Network Working Group David L. Mills
Internet Draft University of Delaware
Document: < draft-ietf-stime-ntpauth-01.txt > April 2001
Category: Standards Track

Public-Key Cryptography for the Network Time Protocol Version 1

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 in 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 intervening servers.

Changes Since the Preceeding Draft

There are a number of changes scattered through this memorandum to clarify the presentation and add a few new features. Among the most important:

1. **An optional parameter negotiation message has been added to the** protocol state machine. The values it may carry and the interpretation of these values are not defined in this memorandum.

2. **A preliminary value exchange has been added to begin the protocol** dance. This is necessary to avoid a vulnerability where unsolicited public key responses could clog the victim with needless signature cycles.

3. **The value exchange, which is piggybacked on the association ID** message, supports a timestamp-based agreement scheme which floods the latest version of the agreement parameters and leapseconds table. Using this scheme any one of a clique of trusted primary servers running symmetric modes with each other and broadcast or client/server modes with the secondary server population can refresh these data at any time and the refreshed data will update all older data everywhere in the NTP subnet within one day.

4. **An optional certificate retrieval operation has been added to the** protocol state machine. While the operation has been implemented and tested, the contents of the certificate itself have not been determined.

5. **A couple of subtle livelock problems with symmetric mode and** broadcast mode were found and fixed. The problem with source addresses not yet bound has been fixed in the reference implementation.

6. **The protocol descriptions and state diagrams have been updated. Some** packet formats have been changed in minor ways.

7. **Provisions for the use of IPv6 addresses in calculating the autokey**

have been added.

8. Provisions for the use of arbitrary identification values to be used in lieu or IP addresses in calculating the autokey have been added.

9. A simplified version of the protocol appropriate for SNTP clients is proposed; details to follow.

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 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 Public-Key Infrastructure (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 is being deployed now 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.

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 all 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 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.

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 processor resources with needless cryptographic calculations.

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 processes. 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 rsaref2.0 algorithms, but other algorithms 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 Byzantine traitors and evil cliques. 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 and can archive** 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 PKI operations.

3. **An intruder can intercept, modify and replay a packet. However, it** cannot permanently prevent the original packet transmission over the net; that is, it cannot break the wire, only congest it.

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 discussed 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** cryptographic 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 keys and agreement parameters, where necessary, 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 eventually be a function of 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 cryptoraphic values are public, so there is no need to associate each interface with different cryptoraphic values. In the NTP design the host name, as returned by the gethostname() 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 as noted.

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 proventicated only when the all 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, including** replay/modification and clogging attacks. In particular, it must be

tolerant of operation or implementation variances, such as packet loss or misorder, or suboptimal configuration.

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 and verification functions.

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

7. **It must not require intricate per-client or per-server configuration** other than the availability of public/private key files and agreement parameter files, as required.

8. **The reference implementation must contain provisions to generate** cryptographic key values, including private/public keys and agreement parameters 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 of the rsaref2.0 library, although other libraries with a compatible interface could be used as well. The reference implementation uses keyed-MD5 message digests to detect packet modification, timestamped RSA digital signatures to verify the source, and Diffie-Hellman key agreements to construct a private shared key from public values. However, there is no reason why alternative signature schemes and agreement algorithms could be supported. 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 given values less than 65536 **and indefinite lifetime. Autokey key IDs have pseudo-random values** equal to or greater than 65536 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 active/passive. All three variants make use of a specially contrived session key called an autokey and a pseudo-random sequence of key IDs called the key list. As in the original NTP Version 3 authentication scheme, the Autokey protocol operates separately for each association, so there may be several key lists operating independently at the same time and with distinct associated values and signatures.

An autokey consists of four fields in network byte order as shown below:

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

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 a private value, depending on the mode.

There are some scenarios where the use of endpoint IP addresses when calculating the autokey 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. As described below, NTP associations are identified by the endpoint IP addresses, so the natural approach is to authenticate associations using these values. For scenarios where this is not possible, an optional identification value can be used instead of the endpoint IP addresses. The Parameter Negotiation message contains an option to specify these data; however, the format, encoding and use of this option are not specified in this memorandum. For the purposes of this memorandum, the endpoint IP addresses will be assumed.

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 private value computed by an agreement algorithm. For packets with extension fields, the cookie has a default public value of zero, since these packets can be validated independently using signed data in the extension fields. The four values are hashed by the message digest algorithm to produce the actual key value, which 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 algorithm.

The key list consists of a sequence of key IDs starting with a random value and each pointing to the next. To generate the next autokey on the key list, the next key ID is the first 32 bits in network byte order of the previous key value. It may happen that a newly generated key ID is less than 65536 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 of that entry, collectively called the autokey values. The list is used in reverse order, so that the first key ID 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 authenticated 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.

The scheme used in client/server mode was suggested by Steve Kent over lunch. The server keeps no state for each client, but uses a fast algorithm and a private value 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 private value. The first 32 bits of the hash is the cookie used for the actual autokey calculation and is returned to the client on request. It is thus specific to each client separately and of no use to other clients or an intruder. A client obtains the cookie and signature using the Autokey protocol and saves it for later use.

In client/server mode the cookie is a relatively weak function of the IP addresses and a server private value. The client uses the cookie and each key ID on the key list in turn to calculate the MAC for the next NTP packet. The server calculates these values and checks the MAC, then generates the MAC for the response using the same values, but with the IP source and destination addresses exchanged. The client calculates and checks the MAC and verifies the key ID matches the 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.

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 temporarily authenticates 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 machine. For this purpose, the broadcast server 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, so that a private cookie can be computed by a strong agreement algorithm. The cookie itself is the first 32 bits of the agreed key. The key list for each direction is generated separately by each peer and used independently, but each is generated with the same cookie.

The server proventic bit is set only when the cookie or autokey values, depending on mode, and the associated timestamp and signature are all valid. If the bit is set, the client processes NTP time values; if the bit is not set, extension field messages are processed in order to run the Autokey protocol, but the NTP time values are ignored. Packets with old timestamps are discarded immediately while avoiding expensive cryptographic algorithms. Bogus packets with newer timestamps must pass the MAC and autokey tests, which is highly unlikely.

Once the proventic bit has been set, the Autokey protocol is normally dormant. In all modes except broadcast server, packets are normally sent without an extension field, 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. This insures that stale values never propagate beyond a clock step.

Public-Key Signatures

Since public-key signatures provide strong protection against misrepresentation of sources, probably the most obvious intruder strategy is to deny or restrict service by replaying old packets with signed cryptographic values in a cut-and-paste attack. The basis values on which the cryptographic operations depend are changed often to deflect brute force cryptanalysis, so the client must be prepared to abandon an old key in favor of a refreshed one. This invites the opportunity for an intruder to clog the client or server by replaying old Autokey messages or to invent bogus new ones. A client receiving such messages might be forced to refresh the correct value from the legitimate server and consume significant processor resources.

In order to foil such attacks, every extension field carries a timestamp in the form of the NTP seconds at the signature time. The signature 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 is synchronized and signatures carry a nonzero (valid) timestamp. Otherwise the system clock is unsynchronized and signatures carry a zero (invalid) timestamp. Extension fields with invalid or old timestamps are discarded before any values are used or signatures verified.

