Network Working Group   David L. Mills
Internet Draft   University of Delaware
Document: < draft-ietf-stime-ntpauth-00.txt >   June 2000
Category: Standards Track

Public-Key Cryptography for the Network Time Protocol Version 1

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 [12].

1. Abstract

This memorandum describes a scheme for authenticating servers to clients for 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 timestamps are always available using authenticated means; 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 the memorandum contains a description of the security model, approach rationale, protocol design and vulnerability

analysis. It obsoletes a previous report [10] 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 document and improvements described in this memorandum has been implemented, tested and documented in the NTP Version 4 software distribution for Unix, Windows and VMS.

While not strictly a security function, the Autokey scheme 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.

2. Introduction

A reliable distributed network service requires provisions to prevent accidental or malicious attacks on the servers and clients in the network. 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. This is especially important in such ubiquitous network services as directory services, cryptographic key management and time synchronization.

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 [1, 4, 11] 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 [10]. 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.

3. Security Model

Over the last several years the IETF has defined and evolved the IPSEC infrastructure for the protection of privacy and authentication of sources in the Internet, The infrastructure includes the Encapsulating Security Payload (ESP) [3] and Authentication Header (AH) [2] 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 and receiver 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 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, intersection and clustering algorithms are designed to fend off Byzantine sources and cliques. While not necessarily designed to defeat determined intruders, these algorithms and accompanying eleven 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 the servers themselves. Without specific further protection, an intruder can do any or all of the following mischiefs. Further discussion on the assumed intruder model is given in [8], 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 if they require expensive PKI operations.

3. An intruder can intercept, modify and replay a packet. However, it cannot permanently prevent 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 authenticated. This circular interdependence of the timekeeping and authentication functions requires special handling.

4. All authentication 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 webs of trust.

4. 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.

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

2. It must provide for the independent collection of cryptographic values and time values. A client is synchronized to an authentic source 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, multicast/manycast 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 webs of trust.

4.1 Autokey Authentication Scheme

The 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 MD5 message digests to detect packet modification, timestamped RSA digital signatures to verify the source, and Diffie-Hellman key agreements to construct a private key from public values. However, there is no reason why alternative signature schemes and agreement algorithms could be supported. What makes the Autokey scheme unique is the way in which these algorithms are used to deflect intruder attacks while maintaining the integrity and accuracy of the time synchronization function.

The NTP Version 3 symmetric-key cryptography uses keyed-MD5 message digests with a 128-bit private key and 32-bit key ID. In order to retain backward compatibility, the key ID space is partitioned in two subspaces at a pivot point of 65536. Symmetric key IDs have values less than 65536 and indefinite lifetime. Autokey keys 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, multicast/manycast 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 scheme operates separately for each association, so there may be several key lists operating independently at the same time and with associated values and signatures.

An autokey consists of four 32-bit fields in the network order shown below:

```
        +-----------+-----------+-----------+-----------+
        | Source IP |  Dest IP  |  key ID   |  cookie   |
        +-----------+-----------+-----------+-----------+
```

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.

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 is a public value (0), since these packets can be validated independently using signed data in an extension field. 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 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. When this happens, which can 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). In the reference implementation the most recent key ID received is saved for comparison with the first 32 bits of the next following key value. This minimizes the number of hash operations in case a packet is lost.

In client/server mode 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, where 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 addresses reversed. 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 multicast/manycast 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 multicast 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.

The server authentic bit is set only when the cookie or autokey values, depending on mode, and signature are both valid. If the bit is set, the client sends valid timestamps in signed responses. If the bit is not set, the data and signature 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 authentic bit is set, the NTP time values are processed, so that eventually the client will synchronize to an authentic source. In client/server and symmetric modes, 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 data and time values for all associations are cleared and the synchronization and authentication procedures start over from scratch. This insures that old cryptographic and synchronization values never propagate beyond a clock reset.

4.2 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 the source is authentic 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:

1. The public agreement 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. For convenience, the public key/host name, agreement parameters and leap table signatures are recomputed at the same time as the public value signature and carries 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 request message. On request, the cookie, signature and timestamp are returned 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 three signature types is saved for comparison. Once a signature with valid timestamp has been received, packets carrying extension fields with invalid timestamps or older valid timestamps of the same type are discarded before any values are used or signatures 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.

While files carrying cryptographic data not specifically signed, the file names have timestamp extensions which reliably determine the time of generation. As the data are forwarded from machine to machine, the filestamps are preserved. This 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.

In order to further reduce the window of opportunity, even for a fanatical intruder, additional causality constraints can be checked.

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 multicast 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 the public key timestamp and signature timestamps have a granularity of one second, so that a timestamp difference of zero seconds is ambiguous. Furthermore, timestamps 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 operates with polling intervals much longer than one second, so that successive timestamps for the same data type can never by ambiguous.

4.3 Filestamps

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. For these reasons, the name of every cryptographic value file includes a filestamp consisting of the decimal NTP seconds at the time of generation.

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 parameter file and constructs the public and private values to be used later in the agreement protocol. Also optionally, it reads the leap table file. When loading these files it checks the filestamps for consistency and validity. When the client is eventually synchronized to an authentic source, it checks the filestamps for validity relative to the system clock time. If the filestamps are later than the clock time, something is seriously wrong and either the time has synchronized in error or the data files are defective.

