and discard spurious packets sent by other than the intended sender or sent by the intended sender, but modified or replayed by an intruder. The cryptographic means of the reference implementation are based on the OpenSSL cryptographic software library available at www.openssl.org, but other libraries with equivalent functionality could be used as well. It is important for distribution and export purposes that the way in which these algorithms are used precludes encryption of any data other than incidental to the construction of digital signatures.

There are a number of defense mechanisms already built in the NTP architecture, protocol and algorithms. The fundamental timestamp exchange scheme is inherently resistant to replay attacks. The engineered clock filter, selection and clustering algorithms are designed to defend against evil cliques of Byzantine traitors. While not necessarily designed to defeat determined intruders, these algorithms and accompanying sanity checks have functioned well over the years to deflect improperly operating but presumably friendly scenarios. However, these mechanisms do not securely identify and authenticate servers to clients. Without specific further protection, an intruder can inject any or all of the following mischiefs. Further discussion on the assumed intruder model is given in [9], but beyond the scope of this memorandum.

1. An intruder can intercept and archive packets forever, as well as all the public values ever generated and transmitted over the net.

2. An intruder can generate packets faster than the server or client can process them, especially if they require expensive cryptographic computations.

3. An intruder can originate, intercept, modify and replay a packet. However, it cannot permanently prevent packet transmission over the net; that is, it cannot break the wire, only tell lies and congest it. In this memorandum a distinction is made between a middleman attack, where the intruder can modify and replace an intercepted packet, and a wiretap attack, where the intruder can modify and replay the packet only after the original packet has been received.

The following assumptions are fundamental to the Autokey design. They are discussed at some length in the briefing slides and links at www.eecis.udel.edu/~mills/ntp.htm and will not be further elaborated in this memorandum.

1. The running times for public key algorithms are relatively long and highly variable. In general, the performance of the synchronization function is badly degraded if these algorithms must be used for every NTP packet.

2. In some modes of operation it is not feasible for a server to retain state variables for every client. It is however feasible to regenerated them for a client upon arrival of a packet from that client.

3. The lifetime of cryptographic values must be enforced, which requires a reliable system clock. However, the sources that synchronize the system clock must be cryptographically proventicated. This circular interdependence of the timekeeping and proventication functions requires special handling.

4. All proventication functions must involve only public values transmitted over the net. Private values must never be disclosed beyond the machine on which they were created.

5. Public encryption keys and certificates must be retrievable directly from servers without requiring secured channels; however, the fundamental security of identification credentials and public values bound to those credentials must be a function of external certificate authorities and/or webs of trust.

Unlike the ssh security model, where the client must be securely identified to the server, in NTP the server must be securely identified to the client. In ssh each different interface address can be bound to a different name, as returned by a reverse-DNS query. In this design separate public/private key pairs may be required for each interface address with a distinct name. A perceived advantage of this design is that the security compartment can be different for each interface. This allows a firewall, for instance, to require some interfaces to proventicate the client and others not.

However, the NTP security model specifically assumes all time values and cryptographic values are public, so there is no need to associate each interface with different cryptographic values. In other words, there is one set of private secrets for the host, not one for each interface. In the NTP design the host name, as returned by the gethostname() Unix library function, represents all interface addresses. Since at least in some host configurations the host name may not be identifiable in a DNS query, the name must be either configured in advance or obtained directly from the server using the Autokey protocol.

Approach

The Autokey protocol described in this memorandum is designed to meet the following objectives. Again, in-depth discussions on these objectives is in the web briefings and will not be elaborated in this memorandum. Note that here and elsewhere in this memorandum mention of broadcast mode means multicast mode as well, with exceptions noted in the web page at www.eecis.udel.edu/~ntp/ntp_spool/html/assoc.htm.

1. It must interoperate with the existing NTP architecture model and protocol design. In particular, it must support the symmetric key scheme described in RFC-1305. As a practical matter, the reference implementation must use the same internal key management system, including the use of 32-bit key IDs and existing mechanisms to store,

activate and revoke keys.

2. It must provide for the independent collection of cryptographic values and time values. A client is synchronized to a proventic source only when the required cryptographic values have been obtained and verified and the NTP timestamps have passed all sanity checks.

3. It must not significantly degrade the potential accuracy of the NTP synchronization algorithms. In particular, it must not make unreasonable demands on the network or host processor and memory resources.

4. It must be resistant to cryptographic attacks, specifically those identified in the security model above. In particular, it must be tolerant of operational or implementation variances, such as packet loss or misorder, or suboptimal configurations.

5. It must build on a widely available suite of cryptographic algorithms, yet be independent of the particular choice. In particular, it must not require data encryption other than incidental to signature encryption and cookie encryption operations.

6. It must function in all the modes supported by NTP, including client/server, broadcast and symmetric modes.

7. It must not require intricate per-client or per-server configuration other than the availability of the required cryptographic keys and certificates.

8. The reference implementation must contain provisions to generate cryptographic key files specific to each client and server. Eventually, it must contain provisions to validate public values using certificate authorities and/or webs of trust.

Autokey Proventication Scheme

Autokey public key cryptography is based on the PKI algorithms commonly used in the Secure Shell and Secure Sockets Layer applications. As in these applications Autokey uses keyed message digests to detect packet modification, digital signatures to verify the source and public key algorithms to encrypt session keys or cookies. What makes Autokey cryptography unique is the way in which these algorithms are used to deflect intruder attacks while maintaining the integrity and accuracy of the time synchronization function.

The NTP Version 3 symmetric key cryptography uses keyed-MD5 message digests with a 128-bit private key and 32-bit key ID. In order to retain backward compatibility, the key ID space is partitioned in two subspaces at a pivot point of 65536. Symmetric key IDs have values less than the pivot and indefinite lifetime. Autokey key IDs have pseudo-random values equal to or greater than the pivot and are expunged immediately after use.

There are three Autokey protocol variants corresponding to each of the three NTP modes: client/server, broadcast and symmetric. All three variants make use of specially contrived session keys, called autokeys,

and a precomputed pseudo-random sequence of autokeys with the key IDs saved in a key list. As in the original NTP Version 3 authentication scheme, the Autokey protocol operates separately for each association, so there may be several autokey sequences operating independently at the same time.

An autokey is computed from four fields in network byte order as shown below:

```
+-----------+-----------+-----------+-----------+
| Source IP |  Dest IP  |  Key ID   |  Cookie   |
+-----------+-----------+-----------+-----------+
```

The four values are hashed by the MD5 message digest algorithm to produce the 128-bit key value, which in the reference implementation is stored along with the key ID in a cache used for symmetric keys as well as autokeys. Keys are retrieved from the cache by key ID using hash tables and a fast lookup algorithm.