There are three signature types and six values to be signed:

1. **The public value is signed at the time of generation, which occurs** when the system clock is first synchronized and about once per day after that in the reference implementation. Besides the public value, the public key/host name, agreement parameters and leapseconds table are all signed as well, even if their values have not changed. All four of these values carry the same timestamp. On request, each of these values and associated signatures and timestamps are returned in an extension field.

2. **The cookie value is computed and signed upon arrival of a cookie** request message. The response message contains the cookie, signature and timestamp in an extension field.

3. **The autokey values are signed when a new key list is generated, which** occurs about once per hour in the reference implementation. On request, the autokey values, signature and timestamp are returned in an extension

field.

The most recent timestamp for each of the six values is saved for comparison. Once a signature with valid timestamp has been received, packets carrying extension fields with invalid timestamps or older valid timestamps for the same value are discarded before the signature is verified. For packets containing signed extension fields, the timestamp deflects replays that otherwise might consume significant processor resources; for other packets the Autokey protocol deflects message modification and replay. 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 agreement algorithms are regenerated from time to time. It is the intent that file regeneration and loading of these values occur without specific warning and without requiring distribution in advance. While files carrying cryptographic data are not specifically signed, the file names have extensions called filestamps which reliably determine the time of generation. The filestamp for a file is a string of decimal digits representing the NTP seconds at the time the file was created.

As the data are forwarded from server to client, the filestamps are preserved, including those for the public key/host name, agreement parameters and leapseconds table. Packets with older filestamps are discarded befor the signature is verified. Filestamps can in principle be used as a total ordering function to verify that the data are consistent and represent the latest available generation. For this reason, the files should always be generated on a machine when the system clock is valid.

When a client or server initializes, it reads its own public and private key files, which are required for continued operation. Optionally, it reads the agreement parameters file and constructs the public and private values to be used later in the agreement algorithm. Also optionally, it reads the leapseconds table file. When reading these files it checks the filestamps for validity; for instance, all filestamps must be later than the time the UTC timescale was established in 1972.

When the client first validates a proventic source and when the clock is stepped and when new cryptographic values are loaded from a server, the client recomputes all signatures and checks the filestamps for validity and consistency with the signature timestmaps:

1. **If the system clock if valid, all timestamps and filestamps must be** earlier than the current clock time.

2. **All signature timestamps must be later than the public key timestamp.**

3. **In broadcast client mode, the cookie timestamp must be later than the** autokey timestamp.

4. In symmetric modes the autokey timestamp must be later than the public value timestamp.

5. **Timestamps for each cryptographic data type must be later than the** filestamps for that type.

In the above constraints, note that timestamps and filestamps have a granularity of one second, so that a difference of zero seconds is ambiguous. Furthermore, timestamps and filestamps can be in error as much as the value of the synchronization distance; that is, the sum of the root dispersion plus one-half the root delay. However, the NTP protocol normally operates with polling intervals much longer than one second, so that successive timestamps for the same data type are nonambiguous. On most machines, the processor time to generate a complete set of key files is longer than one second, so it is not possible to generate two generations in the same second.

However, it may happen that agreement parameters files may be generated on two machines with the same filestamps, which creates an ordering ambiguity. The filestamps for leapseconds files should also be nonambiguous, since these files are created by NIST not more often than twice per year. While filestamp collisions should be so rare as to be safely ignored, a good management approach might require that these files be generated only on a schedule that guarantees that no more than one client or server generates a new key file set on any one day.

Certificates

PKI principles call for the use of certificates to reliably bind the server distinguished name (host name), public key and related values to each other. The certificate includes these values together with the distinguished name of the certificate atuthority (CA) and other values such as serial number and valid lifetime. These values are then signed by the CA using its private key. The Autokey protocol includes provisions to obtain the certificate, but at the present time does nothing with the values. A future version of the protocol is to include provisions to validate the binding using procedures established by the IETF.

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. These tests have been formulated to harden the protocol against defective header and data values. These are summarized below, since they are an integral component of the NTP cryptograhic defense mechanism. There are eleven tests, called TEST1 through TEST11 in the reference implementation, which are performed in a specific order designed to gain maximum diagnostic information while protecting against accidental or malicious errors.

The tests are divided into three groups. The first group is designed to deflect access control and authentication violations. While access control and message digest violations always result immediate discard, it is necessary when first mobilizing an association to disable the autokey test and certain timestamp tests. However, after the proventic bit is set, all tests are enforced.

The second group of tests is designed to deflect packets from broken or unsynchronized servers and replays. In order to synchronize an association in symmetric modes, it is necessary to save the originate and receive timestamps in order to send them at a later time. This happens for the first packet that arrives, even if it violates the autokey test. In the normal case, the second packet to arrive will be accepted and the association marked reachable. However, an agressive intruder could replay old packets that could disrupt the saved timestamps. This could not result in incorrect time values, but could prevent a legitimate client from synchronizing the association.

The third group of tests is designed to deflect packets with invalid header fields or time values with excessive errors. However, these tests do not directly affect cryptographic source proventication or vulnerability, so are beyond the scope of discussion in this document.

For packets containing signed extension fields additional tests apply, depending on request type. There are the usual tests for valid extension field format, length and values. An instantiated variable, such as the public key/host name, agreement paramaters, public value, cookie or autokey values, is valid when the accompaning timestamp and filestamp are valid. The public key must be instantiated before any signatures can be verified. In symmetric modes the agreement parameters must be instantiated before the public and private agreement values can be determined; the public agreement value must be instantiated before the agreement algorithm can be run to determine the cookie. In all modes the cookie value must be determined before the key list can be generated.

The object of the Autokey dances described below is to set the proventic bit. In client/server mode this bit is set when the cookie is validated. In other modes this bit is set when the autokey values are validated. The bit is cleared initially and when the autokey test fails. If once the bit is set and then cleared, the protocol will send an autokey request message at the next poll opportunity and continue to send this message until receiving valid autokey values or a general reset occurs.

This behavior is a compromise between protocol responsiveness, where the current association can be maintained without interruption, and protocol vulnerability, where an intruder can repeatedly clog the receiver with replays that cause the client to needlessly poll the server and refresh the values.

Error Recovery

The protocol state machine which drives the various Autokey functions includes provisions for various kinds of error conditions that can arise due to missing files, corrupted data, protocol violation 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 zero 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 one. If any bit in this register is one, 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 every poll interval until the authentic response is received or a general reset occurs, in which case the protocol restarts from the beginning. 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 action on an association causes the remaining autokeys on the key list to become invalid. When one of these situations happens, the key list and associated keys 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 client 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 cryptographic values are refreshed, the key lists for all** associations are regenerated.

7. **When the client is first proventicated or the system clock is** stepped, the key lists for all associations are regenerated.

Autokey Protocols

This section describes the Autokey protocols supporting cryptographically secure server proventication. There are three subprotocols, called dances, corresponding to the NTP client/server, broadcast and symmetric active/passive modes. While Autokey messages are piggybacked in NTP packets, the NTP protocol assumes clients poll servers at a relatively low rate, such as once per minute, and where possible avoids large packets. In particular, it is assumed that a request sent at one poll opportunity will normally result in a response before the next poll opportunity.