The NTP packet format has been augmented to include one or more extension fields piggybacked between the original NTP header and the message authenticator code (MAC) at the end of the packet. For packets without extension fields, the cookie is a shared private value conveyed in encrypted form. For packets with extension fields, the cookie has a default public value of zero, since these packets can be validated independently using digital signatures.

For use with IPv4, the Source IP and Dest IP fields contain 32 bits; for use with IPv6, these fields contain 128 bits. In either case the Key ID and Cookie fields contain 32 bits. Thus, an IPv4 autokey has four 32-bit words, while an IPv6 autokey has ten 32-bit words. The source and destination IP addresses and key ID are public values visible in the packet, while the cookie can be a public value or shared private value, depending on the mode.

There are some scenarios where the use of endpoint IP addresses may be difficult or impossible. These include configurations where network address translation (NAT) devices are in use or when addresses are changed during an association lifetime due to mobility constraints. For Autokey, the only restriction is that the addresses visible in the transmitted packet must be the same as those used to construct the autokey sequence and key list and that these addresses be the same as those visible in the received packet. Provisions are included in the reference implementation to handle cases when these addresses change, as possible in mobile IP. For scenarios where the endpoint IP addresses are not available, an optional public identification value could be used instead of the addresses. Examples include the Interplanetary Internet, where bundles are identified by name rather than address. Specific provisions are for further study.

The key list consists of a sequence of key IDs starting with a 32-bit random private value called the autokey seed. The associated autokey is computed as above using the specified cookie and the first 32 bits in

network byte order of this value become the next key ID. Operations continue in this way to generate the entire list, which may have up to 100 entries. It may happen that a newly generated key ID is less than the pivot or collides with another one already generated (birthday event). When this happens, which should occur only rarely, the key list is terminated at that point. The lifetime of each key is set to expire one poll interval after its scheduled use. In the reference implementation, the list is terminated when the maximum key lifetime is about one hour.

The index of the last key ID in the list is saved along with the next key ID for that entry, collectively called the autokey values. The list is used in reverse order, so that the first autokey used is the last one generated. The Autokey protocol includes a message to retrieve the autokey values and signature, so that subsequent packets can be validated using one or more hashes that eventually match the first key ID (valid) or exceed the index (invalid). This is called the autokey test in the following and is done for every packet, including those with and without extension fields. In the reference implementation the most recent key ID received is saved for comparison with the first 32 bits in network byte order of the next following key value. This minimizes the number of hash operations in case a packet is lost.

Autokey Operations

Autokey works differently in the various NTP modes. The scheme used in client/server mode was suggested by Steve Kent over lunch some time ago, but considerably modified since that meal. The server keeps no state for each client, but uses a fast algorithm and a private random value called the server seed to regenerate the cookie upon arrival of a client packet. The cookie is calculated in a manner similar to the autokey, but the key ID field is zero and the cookie field is the server seed. The first 32 bits of the hash is the cookie used for the actual autokey calculation by both the client and server. It is thus specific to each client separately and of no use to other clients or an intruder.

In previous versions of the Autokey protocol the cookie was transmitted in clear on the assumption it was not useful to a wiretapper other than to launch an ineffective replay attack. However, an middleman could intercept the cookie and manufacture bogus messages acceptable to the client. In order to reduce the vulnerability to such an attack, the Autokey Version 2 server encrypts the cookie using a public key supplied by the client. While requiring additional processor resources for the encryption, this makes it effectively impossible to spoof a cookie.

[Note in passing. In an attempt to avoid the use of overt encryption operations, an experimental scheme used a Diffie-Hellman agreed key as a stream cipher to encrypt the cookie. However, not only was the protocol extremely awkward, but the processing time to execute the agreement, encrypt the key and sign the result was horrifically expensive - 15 seconds(!) in a vintage Sun IPC. This scheme was quickly dropped in favor of generic public key encryption.]

In client/server mode the client uses the cookie and each key ID on the key list in turn to retrieve the autokey and generate the MAC in the NTP packet. The server uses the same values to generate the message digest and verifies it matches the MAC in the packet. It then generates the MAC for the response using the same values, but with the IP source and destination addresses exchanged. The client generates the message digest and verifies it matches the MAC in the packet. In order to deflect old replays, the client verifies the key ID matches the last one sent. In this mode the sequential structure of the key list is not exploited, but doing it this way simplifies and regularizes the implementation while making it nearly impossible for an intruder to guess the next key ID.

In broadcast mode clients normally do not send packets to the server, except when first starting up to calibrate the propagation delay in client/server mode. At the same time the client runs the Autokey protocol as in that mode. After obtaining and verifying the cookie, the client continues to obtain and verify the autokey values. To obtain these values, the client must provide the ID of the particular server association, since there can be more than one operating in the same server. For this purpose, the NTP broadcast packet includes the association ID in every packet sent, except when sending the first packet after generating a new key list, when it sends the autokey values instead.

In symmetric mode each peer keeps state variables related to the other. A shared private cookie is conveyed using the same scheme as in client/server mode, except that the cookie is a random value. The key list for each direction is generated separately by each peer and used independently, but each is generated with the same cookie. There exists a possible race condition where each peer sends a cookie request message before receiving the cookie response from the other peer. In this case, each peer winds up with two values, one it generated and one the other peer generated. The ambiguity is resolved simply by computing the working cookie as the exclusive-OR of the two values.

Once the client receives and validates the certificate, subsequent packets containing valid signed extension fields are presumed to contain valid time values, unless these values fall outside the valid interval specified on the certificate. However, unless the system clock has already been set by some other proventic means, it is not known whether these values actually represent a truechime or falsetick source. As the protocol evolves, the NTP associations continue to accumulated time values until a majority clique is available to synchronize the system clock. At this point the NTP intersection algorithm culls the falsetickers from the population and the remaining truechimers are allowed to discipline the clock.

The time values for even falsetick sources form a proventic total ordering relative to the applicable signature timestamps. This raises the interesting issue of how to mitigate between the timestamps of different associations. It might happen, for instance, that the timestamp of some Autokey message is ahead of the system clock by some presumably small amount. For this reason, timestamp comparisons between different associations and between associations and the system clock are avoided, except in the NTP intersection and clustering algorithms.

Once the Autokey values have been instantiated, the protocol is normally dormant. In all modes except broadcast, packets are normally sent without extension fields, unless the packet is the first one sent after generating a new key list or unless the client has requested the cookie or autokey values. If for some reason the client clock is stepped, rather than slewed, all cryptographic and time values for all associations are purged and the Autokey protocol restarted from scratch in all associations. This insures that stale values never propagate beyond a clock step.

Public Key Signatures and Timestamps

While public key signatures provide strong protection against misrepresentation of source, computing them is expensive. This invites the opportunity for an intruder to clog the client or server by replaying old messages or to originate bogus messages. A client receiving such messages might be forced to verify what turns out to be an invalid signature and consume significant processor resources.

In order to foil such attacks, every signed extension field carries a timestamp in the form of the NTP seconds at the signature epoch. The signature span includes the timestamp itself together with optional additional data. If the Autokey protocol has verified a proventic source and the NTP algorithms have validated the time values, the system clock can be synchronized and signatures will then carry a nonzero (valid) timestamp. Otherwise the system clock is unsynchronized and signatures carry a zero (invalid) timestamp. Extension fields with invalid timestamps are discarded before any values are used or signatures verified.

There are three signature types currently defined:

1. Cookie signature/timestamp: Each association has a cookie for use when generating a key list. The cookie value is determined along with the cookie signature and timestamp upon arrival of a cookie request message. The values are returned in a a cookie response message.

2. Autokey signature/timestamp: Each association has a key list for generating the autokey sequence. The autokey values are determined along with the autokey signature and timestamp when a new key list is generated, which occurs about once per hour in the reference implementation. The values are returned in a autokey response message.

3. Public values signature/timestamp: The public key, certificate and leapsecond table values are signed at the time of generation, which occurs when the system clock is first synchronized to a proventic source, when the values have changed and about once per day after that, even if these values have not changed. During protocol operations, each

of these values and associated signatures and timestamps are returned in the associated request or response message. While there are in fact

three public value signatures, the values are all signed at the same time, so there is only one public value timestamp.

The most recent timestamp of each type is saved for comparison. Once a valid signature with valid timestamp has been received, messages with invalid timestamps or earlier valid timestamps of the same type are discarded before the signature is verified. For signed messages this deflects replays that otherwise might consume significant processor resources; for other messages the Autokey protocol deflects message modification or replay by a wiretapper, but not necessarily by a middleman. In addition, the NTP protocol itself is inherently resistant to replays and consumes only minimal processor resources.

All cryptographic values used by the protocol are time sensitive and are regularly refreshed. In particular, files containing cryptographic basis values used by signature and encryption algorithms are regenerated from time to time. It is the intent that file regenerations occur without specific advance warning and without requiring prior distribution of the file contents. While cryptographic data files are not specifically signed, every file name includes an extension called the filestamp, which is a string of decimal digits representing the NTP seconds at the generation epoch.

Filestamps and timestamps can be compared in any combination and use the same conventions. It is necessary to compare them from time to time to determine which are earlier or later. Since these quantities have a granularity only to the second, such comparisons are ambiguous if the values are the same. Thus, the ambiguity must be resolved for each comparison operation as described below.

It is important that filestamps be proventic data; thus, they cannot be produced unless the producer has been synchronized to a proventic source. As such, the filestamps represent a total ordering of creation epoches and serve as means to expunge old data and insure new data are consistent. As the data are forwarded from server to client, the filestamps are preserved, including those for certificate and leapseconds files. Packets with older filestamps are discarded before spending cycles to verify the signature.

Autokey Dances

This section presents an overview of the three Autokey protocols, called dances, corresponding to the NTP client/server, broadcast and symmetric active/passive modes. Each dance is designed to be nonintrusive and to require no additional packets other than for regular NTP operations. The NTP protocol and Autokey dance operate independently and simultaneously and use the same packets. When the Autokey dance is over, subsequent packets are authenticated by the autokey sequence and thus considered proventic as well. Autokey assumes clients poll servers at a relatively low rate, such as once per minute. In particular, it is assumed that a request sent at one poll opportunity will normally result in a response before the next poll opportunity.

The Autokey protocol data unit is the extension field, one or more of which can be piggybacked in the NTP packet. An extension field contains either a request with optional data or a response with data. To avoid deadlocks, any number of responses can be included in a packet, but only one request. A response is generated for every request, even if the requestor is not synchronized or proventicated. Some requests and most responses carry timestamped signatures. The signature covers the data, timestamp and filestamp, where applicable. Only if the packet passes all extension field tests is the signature verified.

Dance Steps

The protocol state machine is very simple. The state is determined by nine bits, four provided by the server, five determined by the client association operations. The nine bits are stored along with the digest/signature scheme identifier in the host status word of the server and in the association status word of the client. In all dances the client first sends an Association request message and receives the Association response specifying which cryptographic values the server is prepared to offer and the digest/signature scheme it will use.

If compatible, the client installs the server status word as the association status word and sends a Certificate request message to the server. The server returns a Certificate response including the certificate and signature. The reference implementation requires the certificate to be self-signed, which serves as an additional consistency check. This check may be removed in future and replaced with a certificate trail mechanism. If the certificate contents and signature are valid, NTP timestamps in this and subsequent messages with valid signatures are considered proventic.

In client/server mode the client sends a Cookie request message including the public key of the host key. The server constructs the cookie as described above and encrypts it using this key. It sends a Cookie response including the encrypted cookie to the client and expunges all values resulting from the calculations in order to remain stateless. The client verifies the signature and decrypts the cookie. A similar dance is used in symmetric modes, but the cookie is generated as a random value.

The cookie is used to construct the key list and autokey values in all modes. In client/server mode there is no need to provide these values to the server, so once the cookie has been obtained the client can generate the key list and validate succeeding packets directly. In other modes the client retrieves the autokey values from the server using an Autokey message exchange. Once these values have been received, the client validates succeeding packets using the autokey sequence as described previously.

A final exchange occurs when the server has the leapseconds table, as indicated in the host status word. If so, the client obtains the table and compares the filestamp with its own leapseconds table filestamp, if available. If the server table is newer than the client table, the

client replaces its table with the server table. The client, acting as server, can now provide the most recent table to any of its own dependent clients. In symmetric modes, this results in both peers having the newest table.

Status Words

Each sever and client operating as a server implements a host status word and an association status word with the format and content shown below. The low order four host status bits are lit during host initialization, depending on whether cryptographic data files are present or not; the next four association bits are dark. There are two additional bits implemented separately. The high order 16 bits specify the message digest/signature encryption scheme.

The host status word is provided to clients in the Association response message. The client initializes the association status word and then lights and dims the association bits as the dance proceeds.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |               |L|K|C|A|L|S|E|E|
|     Digest/Signature NID      | Reserved      |P|E|K|U|P|I|N|N|
|                               |               |T|Y|Y|T|F|G|C|B|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The host status bits are defined as follows:

ENB - Lit if the server implements the Autokey protocol and is prepared to dance.

ENC - Lit if the server has loaded a valid encryption key file. This bit is normally lit, but can dim if an error occurs.

SIG - Lit if the server has loaded a valid signature key file. This bit is included primarily for error supervision and can be either lit or dim.

LPF - Lit if the server has loaded a valid leapseconds file. This bit can be either lit or dim.

The client association status bits are defined as follows:

AUT - Lit when the certificate is present and validated. When lit, signed values in subsequent messages are presumed proventic.

CKY - Lit when the cookie is first received and validated.

KEY - Lit when the autokey values are first received and validated. When lit, clients can validate packets without extension fields according to the autokey sequence.