It is important to observe that, while the Autokey dances are obtaining and validating cryptographic values, the underlying NTP protocol continues to operate. Most packets used during the dances contain signatures, so the values can be believed even before the dance has concluded. Since signatures are valid once the certificate has been validated during the initial steps of the dance, by the time the Autokey values are validated the clock is usually already set. In this way the sometimes intricate Autokey dance interactions do not delay the accumulation of time values that will eventually set the clock. Each autokey dance is designed to be nonintrusive and to require no additional packets other than for regular NTP operations. Therefore, the phrase "some time later" in the descriptions applies to 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. Some requests and most responses are protected by timestamped signatures. The signature covers the data, timestamp and filestamp, where applicable. The timestamp is set to the default (zero) when the sender is not proventicated; otherwise, it is set to the NTP seconds when the signature was generated. The following rules are designed to detect invalid header or data fields and to deflect clogging attacks. Each extension field is validated in the following order and discarded if:

1. **The request or response code is invalid or the data field has** incorrect length.

2. The signature field is either missing or has incorrect length.

3. The public key is missing or has incorrect length.

4. **In the case of the agreement algorithm, the agreement parameterss are** missing or have incorrect lengths.

5. **The signature timestamp is earlier than the last received timestamp** of the same type or the two timestamps are equal and the proventic bit is set..

6. **Where applicable, the filestamp is earlier than the last received** filiestamp of the same type.

Only if the extension field passes all the above tests is the signature verified using PKI algorithms. Otherwise and in general, a response is generated for every request, even if the requestor is not proventicated. However, some responses may have truncated data or signature fields under certain conditions. If these fields are present and have correct length, signatures are present and verifiable.

In the Autokey protocol every transmitted packet is associated with an autokey previously computed and stored in the key list. When the last entry in the list is used, a new list is constructed as described above. This requires knowledge of the cookie value. If for some reason the cookie value is changed, the remaining entries in the key list are purged and a new one constructed. However, if an extension field is present, the current autokey is discarded and the autokey reconstructed using a cookie value of zero.

A timestamp-based agreement protocol is used to manage the distribution of the certificate, agreement parameters and leapseconds table. The association ID request and response messages include the certificate, agreement and leapseconds bits from the system status word. one or more of these bits are set when the associated data are present, either loaded from local files or retrieved from another server at some earlier time. If any of these bits are set in the association ID response to a client in client/server mode or a peer in symmetric mode, the data are requested from the server or peer and, once obtained, the bits are reset. However, the response data are stored only if more recent than the data already stored.

In the descriptions below, it is assumed that the client and server have loaded their own private key and public key, as well as certificate, agreement parameters and leapseconds table, where available. Public keys for other servers, as well as the agreement parameters and leapseconds table, can be loaded from local files or retrieved from any server. Further information on generating and managing these files is in Appendix B.

Preliminaries

The first thing the server needs to do is obtain the system status word, which reveals which cryptographic values the server is prepared to offer, and then the public key and certificate. These steps are independent of which mode the server is operating in - client/server, broadcast or symmetric modes.

The following pseudo-code describes the client state machine operations. Note that the packet can one request and one or more responses. The machine requires the association ID, public key and optional certificate, in that order. While not further specified in this memorandum, an optional parameter request message can be used to negotiate algorithm identifiers, parameters and alternate identification values. Note that the association ID response message also contains the system status word, which contains the certificate bit.

```
        if (response_pending)
                send_response;
        if (!parameters)
                request_parameters;
        if (!association_ID)
                request_association_ID;
        else if (!public_key)
                request_public_key;
        else if (certificate_bit)
                request_certificate;
```

The following diagram shows the preliminary protocol dance. In this and following diagrams the NTP packet type is shown above the arrow and the extension field(s) message type shown below. Note that in the client/server mode the server responds immediately to the request, but in the symmetric modes the response may be delayed for a period up to the current poll interval. The following cryptographic values are instantiated by the dance:

public key	server public key
host name	server host name
CA name	certificate authority host name (optional)
filestamp	generation time of public key file
secure bit	set when the public key is stored and validated

```
  server             client
    |                  |
    |    NTP client    |
  1 |<-----------------| mobilize client association; generate key list
    |   assoc ID req   | with default cookie; send status word
    |                  |
    |    NTP server    |
  2 |----------------->| store status word
    |   assoc ID rsp   |
    |                  |
    |    NTP client    |
  3 |<-----------------| request public key and host name
    |   key/name req   |
    |                  |
    |    NTP server    |
  4 |----------------->| store public key, host name, filestamp and
    |   key/name rsp   | timestamp
    |       ...        |
    |                  |
    |    NTP client    |
  5 |<-----------------| request certificate
    |    certif req    |
    |                  |
    |    NTP server    |
  6 |----------------->| store certificate; verify credentials; set
    |    certif rsp    | secure bit
    |       ...        |
```

The dance begins when the client (or symmetric-active peer) on the right mobilizes an association, generates a key list using the default cookie and sends an association ID request message (1) to the server (or symmetric-passive peer) on the left. The server responds with an association ID response message (2) including the server association ID and status word. To protect against a clogging attack, the transmit timestamp in the NTP header in the request must be identical to the originate timestamp in the response. The client retransmits request (1) at every poll opportunity until receiving a valid response (2) or association timeout.

Some time later the client sends a public key/host name request (3) to the server. The server responds with the requested data and associated timestamp and filestamp (4). The client checks the timestamp and filestamp, verifies the signature and initializes the public key and host name. If the certificate bit in the status word is zero, indicating the server is not prepared to send one, and if the client concurs, the secure bit is set at this time and the certificate exchange is bypassed. The client retransmits request (3) at every poll opportunity until receiving a valid response (4) or association timeout.

The public key/host name message can be interpreted as a poor-man's certificate, since it is signed and timestamped. However, strong security requires a CA sign the host name and public key values and establish a period of validity for the signature. As an optional feature, the client sends a certificate request (5) to the server. The server responds with the requested data and assciated timestamp and filestamp (6). The response is signed by the CA's public key, so a further step may be necessary to obtain the CA's certificate, which contains its public key. The details for these additional steps are for further study.

Since (4) is the first signed message received, the timestamp and filestamp have only marginal utility, but do serve to avoid messages from unsynchronized servers and deflect replays. The interesting question is whether to provide automatic update when the server makes a new key generation, since the new generation would have a later filestamp and instantly deprecate all cryptographic values with earlier timestamps. This brings up the question of a distributed greeting protocol, which may be a topic for future study. Meanwhile, the reference implementation accepts only the first message received and discards all others.

When the secure bit is set, data in packets with signatures are valid and the NTP protocol continues in parallel with the Autokey protocol.

Client/Server Modes (3/4)

In client/server modes the server keeps no state variables specific to each of possibly very many clients and mobilizes no associations. The server regenerates a cookie for each packet received from the client. For this purpose, the server hashes the cookie from the IP addresses and private value with the key ID field set to zero, as described previously, then provides it to the client. Both the client and server use the cookie to generate the autokey which validates each packet received. To further strengthen the validation process, the client selects a new key ID for every packet and verifies that it matches the key ID in the server response to that packet.