LPT - Lit when the leapseconds table is received and validated.

An additional bit LST not part of the association status word lights when the key list is regenerated and signed and dims when the autokey values are transmitted. This is necessary to avoid livelock under some conditions.

An additional bit LBK not part of the association status word lights when the association transmit timestamp matches the packet originate timestamp and dims otherwise. If lit, this confirms the packet was received in response to one previously sent by this association.

Host State Variables

Host Name
The name of the host returned by the Unix gethostname() library function. It must agree with the subject and issuer name in the certificate.

Host Key
The RSA key from the host key file and used to encrypt/decrypt cookies. It carries the public value timestamp and the filestamp at the host key file creation epoch. This is also the signature key, unless a signature key is specified.

Public Key
The public encryption key for the Cookie request message and derived from the host key. It carries the public value timestamp and the filestamp at the host key file creation epoch.

Sign Key
The RSA or DSA key from the sign key file and used to encrypt signatures. It carries the public value timestamp and the filestamp at the sign key file creation epoch.

Certificate
The X.509 certificate from the certificate file. It carries the public value timestamp and the filestamp at the certificate file creation epoch.

Leapseconds Table, Leapseconds Table Filestamp
The NIST leapseconds table from the NIST leapseconds file. It carries the public value timestamp and the filestamp at the leapseconds file creation epoch.

Digest/signature NID
The identifier of the message digest/signature encryption scheme derived from the sign key. It must agree with the NID on the certificate.

Client Association State Variables

Peer Association ID
The association ID of the peer as received in a response message.

Host Name
The name of the host returned by the Association response. It must agree with the subject name in the certificate.

Digest/Signature NID
The identifier of the message digest/signature encryption scheme returned in the Association response message. It must agree with the value encoded in the certificate.

Public Values Timestamp
The timestamp returned by the latest Certificate response, Cookie request or Leapseconds message.

Certificate
The X.509 certificate returned in the certificate response message, together with its timestamp and filestamp.

Cookie
The cookie returned in a Cookie response message, together with its timestamp and filestamp.

Receive Autokey values
The autokey values returned in an Autokey response message, together with its timestamp and filestamp.

Server Association State Variables (broadcast and symmetric modes)

Association ID
The association ID of the server for use in client request messages.

Send Autokey Values
The autokey values, signature and timestamp.

Key List
A sequence of key IDs starting with a random autokey seed and each pointing to the next. It is computed timestamped and signed at the next poll opportunity when the key list is empty.

Autokey Seed
The private value used to initialize the key list. It is randomized for each new key list.

Current Key Number
The index of the entry on the Key List to be used at the next poll opportunity.

Send Encrypt Values (symmetric modes only)
The encrypted cookie, signature and timestamp computed upon arrival of the Cookie request message. These data are held until the next poll opportunity.

Server seed

The private value hashed with the IP addresses to construct the cookie used in client/server mode. It is randomized when the public value signatures are refreshed.

Autokey Messages

There are currently five Autokey request types and five corresponding responses. An abbreviated description of these messages is given below; the detailed formats are described in Appendix A.

Association Message (1)
The client sends the request to retrieve the host status word and host name. The server responds with these values.

Certificate Message (2)
The client sends the request to retrieve the server certificate. The server responds with the certificate.

Cookie Message (3)
The client sends the request, including the public member of the host key, to retrieve the cookie. The server responds with the cookie encrypted with the public key.

Autokey Message (4)
The client sends the request to retrieve the autokey values, if available. The server responds with these values.

Leapseconds Message (5)
The client sends the request including its leapseconds table, if available. The server responds with its own leapseconds table. Both the client and server agree to use the version with the latest filestamp.

State Transitions

The state transitions of the three dances are shown below. The capitalized truth values represent the association status word bits, except for the SYNC value, which is true when the host is synchronized to a proventic source and false otherwise. All truth values are initialized false and become true upon the arrival of a specific response messages, as detailed in the above status bits description.

Client/Server Dance

The client/server dance begins when the client sends an Association request message to the server. It ends upon arrival of the Cookie response, which lights the CKY and KEY bits. Subsequent packets received without extension fields are validated by the autokey sequence. An optional final exchange is possible to retrieve the leapseconds table.

```
        while (1) {
                wait_for_next_packet;
                make_NTP_header;
                if (response_ready)
```



```
                        send_response;
                if (!ENB)
                        send_Association_request;
                else if (!CRF)
                        send_Certificate_request;
                else if (!CKY)
                        send_Cookie_request;
                else if (LPF & !LPT)
                        send_Leapseconds_request;
        }
```

Broadcast Client Dance

The broadcast client dance begins when the client receives the first broadcast packet, which includes an Association response with the association ID. The broadcast client uses the association ID to initiate a client/server dance in order to calibrate the propagation delay. The dance ends upon arrival of the Autokey response, which lights the KEY bit. Subsequent packets received without extension fields are validated by the autokey sequence. An optional final exchange is possible to retrieve the leapseconds table. When the server generates a new key list, the server replaces the Association response with an Autokey response in the first packet sent.

```
        while (1) {
                wait_for_next_packet;
                make_NTP_header;
                if (response_ready)
                        send_response;
                if (!ENB)
                        send_Association_request;
                else if (!CRF)
```

```
                send_Certificate_request;
        else if (!CKY)
                send_Cookie_request;
        else if (!KEY)
                send_Autokey_request;
        else if (LPF & !LPT)
                send_Leapseconds_request;
}
```

Symmetric Dance

The symmetric active dance begins when the active peer sends an Association request to the passive peer. The passive peer mobilizes an association and steps the same dance from the beginning. Until the active peer is synchronized to a proventic source (which could be the passive peer) and can sign messages, the passive peer will loop waiting to light the CRF bit and the active peer will skip the cookie exchange.

Meanwhile, the active peer retrieves the certificate and autokey values from the passive peer and lights the KEY bit. When for some reason either peer generates a new key list, at the first opportunity the peer

sends the autokey values; that is, it pushes the values rather than pulls them. This is to prevent a possible deadlock where each peer is waiting for values from the other one.

```
while (1) {
        wait_for_next_packet;
        make_NTP_header;
        if (response_ready)
                send_response;
        if (!ENB)
                send_Association_request;
        else if (!CRF)
                send_Certificate_request;
        else if (!CKY & SYNC)
                send_Cookie_request;
        else if (LST)
                send_Autokey_response;
        else if (!KEY)
                send_Autokey_request;
        else if (LPF & !LPT & SYNC)
                send_Leapseconds_request;
}
```

Once the active peer has synchronized to a proventic source, it includes timestamped signatures with its messages. The passive peer, which has been stalled waiting for the CRF bit to light and the active peer, which now finds the SYNC bit lit, continues their respective dances. The next message sent by either peer is a Cookie request. The recipient rolls a random cookie, lights its CKY bit and returns the encrypted cookie in the Cookie response. The recipient decrypts the cookie and lights its CKY bit.

It is not a protocol error if both peers happen to send a cookie request at the same time. In this case both peers will have two values, one generated by one peer and the other received from the other peer. In such cases the working cookie is constructed as the exclusive-OR of the two values.

At the next packet transmission opportunity, either peer generates a new key list and lights the LST bit; however, there may already be an Autokey request queued for transmission and the rules say no more than one request in a packet. When available, either peer sends an Autokey response and clears the LST bit. The recipient initializes the autokey values, clears the LST bit and lights the KEY bit. Subsequent packets received without extension fields are validated by the autokey sequence.

The above description assumes the active peer synchronizes to the passive peer, which itself is synchronized to some other source, such as a radio clock or another NTP server. In this case, the active peer is operating at a stratum level one greater than the passive peer and so the passive peer will not synchronize to it unless it loses its own sources and the active peer itself has another source.

Key Refreshment

About once per day the server seed is randomized and the signatures recomputed. The operations are:

```
while (1) {
        wait_for_next_refresh;
        crank_random_generator;
        generate_autokey_private_value;
        if (!SYNC)
                continue;
        update_public_value_timestamp;
```

```
                compute_signatures;
        }
```

Error Recovery

The protocol state machine which drives the various Autokey operations includes provisions for various kinds of error conditions that can arise due to missing files, corrupted data, protocol violations and packet loss or misorder, not to mention hostile intrusion. There are two mechanisms which maintain the liveness state of the protocol, the reachability register defined in RFC-1305 and the watchdog timer, which is new in NTP Version 4.

The reachability register is an 8-bit register that shifts left with 0 replacing the rightmost bit. A shift occurs for every poll interval, whether or not a poll is actually sent. If an arriving packet passes all authentication and sanity checks, the rightmost bit is set to 1. If any bit in this register is a 1, the server is reachable, otherwise it is unreachable. If the server was once reachable and then becomes unreachable, a general reset is performed. A general reset reinitializes all association variables to the state when first mobilized and returns all acquired resources to the system. In addition, if the association is not configured, it is demobilized until the next packet is received.

The watchdog timer increments for every poll interval, whether or not a poll is actually sent and regardless of the reachability state. The counter is set to zero upon arrival of a packet from a proventicated source, as determined by the Autokey protocol. In the reference implementation, if the counter reaches 16 a general reset is performed. In addition, if the association is configured, the poll interval is doubled. This reduces the network load for packets that are unlikely to elicit a response.

At each state in the protocol the client expects a particular response from the server. A request is included in the NTP message sent at each poll interval until a valid response is received or a general reset occurs, in which case the protocol restarts from the beginning. In some cases noted below, certain kinds of errors cause appropriate action which avoids the somewhat lengthy timeout/restart cycle. While this behavior might be considered rather conservative, the advantage is that

old cryptographic and time values can never persist from one mobilization to the next.

There are a number of situations where some event happens that causes the remaining autokeys on the key list to become invalid. When one of these situations happens, the key list and associated autokeys in the key cache are purged. A new key list, signature and timestamp are generated when the next NTP message is sent, assuming there is one. Following is a list of these situations.

1. When the cookie value changes for any reason.

2. When a client switches from client/server mode to broadcast mode. There is no further need for the key list, since the client will not transmit again.

3. When the poll interval is changed. In this case the calculated expiration times for the keys become invalid.

4. When a general reset is performed.

5. If a problem is detected when an entry is fetched from the key list. This could happen if the key was marked non-trusted or timed out, either of which implies a software bug.

6. When the signatures are refreshed, the key lists for all associations are purged.

7. When the client is first synchronized or the system clock is stepped, the key lists for all associations are purged.

There are special cases designed to quickly respond to broken associations, such as when a server restarts or refreshes keys. Since the client cookie is invalidated, the server rejects the next client request and returns a crypto-NAK packet. Since the crypto-NAK has no MAC, the problem for the client is to determine whether it is legitimate or the result of intruder mischief. In order to reduce the vulnerability to such mischief, the crypto-NAK is believed only if the result of a previous packet sent by the client, as confirmed by the LBK status bit. This bit is lit in the NTP protocol if the packet originate timestamp matches the association transmit timestamp. While this defense can be easily circumvented by a middleman, it does deflect other kinds of intruder warfare. The LBK bit is also used to validate most responses and some requests as well.

Security Analysis

This section discusses the most obvious security vulnerabilities in the various Autokey dances. Throughout the discussion the cryptographic algorithms themselves are assumed secure; that is, a brute force cryptanalytic attack will not reveal the host private key or sign private key or cookie value or server seed or autokey seed or be able to predict the random generator values.

There are three tiers of defense against intruder attacks. The first is a keyed message digest including a secret cookie conveyed in encrypted form. A packet is discarded if the message digest does not match the MAC. The second tier is the autokey sequence, which is generated by repeated hashes starting from a secret server seed and used in reverse order. While any receiver can authenticate a packet relative to the last one received and by induction to a signed extension field, as a practical matter a wiretapper cannot predict the next autokey and thus cannot spoof a valid packet. The third tier is timestamped signatures which reliably bind the autokey values to the private key of a trusted server.

In addition to the three-tier defense strategy, all packets are protected by the NTP sanity checks. Since NTP packets carry time values, replays of old or bogus packets can be deflected once the client has synchronized to proventic sources. Additional sanity checks involving timestamps and filestamps are summarized in Appendix C.

During the Autokey dances when extension fields are in use, the cookie is a public value (0) rather than a shared private value. Therefore, an intruder can easily construct a packet with a valid MAC; however, once the certificate is stored, extension fields carry timestamped signatures and bogus packets are readily avoided. While most request messages are unsigned, only the Association response message is unsigned. This message is used in the first packet sent by a server or peer and in most NTP broadcast packets.

A bogus Association response message can cause a client livelock or deadlock condition. However, these packets do not affect NTP time values and do not consume significant resources. To reduce the vulnerability to bogus packets, the NTP transmit timestamp in the Association and Certificate request messages is used as a nonce. The NTP server copies this value to the originate timestamp in the NTP header, so that the client can verify that the message is a response to the original request. To minimize the possibility that an intruder can guess the nonce, the client should fill in the low order unused bits in the transmit timestamp with random values. In addition, replays of all except Autokey response messages are discarded before the signatures are verified.

In client/server and symmetric modes extension fields are no longer needed after the Autokey dance has concluded. The client validates the packet using the message digest and autokey sequence. A successful middleman attack is unlikely, since without the server seed the intruder cannot produce the cookie and without the cookie cannot produce a valid MAC. In broadcast mode a wiretapper cannot synthesize a valid packet without the autokey seed, so cannot manufacture an bogus packet acceptable to the receiver. The most the intruder can do is replay an old packet causing the client to repeat hash operations until exceeding the maximum key number. On the other hand, a middleman could do real harm by intercepting a packet, using the key ID to generate a correct autokey and then synthesizing a bogus packet. There does not seem to be

a suitable solution for this as long as the server has no per-client state.