Before proceeding to the full protocol description, it should be noted that in the case of lightweight SNTP protocol associations, it is not necessary to proceed beyond the preliminary protocol defined above. Most if not all SNTP implementations send only a single client-mode packet and expect only a single NTP server-mode packet in return. Since the Autokey protocol is piggybacked in the NTP packet, the clock can be set and the server authenticated with a single packet exchange if a certificate is not required and in two exchanges if it is. Details of this simplified protocol remain to be determined.

The following pseudo-code describes the client state machine operations. The machine requires the association ID, public key, optional certificate, cookie, autokey values and leapseconds table in that order, but the autokey values are required only if broadcast client mode.

```
        if (response_pending)
                send_response;
        if (!cookie)
                request_cookie;
        else if (!autokey_values && broadcast_client))
                request_autokey_values;
        else if (!leapseconds_table)
                request_leapseconds_table;
```

The following diagram shows the protocol dance in client/server mode. The following cryptographic values are instantiated by the dance:

public key	server public key
host name	server host name
filestamp	generation time of public key file
timestamp	signature time of public key/host name values
cookie	cookie determined by the server for this client

timestamp   signature time of cookie
proventic bit   set when client clock is synchronized to source

server   client

```
    |                  |
    |    NTP client    |
  7 |<-----------------| request cookie
    |    cookie req    |
    |                  |
    |    NTP server    |
  8 |----------------->| store cookie and timestamp; set proventic bit;
    |    cookie rsp    |
    |       ...        |
    |                  |
    |    NTP client    |
  9 |<-----------------| regenerate key list with server cookie
    |                  |
    |    NTP server    |
 10 |----------------->|
    |                  |
    |     continue     |
    =  client/server   =
```

The dance begins when the client on the right mobilizes an association and validates the public key as in the preliminary dance above. Some time later the client sends a cookie request (7). The server immediately responds with the cookie and timestamp (8). The client checks the timestamp, verifies the signature and initializes the cookie and cookie timestamp, then sets the proventic bit. Since the cookie has changed, the client regenerates the key list with this cookie when the next packet is sent. The client retransmits request (7) at every poll opportunity until receiving a valid response (8) or association timeout.

After successful verification, there is no further need for extension fields, unless an error occurs or the server generates a new private value. When this happens, the server fails to authenticate packet (9) and, following the original NTP protocol, responds with a NAK packet (10), which the client ignores. Eventually, an association timeout and general reset occurs and the dance restarts from the beginning. Of course, the NAK client could interpret the NAK message to restart the protocol immediately and avoid the timeout. However, this invites the opportunity for an intruder to destabilize the state machine with spurious NAK messages.

Broadcast Mode (5)

In broadcast mode, packets are always sent with an extension field. Since the autokey values for these packets use a well known default cookie (zero), they can in principle be remanufactured with a new MAC acceptable to the receiver; however, the key list provides the authentication function as described earlier. The broadcast server keeps no state variables specific to each of possibly very many clients and mobilizes no associations for them.

The following pseudo-code describes the broadcast server state machine operations. Each broadcast packet includes one response message containing either the signed autokey values, if the first autokey on the key list, or the association ID and status word otherwise. Note however, when a broadcast client first comes up, the state machine also responds to client requests as in client/server mode without affecting the broadcast packets. Note that the association ID request and response messages also contain the system status word.

```
        if (new_list)
                send_autokey_values;

        else
                send_association_ID;
```

The server on the left in the diagram below sends packets that are received by each of a possibly large number of clients, one of which is shown on the right. Ordinarily, clients do not send packets to the server, except to calibrate the propagation delay and to obtain cryptographic values such as the cookie and autokey values. The following diagram shows the protocol dance in broadcast mode. The following cryptographic values are instantiated by the dance:

public key	server public key
host name	server host name
filestamp	generation time of public key file
timestamp	signature time of public key/host name values
cookie	cookie determined by the server for this client
timestamp	signature time of cookie
autokey values	initial key ID, initial autokey
timestamp	signature time of autokey values
proventic bit	set when client clock is synchronized to source

```
  server              client
    |                  |
    |   NTP broadcast  |
  1 |----------------->| mobilize broadcast client association; set
    |   assoc ID rsp   | initially to operate in client/server mode
    |                  |
    |       ...        | continue as in preliminary protocol above
    |                  |
    |    NTP client    |
  7 |<-----------------| request cookie
    |    cookie req    |
    |                  |
    |    NTP server    |
  8 |----------------->| store cookie and timestamp
    |    cookie rsp    |
    |       ...        |
    |                  |
    |    NTP client    |
  9 |<-----------------| regenerate key list with server cookie
    |    autokey req   |
    |                  |
    |    NTP server    |
 10 |----------------->| store autokey values and timestamp; set
    |    autokey rsp   | proventic bit
    |       ...        |
    |                  |
    |    NTP client    |
    |<-----------------| continue to accumulate time values
    |                  |
    |    NTP server    |
    |----------------->|
    |                  |
    |     continue     |
    =      volley      =
    |                  |
    |    NTP client    |
    |<-----------------|
    |                  |
    |    NTP server    |
    |----------------->| set clock and propagation estimate; discard
    |                  | remaining keys; switch to broadcast client mode
    |     continue     |
    =     broadcast    =
    |                  |
    |   NTP broadcast  |
    |----------------->| server rolls new key list; client refreshes
    |    autokey rsp   | autokey values
    |                  |
    =                  =
```

The server sends broadcast packets (1) continuously at intervals of about one minute using the key list and regenerating the list as required. The first packet sent after regenerating the list includes the autokey values and signature; other packets include only the association ID and status word.

The dance begins when the client on the right receives a broadcast message (1). It mobilizes a broadcast client association set initially to operate in client/server mode. It then continues to operate as in the prelimiary protocol to obtain and validate the public key and host name values. However, the client does not initiate the dance until some time later (to avoid implosion at the server). However, in addition to the status word, the association ID response includes the association ID of the server, so the correct association, if more than one, can be identified.

Some time later the client sends a cookie request (7). The server immediately responds with the requested value (8). The client checks the timestamp, verifies the signature and initializes the cookie and cookie timestamp. Since the cookie has changed, the client regenerates the key list with this cookie when the next packet is sent. The client retransmits request (7) at every poll opportunity until receiving a valid response (8) or association timeout.

If an autokey response happens to be in one of the server packets (1), the client has stored the autokey values and autokey timestamp, so can switch immediately to broadcast client mode and send no further packets. Otherwise, some time later the client sends an autokey request (9). The server immediately responds with the values (10). The client checks the timestamp, verifies the signature and initializes the autokey values and autokey timestamp and sets the proventic bit. The client retransmits packet (9) until receiving a valid response (10) or association timeout.

After successful verification, there is no further need for extension fields and the client can switch to broadcast client mode and send no additional packets. However, it is the usual practice to send additional client/server packets in order for the client mitigation algorithms to refine the clock offset/delay estimates. When a sufficient number of estimates are available, the client discards the cookie and remaining keys on the key list, switches to broadcast client mode, calculates the propagation delay and sets the clock.

When the server regenerates the key list, it sends an autokey response in the first packet, which allows the clients to validate it and reset the autokey values. Unless this packet happens to be lost, the clients can continue with no further interaction with the server. Otherwise, the client fails to authenticate the packets (1). Eventually, an association timeout and general reset occurs and the dance restarts from the beginning.