A client instantiates cryptographic variables only if the server is synchronized to a proventic source. A host does not sign values or generate cryptographic data files unless synchronized to a proventic source. This raises an interesting issue; how does a client generate proventic cryptographic files before it has ever been synchronized to a proventic source? Who shaves the barber if the barber shaves everybody in town who does not shave himself? In principle, this paradox is resolved by assuming the primary (stratum 1) servers are proventicated by external phenomological means.

Cryptanalysis

Some observations on the particular engineering constraints of the Autokey protocol are in order. First, the number of bits in some cryptographic values are considerably smaller than would ordinarily be expected for strong cryptography. One of the reasons for this is the need for compatibility with previous NTP versions; another is the need for small and constant latencies and minimal processing requirements. Therefore, what the scheme gives up on the strength of these values must be regained by agility in the rate of change of the cryptographic basis values. Thus, autokeys are used only once and basis values are regenerated frequently. However, in most cases even a successful cryptanalysis of these values compromises only a particular client/server association and does not represent a danger to the general population.

While the protocol has not been subjected to a formal analysis, a few preliminary assertions can be made. The protocol cannot loop forever in any state, since the association timeout and general reset insure that the association variables will eventually be purged and the protocol restarted from the beginning. However, if something is seriously wrong, the timeout/restart cycle could continue indefinitely until whatever is wrong is fixed.

Clogging Attacks

There are two clogging vulnerabilities exposed in the protocol design: a sign attack where the intruder hopes to clog the victim server with needless signature computations, and a verify attack where the intruder attempts to clog the victim client with needless verification computations. Autokey uses public key encryption algorithms for both signature and cookie encryption and these algorithms require significant processor resources.

In order to reduce the exposure to a sign attack, signatures are computed only when the data have changed. For instance, the autokey values are signed only when the key list is regenerated, which happens about once an hour, while the public values are signed only when the values are refreshed, which happens about once per day. However, in client/server mode the protocol precludes server state variables on

behalf of an individual client, so the cookie must be computed, encrypted and signed for every cookie response. Ordinarily, cookie requests are seldom used, except when the server seed or public value signatures are refreshed. However, a determined intruder could replay cookie requests at high rate, which may very well clog the server. There appears no easy countermeasure for this particular attack.

A verify attack attempts to clog the receiver by provoking spurious signature verifications. The signature timestamp is designed to deflect replays of packets with old or duplicate extension fields before invoking expensive signature operations. A bogus signature with a timestamp in the future could do this, but the autokey sequence would detect this, since success would require cryptanalysis of both the server seed and autokey seed.

Since the Certificate response is signed, a middleman attack will not compromise the certificate data; however, a determined middleman could hammer the client with intentionally defective Certificate responses before a valid one could be received and force spurious signature verifications, which of course would fail. An intruder could flood the server with Certificate request messages, but the Certificate response message is signed only once, so the result would be no worse than flooding the network with spurious packets.

An interesting vulnerability in client/server mode is for an intruder to replay a recent client packet with an intentional bit error. This could cause the server to return a crypto-NAK packet, which would then cause the client to request the cookie and result in a sign attack on the server. This results in the server and client burning spurious machine cycles and resulting in denial of service. As in other cases mentioned previously, the NTP timestamp check greatly reduces the likelihood of a successful attack.

In broadcast and symmetric modes the client must include the association ID in the Autokey request. Since association ID values for different invocations of the NTP daemon are randomized over the 16-bit space, it is unlikely that a very old packet would contain a valid association ID value. An intruder could save old server packets and replay them to the client population with the hope that the values will be accepted and cause general chaos. The conservative client will discard them on the basis of invalid timestamp.

As mentioned previously, an intruder could pounce on the initial volley between peers in symmetric mode before both peers have determined each other reachable. In this volley the peers are vulnerable to an intruder using fake timestamps. The result can be that the peers never synchronize the timestamps and never completely mobilize their associations. A clever intruder might notice the interval between public value signatures and concentrate attack on the vulnerable intervals. An obvious countermeasure is to randomize these intervals. A more comprehensive countermeasure remains to be devised.

Appendix A. Packet Formats

The NTP Version 4 packet consists of a number of fields made up of 32-bit (4 octet) words in network byte order. The packet consists of three components, the header, one or more optional extension fields and an optional message authenticator code (MAC), consisting of the Key ID and Message Digest fields. The format is shown below, where the size of some multiple word fields is shown in words.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|LI | VN  |Mode |    Stratum    |     Poll      |   Precision   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
|                          Root Delay                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Root Dispersion                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Reference ID                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                   Reference Timestamp (2)                     |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                   Originate Timestamp (2)                     |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                    Receive Timestamp (2)                      |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                    Transmit Timestamp (2)                     |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                      Extension Field(s)                       =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Key ID                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
|                      Message Digest (4)                       |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The NTP header extends from the beginning of the packet to the end of the Transmit Timestamp field. The format and interpretation of the header fields are backwards compatible with the NTP Version 3 header

fields as described in RFC-1305, except for a slightly modified computation for the Root Dispersion field. In NTP Version 3, this field includes an estimated jitter quantity based on weighted absolute differences, while in NTP Version 4 this quantity is based on weighted root-mean-square (RMS) differences.

An unauthenticated NTP packet includes only the NTP header, while an authenticated one contains in addition a MAC. The format and interpretation of the NTP Version 4 MAC is described in RFC-1305 when using the Digital Encryption Standard (DES) algorithm operating in Cipher-Block Chaining (CBC) node. This algorithm and mode of operation is no longer supported in NTP Version 4. The preferred replacement in both NTP Version 3 and 4 is the Message Digest 5 (MD5) algorithm, which is included in the distribution. For MD5 the Message Digest field is 4 words (8 octets), but the Key ID field remains 1 word (4 octets).

Extension Field Format

In NTP Version 4 one or more extension fields can be inserted after the NTP header and before the MAC, which is always present when an extension field is present. The extension fields can occur in any order; however, in some cases there is a preferred order which improves the protocol efficiency. While previous versions of the Autokey protocol used several different extension field formats, in version 2 of the protocol only a single extension field format is used.

Each extension field contains a request or response message in the following format:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|R|E|  Version  |     Code      |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Filestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Value Length                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                             Value                             =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Signature Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                           Signature                           =
```

```
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Each extension field except the last is padded to a word (4 octets) boundary, while the last is padded to a doubleword (8 octets) boundary. The Length field covers the entire field length, including the Length field itself and padding. While the minimum field length is 8 octets, a maximum field length remains to be established. The reference implementation discards any packet with an extension field length over 1024 octets.

The presence of the MAC and extension fields in the packet is determined from the length of the remaining area after the header to the end of the packet. The parser initializes a pointer just after the header. If the length is not a multiple of 4, a format error has occurred and the packet is discarded. If the length is zero the packet is not authenticated. If the length is 4 (1 word), the packet is an error report or crypto-NAK resulting from a previous packet that failed the message digest check. The 4 octets are presently unused and should be set to 0. If the length is 8 (2 words), 12 (3 words) or 16 (4 words), the packet is discarded with a format error. If the length is greater than 20 (5 words), one or more extension fields are present.

If an extension field is present, the parser examines the length field. If the length is less than 4 or not a multiple of 4, a format error has occurred and the packet is discarded; otherwise, the parser increments the pointer by this value. The parser now uses the same rules as above to determine whether a MAC is present and/or another extension field. An additional implementation-dependent test is necessary to ensure the pointer does not stray outside the buffer space occupied by the packet.

In the Autokey Version 2 format, the Code field specifies the request or response operation, while the Version field is 2 identifying the current protocol version. There are two flag bits defined. Bit 0 is the response flag (R) and bit 1 is the error flag (E); the other six bits are presently unused and should be set to 0. The remaining fields will be described later.

In the most common protocol operations, a client sends a request to a server with an operation code specified in the Code field and the R bit set to 0. Ordinarily, the client sets the E bit to 0 as well, but may in future set it to 1 for some purpose. The Association ID field is set to the value previously received from the server or 0 otherwise. The server returns a response with the same operation code in the Code field and the R bit set to 1. The server can also set the E bit to 1 in case of error. The Association ID field is set to the association ID sending the response as a handle for subsequent exchanges. If for some reason the association ID value in a request does not match the association ID of any mobilized association, the server returns the request with both the R and E bits set to 1. Note that, it is not a protocol error to send an unsolicited response with no matching request.

In some cases not all fields may be present. For instance, when a client has not synchronized to a proventic source, signatures are not valid. In such cases the Timestamp and Signature Length fields are 0 and the Signature field is empty. Some request and error response messages carry no value or signature fields, so in these messages only the first two words are present. The extension field parser verifies that the extension field length is at least 8 if no value field is expected and at least 24 if it is. The parser also verifies that the sum of the value and signature lengths is equal to or less than the extension field length.

The Timestamp and Filestamp words carry the seconds field of the NTP timestamp. The Timestamp field establishes the signature epoch of the data field in the message, while the filestamp establishes the generation epoch of the file that ultimately produced the data that was signed. Since a signature and timestamp are valid only when the signing host is synchronized to a proventic source and a cryptographic data file can only be generated if a signature is possible, the filestamp is always nonzero, except in the Association Response message, where it contains the server status word.

Autokey Version 2 Messages

Association Message

The Association message is used to obtain the host name and related values. The request message has the following format:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|0|     1     |       1       |               8               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
|                               0                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The response message has the following format:

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|E|     1     |       1       |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               0                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Public Value Timestamp                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Status Word                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Host Name Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
```



```
=                           Host Name                           =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               0                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This is the only response that is accepted if the association status word is zero; otherwise, it is ignored. As this is the first request sent and the response is not from an association, the Association ID fields are 0. The Host Name field contains the unterminated string returned by the Unix gethostname() library function. The Status Word is defined in previously in this memorandum. While minimum and maximum host name lengths remain to be established, the reference implementation uses the values 4 and 256, respectively.

Certificate Message

The Certificate message is used to obtain the certificate and related values. The request message has the following format:

```
                    1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|0|     2     |       2       |               8               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The response message has the following format:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|E|     2     |       2       |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Public Values Timestamp                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Certificate Filestamp                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Certificate Length                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                          Certificate                          =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Certificate Signature Length                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
```



```
|                                                               |
=                     Certificate Signature                     =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The response is accepted only if the association status word is nonzero, AUT = 0 and LBK = 1. The certificate is encoded in X.509 format using ASN.1 syntax. If the certificate has expired or for some reason is no longer available, the response includes only the first two words with the E bit set. The remaining fields are defined previously in this memorandum.

Cookie Message

The Cookie is used in client/server and symmetric modes to obtain the server cookie. The request message has the following format:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|0|     3     |       3       |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Public Values Timestamp                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Certificate Filestamp                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Public Key Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                           Public Key                          =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Public Key Signature Length                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                       Public Key Signature                    =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The request is accepted only if AUT = 1, CKY = 0 and LBK = 1. The Public Key field contains the server public key values to be used for cookie encryption. The values are encoded in ASN.1 format. The remaining fields are defined previously in this memorandum.

The response message has the following format:


```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
|1|E|     3     |       3        |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Cookie Timestamp                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Certificate Filestamp                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Encrypted Cookie Length                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                        Encrypted Cookie                       =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Cookie Signature Length                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                        Cookie Signature                       =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The response is accepted only if AUT = 1 and LBK = 1. The Cookie Timestamp, Encrypted Cookie and Cookie Signature fields are determined upon arrival of the request message. The Encrypted Cookie field contains the encrypted cookie value according to the public key provided in the request. If CKY = 0, the decrypted cookie is used directly. If CKY = 1, the decrypted cookie is exclusive-ORed with the existing cookie. If an error occurs when decoding the public key or encrypting the cookie, the response includes only the first two words with the E bit set. The remaining fields are defined previously in this memorandum.

Autokey Message

The Autokey message is used to obtain the autokey values. The request message has the following format:

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|0|     2     |       4        |              8                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The response message has the following format:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|E|     4     |       4       |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Autokey Timestamp                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Certificate Filestamp                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               8                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Key ID                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Key Number                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Autokey Signature Length                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                       Autokey Signature                       =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The response is accepted only if AUT = 1 and KEY = 0 in the association status word; otherwise, it is ignored. The Autokey Timestamp, Key ID, Key Number and Autokey Signature fields are determined when the most recent key list was generated. If a key list has not been generated or the association ID matches no mobilized association, the response includes only the first two words with the E bit set. The remaining fields are defined previously in this memorandum.