Symmetric Active/Passive Mode (1/2)

In symmetric modes there is no explicit client/server relationship, since each peer in the relationship can operate as a server with the other operating as a client. Which peer acts as the server depends on which peer has the smallest root synchronization distance to its ultimate reference source, and the choice may change from time to time. This requirement results in a quite complex interaction between the peers, especially when considering the many possibilities of failure and recovery.

There are two protocol scenarios involving symmetric modes. The simplest scenario is where both peers have configured associations that operate continuously in symmetric active mode and cryptographic values such as the public key/host name, certificate, agreement parameters and public value can be configured in advance. A more interesting scenario is when a symmetric active peer with a configured association begins operation with a symmetric-passive peer initially without such an association.

The following pseudo-code describes the symmetric state machine operations. Note that the packet can contain one request and one or two responses. The machine requires the association ID, public key, certificate, agreement parameters, agreement public value, autokey values and leapseconds table in that order. There is a provision to send the current autokey values when the peer has not requested them. This happens when a peer first proventicates and recomputes the key list using the agreed cookie.

```
        if (response_pending)
                send_response;
        if (!agreement_parameters)
                request_agreement_parameters;
        else if (!agreement)
                send_agreement;
        else if (!autokey_values)
                request_autokey_values;
        else if (!new_list)
                send_autokey_values;
        else if (!leapseconds_table)
                request_leapseconds_table;
```

The following diagrams show the protocol dance in symmetric active/passive mode. The dance in symmetric active/active mode is much simpler and similar to two independent client/server modes, one for each direction, but with the cookie requests replaced by an agreement algorithm. Note that in the following the NTP client header is replaced by the NTP symmetric active header and the NTP server header is replaced by the NTP symmetric passive header. The following cryptographic values are instantiated by each peer in the dance:

public key	server public key
host name	server host name
filestamp	generation time of public key file
timestamp	signature time of public key/host name values
cookie	cookie determined by the agreement algorithm
timestamp	signature time of cookie
autokey values	initial key ID, initial autokey
timestamp	signature time of autokey values
proventic bit	set when client clock is synchronized to source

```
 passive             active
    |                  |
    |    NTP active    |
  1 |<-----------------| mobilize symmetric active association; generate
    |   assocID req    | key list with default cookie; send status word
    |                  |
    |       ...        | continue as in preliminary protocol above
    |                  |
    |    NTP passive   |
  2 |----------------->| store status word
```

```
    |   assoc ID rsp   |
    |                  |
    |    NTP active    |
  1 |<-----------------| generate key list with default cookie; request
    |   key/name req   | passive key/name
    |       ...        |
    |                  |
    |    NTP passive   |
  2 |----------------->| verify passive credentials
    |   key/name rsp   |
    |   key/name req   |
    |       ...        |
    |                  |
    |    NTP active    |
  3 |<-----------------| send active key/name; request agreement
    |   key/name rsp   | parameters
    |     param req    |
    |       ...        |
    |                  |
    |    NTP passive   |
  4 |----------------->| store agreement parameters;  and timestamp; set
    |     param rsp    | proventic bit
    |     agree rsp    |
    |       ...        |
    |                  |
    |    NTP active    |
  3 |<-----------------| send active key/name; request agreement
    |   key/name rsp   | parameters
    |     param req    |
    |       ...        |
    |                  |
    |    NTP passive   |
  4 |----------------->| store autokey values and timestamp; set
    |   key/name req   | proventic bit
    |    autokey rsp   |
    |       ...        |
    |                  |
    |    NTP active    |
  5 |<-----------------| continue to accumulate time values
    |   key/name rsp   |
    |                  |
    =     continue     =
    |                  |
    |    NTP passive   |
  6 |----------------->| set clock
    |   key/name req   |
    |                  |
    |  continue below  |
    =                  =
```

The dance begins when the active peer on the right generates a key list with default cookie and timestamp and sends a public key/host name request to the passive peer on the left (1). The passive peer checks its access control list and (optionally) queries the DNS using the server IP address to obtain related cryptographic values. If successful, the peer mobilizes an association in symmetric passive mode, but takes no further action until the next poll interval, as required by the NTP protocol. From this point the passive peer responds to requests, but otherwise ignores all time values until the active peer has set its clock and can provide valid timestamps.

Some time later the passive peer generates a key list with default cookie and timestamp and sends its public key/host name values along with a request for the public key/host name values of the active peer (2). Subsequently, the active peer sends these values, but they are ignored since the timestamps are invalid. Meanwhile, the active peer checks the timestamp, verifies the signature and initializes the public key/host name values, filestamp and timestamp. The active peer retransmits request (1) at every poll opportunity until receiving a valid response (2) or until association timeout.

Some time later the active peer sends the requested public key/host name values along with an autokey request (3). The passive peer retransmits request (2) at every poll opportunity until receiving a valid timestamp and verified signature or until association timeout. Since the cookies for each peer already have a common value and the active peer is unsynchronized, it is pointless to run the agreement algorithm.

Some time later the passive peer sends the requested autokey values (4). The active peer checks the timestamp, verifies the signature and initializes the autokey values and timestamp and sets the proventic bit. At this point the active peer has authenticated the passive peer, but may not have accumulated sufficient time values to set the clock and provide valid timestamps. Operation continues in rounds where the passive peer requests the public key/host name values and the active peer returns them, but the passive peer ignores them. Eventually, the active peer accumulates sufficient time values to set the clock. While the cookie has not changed, the timestamp has, so the key list is regenerated with the default key (strictly speaking, only the signature needs to be recomputed). The active peer is now proventicated, but the passive peer has not yet authenticated the active peer.

Some understanding of the tricky actions to follow can be gained from the observation that, up until this point every message received by the active peer had a signed response field, so that the cookie value is the default. However, at this point the active peer has all the cryptographic means at hand and does not need to request anything further from the passive peer. Thus, the passive peer sends nothing but requests and these are not signed or timestamped. Since the cryptograhic security relies entirely on the autokey test, it is important that both peers generate key lists with the same cookie.

The steps now taken are shown below with the active peer on the left and the passive peer on the right.

```
  active            passive
```

```
    |                  |
    |    NTP active    |
  1 |----------------->| validate active peer, compute agreed key,
    |   key/name rsp   | regenerate key list with peer key
    |    public req    |
    |                  |
    |    NTP passive   |
  2 |<-----------------| active computes agreed key, regenerates key
    |    public rsp    | list with agreed key
    |    autokey req   |
    |       ...        |
    |                  |
    |    NTP active    |
  3 |----------------->| set authentic
    |    autokey rsp   |
    |    autokey req   |
    |       ...        |
    |                  |
    |    NTP passive   |
  4 |<-----------------|
    |    autokey rsp   |
    |       ...        |
    |                  |
    |    NTP active    |
  5 |----------------->| regular operation (no extension fields)
    |       ...        |
    |                  |
    |    NTP passive   |
  6 |<-----------------|
    |                  |
    |     continue     |
    =  active/passive  =
```

The agreement parameters must have been previously obtained by at least one of the peers, either directly from a file or indirectly from another server running the Autokey protocol. A peer needing the parameters sends an agreement parameters request to the other peer and that peer responds with the requested data. This exchange, along with the leapseconds table exchange, is similar to the public key/host name exchange, but not shown here.

Once the proventic bit is set, the next message sent by the active peer contains the public key/host name requested by the passive peer, but now with valid timestamp, plus a public value request containing the active peer public value (1). The passive peer checks the public key/host name filestamp and timestamp, verifies the signature and initializes the values. Optionally, it checks its access control list and queries the DNS using the server IP address to obtain related cryptographic values. Conceivably, the active peer could be found bogus at this time; what to do in this case is for further study.

The passive peer next checks the public value request timestamp, verifies the signature and runs the agreement algorithm to construct the shared cookie. Since the cookie has changed, the peer regenerates the key list with this cookie when the next packet is sent.

Some time later the passive peer sends a public value response including its own public value together with an autokey request (2). The active peer checks the timestamp, verifies the signature and runs the agreement algorithm to construct the shared cookie. Since the cookie has changed, the peer regenerates the key list with this cookie when the next packet is sent. The active peer retransmits the public value request (only) (1) at every poll opportunity until receiving a valid response (2) or association timeout.

Some time later the active peer sends its autokey values as requested together with an autokey request (3). The passive peer checks the timestamp, verifies the signature, initializes the autokey values and sets its proventic bit. The passive peer retransmits request (2) at every poll opportunity until receiving a valid response (3) or association timeout.

Some time later the passive peer sends its autokey values as requested (4). The active peer checks the timestamp, verifies the signature, and initializes the autokey values (the proventic bit is already set). The active retransmits the autokey request (only) (3) until receiving a valid response (4) or association timeout.

At this point both peers have completed the Autokey dance and each is authenticated to the other. However, note that the NTP rules require a peer operating at a lower stratum disregards time values from a hither stratum peer; so, while the peers continue to exchange time values, the values will not be used unless the passive server for some reason loses its synchronization source.

After successful authentication, there is no further need for extension fields, unless an error occurs or one of the peers generates new public values. The protocol requires that, if a peer receives a public value resulting in a different cookie, it must send its own public value. Since the autokey values are included in an extension field when a new key list is generated, there is ordinarily no need to request these values, unless one or the other peer restarts the protocol or the packet containing the autokey values is lost. Eventually, an association timeout and general reset occurs and the dance restarts from the beginning.

Security Analysis

This section discusses the most obvious security vulnerabilities in the various modes and phases of operation. Throughout the discussion the cryptographic algorithms themselves are assumed secure; that is, a successful brute force attack on the algorithms or public/private keys or agreement values is unlikely. However, vulnerabilities remain in the way the actual cryptographic data, including the cookie and autokey values, are computed and used.

While the protocol has not been subjected to a formal analysis, a few preliminary observations are warranted. 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 will start from the beginning. A general reset is performed on all associations when the clock is first set and when it is stepped after that. This purges all cryptographic values and time values dependent on unproventicated sources.

The first exchange in all protocol modes involves an association ID request and response cycle. Bits in the server status word indicate whether the server has the agreement paramters and/or leapseconds table. The association ID messages are not protected by a signature, so presumably an intruder can manufacture fake bits causing a client livelock or deadlock condition. To protect against this vulnerability, the transmit timestamp of the request is matched against the originate timestamp of the response. The response is accepted only if the two values match. An intruder is unlikely to predict the transmit timestamp, which in this case is an effective nonce.

Once the clock is set, and except for the special cases summarized below, no old or duplicate values will be accepted in any state and an intruder cannot induce a clogging attack, since the MAC, autokey and timestamp tests will discard packets before a clogging vulnerability is exposed. While significant vulnerabilities exist during the initial protocol states while the necessary values are being obtained, the most an intruder can do is prevent the protocol dance from completing. If it does complete, it must complete correctly.

The cryptographic values are always obtained in the same order and in the same order as the dependency relationships between them. No cryptographic variables or time variables are instantiated unless the server is proventic and proventicated. The public key and host name must be obtained first and no other messages are accepted until they have been obtained. The cookie must be obtained before the autokey values that depend on them, etc. Finally, in symmetric modes, both peers obtain cryptographic values in the same order, so deadlock cannot occur.

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.

There are three tiers of defense against hostile intruder interference.

The first is the message authentication code (MAC) based on a keyed message digest or autokey generated as the hash of the IP address fields, key ID field and a special cookie, which can be public or the result of an agreement algorithm. If the message digest computed by the client does not match the value in the MAC, either the autokey used a different cookie than the server or the packet was modified by an intruder. Packets that fail this test are discarded without further processing; in particular, without spending processor cycles on expensive public-key algorithms.

The second tier of defense involves the key list, which is generated as a repeated hash of autokeys and used in the reverse order. While any receiver can authenticate a message by hashing to match a previous key ID, as a practical matter an intruder cannot predict the next key ID and thus cannot spoof a packet acceptable to the client. In addition, tedious hashing operations provoked by replays of old packets are suppressed because of the basic NTP protocol design. Finally, spurious public-key computations provoked by replays of old packets with extension fields are suppressed because of the signature timestamp check.

The third tier of defense is represented by the Autokey protocol and extension fields with timestamped signatures. The signatures are used to reliably bind the autokey values to the private key of a trusted server. Once these values are instantiated, the key list authenticates each packet relative to its predecessors and by induction to the instantiated autokey values.

In addition to the three-tier defense strategy, all packets are protected by the NTP sanity checks. Since all packets carry time values, replays of old or bogus packets can be deflected once the client has synchronized to proventic sources. However, the NTP sanity checks are only effective once the packet has passed all cryptographic tests. This is why the signature timestamp is necessary to avoid expensive calculations that might be provoked by replays. Since the signature and verify operations have a high manufacturing cost, in all except client/server modes the protocol design protects against a clogging attack by signing cryptographic values only when they are created or changed and not on request.

Specific Attacks

While the above arguments suggest that the vulnerability of the Autokey protocols to cryptanalysis is suitably hard, the same cannot be said about the vulnerability to a replay or clogging attack, especially when a client is first mobilized and has not yet proventicated. In the following discussion a clogging attack is considered a replay attack at high speed which can clog the network and deny service to other network users or clog the processor and deny service to other users on the same machine. While a clogging attack can be concentrated on any function or algorithm of the Autokey protocol, the must vulnerable target is the public key routines to sign and verify public values. It is vital to shield these routines from a clogging attack.

In all modes the cryptographic seed data used to generate cookies and autokey values are changed from time to time. Thus, a determined intruder could save old request and response packets containing these values and replay them before or after the seed data have changed. Once the client has proventicated, the client will detect replays due to the old timestamp and discard the data. This is why the timestamp test is done first and before the signature is computed. However, before this happens, the client is vulnerable to replays whether or not they result in clogging.

There are two vulnerabilities exposed in the protocol design: a sign attack where the intruder hopes to clog the victim with needless signature computations, and a verify attack where the intruder attempts to clog the victim with needless verification computations. The reference implementation uses the RSA public key algorithms for both sign and verify functions 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 agreement values are regenerated, which happens about once per day. However, a server is vulnerable to a sign attack where the intruder can clog the server with cookie-request messages. The protocol design precludes server state variables stored on behalf of any client, so the signature must be recomputed for every cookie request. Ordinarily, cookie requests are seldom used, except when the private values are regenerated. However, a determined intruder could replay intercepted cookie requests at high rate, which may very well clog the server. There appears no easy countermeasure for this particular attack.