Leapseconds Table Message

The Leapseconds Table message is used to exchange leapseconds tables. The request and response messages have the following format, except that the R bit is set in the response:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
|0|0|     2     |       5       |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Public Values Timestamp                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Leapseconds Filestamp                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Leapseconds Table Length                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```



```
|                                                               |
|                                                               |
=                       Leapseconds Table                       =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Leapseconds Signature Length                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                     Leapseconds Signature                     =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The response is accepted only if AUT = 1 and LPT = 0 in the association status word; otherwise, it is ignored. The Leapseconds Table field contains the leapseconds table as parsed from the leapseconds file available from NIST. In client/server mode the client requests the table from the server when the LPF bit is set in the host status word. If the client already has a copy, it uses the one with the latest filestamp. In symmetric modes the peers exchange tables and both use the one with the latest filestamp. If the leapseconds table is requested but unavailable, the response includes only the first two words with the E bit set. The remaining fields are defined previously in this memorandum.

Appendix B. Key Generation and Management

The ntp-genkeys utility program in the NTP software distribution generates public/private key, certificate request and certificate files. A set of files is generated for every message digest and signature encryption scheme supported by the OpenSSL software library. All files are based on a pseudo-random number generator seeded in such a way that random values are exceedingly unlikely to repeat. The files are PEM encoded in printable ASCII format suitable for mailing as MIME objects. The file names include the name of the generating host together with the filestamp, as described previously in this memorandum.

The generated files are typically stored in a shared directory in NFS mounted file systems, with files containing private keys obscured to all but root. Links from default file names assumed by the NTP daemon are installed to the selected files for the host key, sign key and host certificate. Since the files of successive generations and different hosts have unique names, there is no possibility of name collisions. An extensive set of consistency checks avoids linking from a particular host to the files of another host, for example.

The ntp-genkeys program generates public/private key files for both the RSA and DSA encryption algorithms with a default modulus of 512 bits. The host key used for cookie encryption must be RSA. By default, the same key is used for signature encryption. However, a different RSA key or a DSA key can be specified for signature encryption.

The ntp-genkeys program also generates certificate request and self-signed certificate files. The X.509 certificate request used by Autokey includes at the minimum these values and possibly related information needed by an external certificate authority. Autokey expects the subject name and issuer name to be the same as the generating host name.

The program avoids the need for a serial number file by using the filestamp as the certificate serial number. By default, certificates are valid for one year following the time of generation, although these conventions may change. Also, the program assumes X.509 version 1 formats, although this may change to version 3 in future. Other implementations might have different conventions.

Appendix C. Packet Processing Rules

Exhaustive examination of possible vulnerabilities at the various processing steps of the NTP protocol as specified in RFC-1305 have resulted in a revised list of packet sanity tests. There are 12 tests, called TEST1 through TEST12 in the reference implementation, which are performed in a specific order designed to gain maximum diagnostic information while protecting against an accidental or malicious clogging attack. These tests are described in detail in the Flash Codes section of the ntpq documentation page at www.eecis.udel.edu/~ntp/ntp_spool/html/ntpq.htm.

The sanity tests are divided into three tiers as previously described. The first tier deflects access control and packet message digest violations. The second deflects packets from broken or unsynchronized servers and replays. The third deflects packets with invalid header fields or time values with excessive errors. However, the tests in this last group do not directly affect cryptographic the protocol vulnerability, so are beyond the scope of discussion here.

When a host initializes, it reads its own host key, sign key and certificate files, which are required for continued operation. Optionally, it reads the leapseconds file, when available. When reading these files the host checks the filestamps for validity; for instance, all filestamps must be later than the time the UTC timescale was established in 1972 and the certificate filestamp must not be earlier than the sign key filestamp (or host key filestamp, if that is the default sign key). In general, at the time the files are read, the host is not synchronized, so it cannot determine whether the filestamps are bogus other than these simple checks.

Once a client has synchronized to a proventic source, additional checks are implemented as each message arrives. In the following the relation A -> B is Lamport's "happens before" relation which is true if event A happens before event B. Here the relation is assume to hold if event A is simultaneous with event B, unless noted to the contrary. The following assertions are required:

1. For timestamp T and filestamp F, F->T; that is, the timestamp must not be earlier than the filestamp.

2. In client and symmetric modes, for host key filestamp H, public key timestamp P, cookie timestamp C and autokey timestamp A, H->P->C->A; that is, once the cookie is generated an earlier cookie will not be accepted, and once the key list and autokey values are generated, earlier autokey values will not be accepted.

3. For sign file S and certificate filestamp C specifying begin time B and end time E, S->C->B->E; that is, the valid period must be nonempty and not retroactive.

4. For timestamp T, begin time B and end time E, B->T->E; that is, the timestamp T is valid from the beginning if second B through the end of

second E. This raises the interesting possibilities where a truechimer server with expired certificate or a falseticker with valid certificate are not detected until the client has synchronized to a clique of proventic truechimers.

5. For each of signatures, the client saves the most recent valid timestamp T0 and filestamp F0. For every received message carrying timestamp T1 and filestamp F1, the message is discarded unless T0->T1 and F0->F1; however, if the KEY bit of the association status word is dim, the message is not discarded if T1 = T0; that is, old messages are discarded and, in addition, if the server is proventic, the message is discarded if an old duplicate.

An interesting question is what happens if during regular operation a certificate becomes invalid. The behavior assumed is identical to the case where an incorrect sign key were used. Thus, the next time a client attempts to verify an autokey signature, for example, the operation would fail and eventually cause a general client reset and restart.

Security Considerations

Security issues are the main topic of this memorandum.

References

Note: Internet Engineering Task Force documents can be obtained at www.ietf.org. Other papers and reports can be obtained at www.eecis.udel.edu/~mills. Additional briefings in PowerPoint, PostScript and PDF are at that site in ./autokey.htm.

1. Bradner, S. Key words for use in RFCs to indicate requirement levels. Request for Comments RFC-2119, BCP 14, Internet Engineering Task Force, March 1997.

2. Karn, P., and W. Simpson. Photuris: session-key management protocol. Request for Comments RFC-2522, Internet Engineering Task Force, March 1999.

3. Kent, S., R. Atkinson. IP Authentication Header. Request for Comments RFC-2402, Internet Engineering Task Force, November 1998.

4. Kent, S., and R. Atkinson. IP Encapsulating security payload (ESP). Request for Comments RFC-2406, Internet Engineering Task Force, November

1998.

5. Maughan, D., M. Schertler, M. Schneider, and J. Turner. Internet security association and key management protocol (ISAKMP). Request for Comments RFC-2408, Internet Engineering Task Force, November 1998.

6. Mills, D.L. Authentication scheme for distributed, ubiquitous, real-time protocols. Proc. Advanced Telecommunications/Information Distribution Research Program (ATIRP) Conference (College Park MD, January 1997), 293-298.

7. Mills, D.L. Cryptographic authentication for real-time network protocols. In: AMS DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 45 (1999), 135-144.

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

9. Mills, D.L. Proposed authentication enhancements for the Network Time Protocol version 4. Electrical Engineering Report 96-10-3, University of Delaware, October 1996, 36 pp.

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

11. Mills, D.L. Public key cryptography for the Network Time Protocol. Electrical Engineering Report 00-5-1, University of Delaware, May 2000. 23 pp.

12. Orman, H. The OAKLEY key determination protocol. Request for Comments RFC-2412, Internet Engineering Task Force, November 1998.

Author's Address

David L. Mills
Electrical and Computer Engineering Department
University of Delaware
Newark, DE 19716
mail mills@udel.edu, phone 302 831 8247, fax 302 831 4316
web www.eecis.udel.edu/~mills

Full Copyright Statement