The intruder might be more successful with a verify attack. Once the client has proventicated, replays are detected and discarded before the signature is verified. However, if the cookie is known or compromised, the intruder can replace the timestamp in an old message with one in the future and construct a packet with a MAC acceptable to a client, even if it has bogus signature and incorrect autokey sequence. The packet passes the MAC test, but then tricks the client to verify the signature, which of course fails. What makes this kind of attack more serious is the fact that the cookie used when extension fields are present is well known (zero). Since all broadcast packets have an extension field, all the intruder has to do is clog the clients with responses including timestamps in the future. Assuming the intruder has joined the NTP broadcast group, the attack could clog all other members of the group. This attack can be deflected by the autokey test, which in the reference implementation is after extension field processing, but this requires very intricate protocol engineering and is left for a future refinement.

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 the special NAK packet. A naive client might conclude the server had refreshed its private value and so attempt to refresh the server cookie using a cookie-request message. This results in the server and client burning spurious machine cycles and invites a clogging attack. This is why the reference implementation simply discards all protocol and procedure errors and waits for timeout in order to refresh the values. However, a more clever client may notice that the NTP originate timestamp does not match the most recent client packet sent, so can discard the bogus NAK immediately.

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 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 earlier in this memorandum, 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.

Present Status and Unifinished Business

The Autokey protocol described in this memorandu has been implemented in the public software distribution for NTP Version 4 and has been tested in machines of either endian persuasion and both 32- and 64-bit architectures and kernels. Testing the implementation has been complicated by the many combinations of modes and failure/recovery mechanisms, including daemon restarts, key expiration, communication failures and various management mistakes. The experience points up the fact that many little gotchas that are survivable in ordinary protocol designs become showstoppers when strong cryptographic assurance is required.

The analysis, design and implementation of the Autokey protocol is basically mature; however, There are several remaining implementation issues. One has to do with cryptographic parameter negotiation, as in IPSEC protocols such as Photuris. As with Photuris, there may be a need to offer and agree to one of possibly several hashing algorithms, signature algorithms and agreement algorithms. A message type has been defined for this purpose, but its syntax and semantics remain to be provoked.

Another issue is support for certificates and certificate authorities, in particular Secure DNS services. In the reference implementation a complicating factor is the existing banal state of the configuration and resolver code. Over the years this code has sprouted to a fractal-like state where possibly the only correct repair is a complete rewrite.

Appendix A. Packet Formats

The NTP Version 4 packet consists of a number of fields made up of 32-bit (4 octet) words. 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 bits.

```
                     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 (64)                    |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                   Originate Timestamp (64)                    |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                    Receive Timestamp (64)                     |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                    Transmit Timestamp (64)                    |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                      Extension Field(s)                       =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Key ID                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
|                      Message Digest (128)                     |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 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. While this algorithm and mode of operation is supported in NTP Version 4, the DES algorithm has been removed from the standard software distribution and must be obtained via other sources. The preferred replacement for NTP Version 4 is the Message Digest 5 (MD5) algorithm, which is included in the distribution. The Message Digest field is 64 bits for DES-CBC and 128 bits for MD5, while the Key ID field is 32 bits for either algorithm.

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. Each extension field contains a request or response message, which consists of a 16-bit length field, an 8-bit control field, an 8-bit flags field and a variable length data field, all in network byte order:

```
                     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            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
=                              Data                             =
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 Version field identifies the version number of the extension field protocol; this memorandum specifies version 1. The Code field specifies the operation in request and response messages. The length includes all octets in the extension field, including the length field itself. Each extension field is rounded up to the next multiple of 4 **octets and the last field rounded up to the next multiple of 8 octets.** The extension fields can occur in any order; however, in some cases there is a preferred order which improves the protocol efficiency.

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 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 12 (3 words), a MAC (DES-CBC) is present, but no extension field; if 20 (5 words), a MAC (MD5) is present, but no extension field;

If the length is 8 (2 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 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 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. However, it is not a protocol error to send an unsolicited response with no matching request.

There are currently five request and six response messages. All request messages except the Association ID request message have 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     |      Code     |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Association ID                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Association ID field is used to match a client request to a particular server association. By convention, servers set the association ID in the response and clients include the same value in requests. Also by convention, until a client has received a response from a server, the client sets the Association ID field to 0. 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.

The following request and response messages have been defined.

Parameter Negotiation (1)

This extension field is reserved for future use as an algorithm and algorithm parameter offer/select exchange, as well as to provide the optional identification value to use in lieu of endpoint IP addresses when calculating the autokey. The format, encoding and use of these data remain to be specified. The command code is reserved.

Association ID (2)

A client sends the request to obtain the association ID and status flags. A broadcast server sends an unsolicited response for all except the first autokey sent from the key list. The request and response have the following format (except for the response bit):

```
                     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|E|     1     |       2       |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Flags                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Association ID field contains the association ID of the server. The status flags currently defined are

Bit	Function
31	**autokey is enabled**
30	**public and private keys have been loaded**
29	**agreement parameters have been loaded**
28	**leapseconds table has been loaded**

Additional bits may be defined in future, so for now bits 0-27 should be set to zero. There is no timestamp or signature associated with this message.

Autokey (3)

A broadcast server or symmetric peer sends the request to obtain the autokey values. The response 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     |       4       |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Initial Sequence                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Initial Key ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Signature Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                           Signature                           =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The response is also sent unsolicited when the server or peer generates a new key list. The Initial Sequence field contains the first key number in the current key list and the Initial Key ID field contains the next key ID associated with that number. If the server is not synchronized to a proventicated source, the Timestamp field contains 0; otherwise, it contains the NTP seconds when the key list was generated and signed. The signature covers all fields from the Timestamp field through the Initial Key ID field. If for some reason these values are unavailable or the signing operation fails, the Initial Sequence and Initial Key ID fields contain 0 and the extension field is truncated following the Initial Key ID field.

Cookie (4)

A client sends the request to obtain the server cookie. The response 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     |       3       |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Cookie                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Signature Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                           Signature                           =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Since there is no server association matching the client, the association ID field for the request and response is 0. The Cookie field contains the cookie used in client/server modes. If the server is not synchronized to a proventicated source, the Timestamp field contains 0; otherwise, it contains the NTP seconds when the cookie was computed and signed. The signature covers the Timestamp and Cookie fields. If for some reason the cookie value is unavailable or the signing operation fails, the Cookie field contains 0 and the extension field is truncated following this field.

Diffie-Hellman Parameters (5)

A symmetric peer uses the request and response to send the public value and signature to its peer. The response 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     |       5       |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Parameters Filestamp                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Parameters Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                           Parameters                          =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Signature Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                           Signature                           =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Parameters field contains the Diffie-Hellman parameters used to compute the public and private values. The Parameters Filestamp field contains the NTP seconds when the Diffie-Hellman parameter file was generated. If the server is not synchronized to a proventicated source, the Timestamp field contains 0; otherwise, it contains the NTP seconds when the public value was generated and signed. The signature covers the Timestamp, Parameters Length and Parameters fields. If for some reason these values are unavailable or the signing operation fails, the Parameters Length field contains 0 and the extension field is truncated following this field.

Public Value (6)

A symmetric peer uses the request and response to send the public value and signature to its peer. The response 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     |       5       |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Filestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Public Value Length                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                          Public Value                         =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Signature Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                           Signature                           =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Public Value field contains the Diffie-Hellman public value used to compute the agreed key.

The Filestamp field contains the NTP seconds when the Diffie-Hellman parameter file was generated. If the server is not synchronized to a proventicated source, the Timestamp field contains 0; otherwise, it contains the NTP seconds when the public value was generated and signed. The signature covers all fields from the Timestamp field through the Public Value field. If for some reason these values are unavailable or the signing operation fails, the Public Value Length field contains 0 and the extension field is truncated following this field.

Public Key/Host Name (7)

A client uses the request to retrieve the public key, host name and signature. The response 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     |       7       |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Public Key ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Filestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Public Key Length                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                          Public Key                           =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Host Name Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                           Host Name                           =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Signature Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                           Signature                           =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Since the public key and host name are a property of the server and not any particular association, the association ID field for the request and response is 0. The Public Key field contains the RSA public key in rsaref2.0 format; that is, the modulus length (in bits) as the first word followed by the modulus bits. Note that in some architectures the rsaref2.0 modulus word may be something other than 32 bits. The Host Name field contains the host name string returned by the Unix gethostname() library function.

The Filestamp field contains the NTP seconds when the public/private key files were generated. If the server is not synchronized to a proventicated source, the Timestamp field contains 0; otherwise, it contains the NTP seconds when the public value was generated and signed. The signature covers all fields from the Timestamp field through the Host Name field. If for some reason these values are unavailable or the signing operation fails, the Host Name Length field contains 0 and the extension field is truncated following this field.

Leapseconds table (8)

The civil timescale (UTC), which is based on Earth rotation, has been diverging from atomic time (TAI), which is based on an ensemble of cesium oscillators, at about one second per year. Since 1972 the International Bureau of Weights and Measures (BIPM) declares on occasion a leap second to be inserted in the UTC timescale on the last day of June or December. Sometimes it is necessary to correct UTC as disseminated by NTP to agree with TAI on the current or some previous epoch.

A client uses the request to retrieve the leapseconds table and signature. The response 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     |       8       |           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Public Key ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Filestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Leapseconds table Length                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                        Leapseconds table                       =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Signature Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                           Signature                           =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The NTP extension field format consists of a table with one entry in NTP seconds for each leap second.

Since the leapseconds table is a property of the server and not any particular association, the association ID field for the request and response is 0. The Leapseconds table field contains a list of the historic epoches that leap seconds were inserted in the UTC timescale. Each list entry is a 32-bit word in NTP seconds, while the table is in order from the most recent to the oldest insertion. At the first insertion in January, 1972 UTC was ahead of TAI by 10 s and has increased by 1 s for each insertion since then. Thus, the table length in bytes divided by four plus nine is the current offset of UTC relative to TAI.

The Filestamp field contains the NTP seconds when the leapseconds table was generated at the original host, in this case one of the public time servers operated by NIST. If the value of the filestamp is less than the first entry on the list, the first entry is the epoch of the predicted next leap insertion. The filestamp must always be greater than the second entry in the list. If the server is not synchronized to a proventicated source, the Timestamp field contains 0; otherwise, it contains the NTP seconds when the public value was generated and signed. The signature covers all fields from the Timestamp field through the Leapseconds table field. If for some reason these values are unavailable or the signing operation fails, the Host Name Length field contains 0 and the extension field is truncated following this field.

Appendix B. Key Generation and Management

In the reference implementation the lifetimes of various cryptographic values are carefully managed and frequently refreshed. While permanent keys have lifetimes that expire only when manually revoked, autokeys have a lifetime specified at the time of generation. When generating a key list for an association, the lifetime of each autokey is set to expire one poll interval later than it is scheduled to be used. Ordinarily, key lists are regenerated and signed about once per hour and private cookie values and public agreement values are refreshed and signed about once per day. The protocol design is specially tailored to make a smooth transition when these values are refreshed and to avoid vulnerabilities due to clogging and replay attacks while refreshment is in progress.

Autokey key management can be handled in much the same way as in the ssh facility. A set of public and private keys and agreement parameters are generated by a utility program designed for this purpose. The program generates four files, one containing random DES/MD5 private keys, which are not used in the Autokey protocol, a second containing the RSA private key, a third the RSA public key, and a fourth the Diffie-Hellman agreement parameters. In addition, the leapseconds table is generated and stored in public time servers maintained by NIST. The means to do this are beyond the scope of this memorandum.

All files are based on random strings seeded by the system clock at the time of generation and are in printable ASCII format with PEM (base-64) encoding. The name of each file includes an extension consisting of the NTP seconds at the time of generation. This is interpreted as a key ID in order to detect incorrect keys and to handle key changeovers in an orderly way. In the recommended method, all files except the RSA private key file are installed in a shared directory /usr/local/etc, which is where the daemon looks for them by default. The private RSA key file is installed in an unshared directory such as /etc. It is convenient to install links from the default file names, which do not have filestamp extensions, to the current files, which do. This way when a new generation of keys is installed, only the links need to be changed.

When a server or client first initializes, it loads the RSA public and private key files, which are required for continued operation. It then attempts to load the agreement parameters file, certificate file and leapseconds table file. If one or more of these files are present, the associated bit is set in the system status word. Neither of these files are necessary at this time, since the data can be retrieved later from another server. If obtaining these data from another server is considered a significant vulnerability, the files should be present.

In the current management model, the keys and parameter files are generated on each machine separately and the private key obscured. For the most demanding applications, the public key files for a community of users can be copied to all of those users, while one of the parameter files can be selected and copied to all users. However, if security considerations permit, the public key and parameter values, as well as the certificate file and leapseconds table file, can be obtained from other servers during operation. These data completely define the security community and the servers configured for each client. In broadcast client and symmetric passive modes the identity of a particular server may not be known in advance, so the protocol obtains and verifies the public key and host name directly from the server. Ultimately, these procedures may be automated using public certificates retrieved from secure directory services.

Since all files carry a filestamp incorporated in the file name, newer file generations are detected in the data obtained from the one or more configured servers. When detected, the newer generations replace the older ones automatically and the newer ones made available to dependent clients as required. Since the filestamp signatures are refreshed once per day, which causes all associations to reset, the newer generations will eventually overtake all older ones throughout the subnet of servers and dependent clients.

Where security considerations permit and the public key, certificate and agreement parameter files can be retrieved directly from the server, these data can be easily automated. Each server and client runs a shell script perhaps once per month. The script generates new key and parameter files, updates the links and then restarts the daemon. The daemon loads the necessary files and then restarts the protocol with each of its servers, refreshing public keys and parameter files during the process. Clients will not be able to authenticate following daemon restart, but the protocol design is such that they will eventually time out, restart the protocol and retrieve the latest data.

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

Edited into Internet-draft form by:

Patrick Cain. Please notify pcain@genuity.com of editorial omissions or errors.

Full Copyright Statement