When a client mobilizes an association, it retrieves the server host name and public key from the server using the Autokey protocol. Optionally, it retrieves the agreement parameters and leap table, if required, and constructs the public and private values. Optionally, and before going further, it verifies the server credentials using certificate authorities and a web of trust. As above, when the client is eventually synchronized to an authentic source, it again checks the filestamps for validity.

4.4 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 key or agreement parameter files, corrupted data, protocol state mismatches 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. Thus, the pattern of ones and zeros in this register reveals the reachability status of the server for the last eight poll intervals.

With respect to the issues at hand, if this register is nonzero, 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 server 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 cryptographically authenticated packet, 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.

The general approach to Autokey error recovery is to retry the request message at intervals of about one minute until the watchdog timer expires and then restart the protocol from the beginning. At

each state in the protocol the client expects a particular variable
to be received from the server. A NTP packet including the
appropriate request is sent at every poll interval until the variable

is received or a general reset occurs. While this behavior might be considered rather conservative, the advantage is that old cryptographic values can never persist from one mobilization to the next.

There are a number of situations where some action 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 is generated when the next NTP message is sent, assuming there is one. Following is a list of these situations.

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

2. When the poll interval is changed in an association. In this case the calculated expiration times for the keys become invalid.

3. When a general reset is performed in an association.

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

5. When the cryptographic values are refreshed, the key lists for all associations are regenerated.

6. When the client is first synchronized to an authentic source or the system clock is stepped, the key lists for all associations are regenerated.

5 Autokey Protocols

This section describes the Autokey protocols supporting cryptographically secure server and peer authentication. There are three protocols corresponding to the NTP client/server, multicast/manycast and symmetric active/passive modes.

In the descriptions below, it is assumed that the client has the public key and agreement parameters, where required, for the server. These data can be loaded from local files or automatically retrieved from the server as described later in this memorandum. Further information on generating and managing these files is in Appendix B.

The Autokey protocol data unit is the extension field, which 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,
which is set valid (nonzero) only if the sender is synchronized to an

authentic source. An extension fields are discarded before the signature is verified if a signature timestamp is the same as or earlier than the last received timestamp of the same type.

An extension field is also discarded if a protocol or procedure error occurs or the required cryptographic data are incomplete or unavailable or the field values are inconsistent with these data. Otherwise and in general, a response is generated for every request, even if the requestor is not synchronized to an authentic source. However, some responses may have truncated data fields under certain conditions, although the signatures are always present and verifiable.

5.1 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 cookie is hashed from the IP addresses and private value with the key ID field set to zero, as described previously. 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.

The following diagram shows the protocol dance in client/server mode. In this and following diagrams the NTP packet type is shown above the arrow and the extension field(s) message type shown below. There are three cryptographic values instantiated by the dance: the cookie, signature timestamp and authentic bit.

```
server             client
  |                  |
  |    NTP client    |
1 |<-----------------| mobilize association; generate key list;
  |    cookie req    | DNS lookup for canonical name, certificate
  |                  | and public key
  |    NTP server    |
2 |----------------->| store cookie; verify server credentials
  |    cookie rsp    |
  |       ...        |
  |                  |
  |    NTP client    |
3 |<-----------------|
  |                  |
  |    NTP server    |
4 |----------------->|
  |                   |
  |     continue      |
  =  client/server    =
```

The dance begins when the client on the right sends a packet (1) including a cookie request to the server on the left. The server immediately responds with the cookie, signature and timestamp. Upon arrival of this packet (2), the client checks the timestamp, verifies the signature and, if successful, initializes the cookie and signature timestamp and sets the authentic bit. The client will retransmit packet (1) until receiving a valid timestamp and verified signature (2) or until 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 the packet (3) and, following the original NTP protocol, responds with a NAK packet (4), which the client ignores. Eventually, a general reset occurs and the dance restarts from the beginning.

5.2 Multicast/Manycast Mode (5)

In multicast mode, packets are always sent with an extension field. Since the autokey values for these packets use a well known 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 multicast server keeps no state variables specific to each of possibly very many clients and mobilizes no associations for them. 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 multicast mode. There are four cryptographic values instantiated by the dance: the signature timestamp, cookie, autokey values and authentic bit.

```
  server             client
    |                  |
    |   NTP multicast  |
  1 |----------------->| mobilize association; generate key list;
    |   assoc ID rsp   | reverse DNS lookup for canonical name,
    |       ...        | certificate and public key
    |                  |
    |    NTP client    |
  2 |<-----------------|
    |    cookie req    |
    |                  |
    |    NTP server    |
  3 |------------------>| store cookie; verify server credentials
    |    cookie rsp     |
    |       ...         |
    |                  |
    |    NTP client    |
  4 |<-----------------|
    |    autokey req   |
    |                  |
    |    NTP server    |
  5 |----------------->| store autokey
    |    autokey rsp   |
    |       ...        |
    |                  |
    |    NTP client    |
    |<-----------------|
    |                  |
    |    NTP server    |
    |----------------->|
    |                  |
    |     continue     |
    =      volley      =
    |                  |
    |    NTP client    |
    |<-----------------|
    |                  |
    |    NTP server    |
    |----------------->| initialize delay estimate; discard cookie
    |                  | and remaining keys; switch to multicast
    |                  | client mode
    |     continue     |
    =     multicast    =
    |                  |
    |   NTP multicast  |
    |----------------->| server rolls new key list; client refreshes
    |    autokey rsp   | autokey
    |     signature    |
    =                  =
```

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

Upon arrival of the first packet (1), the multicast client mobilizes an association and loads the canonical name and public key as described above. Alternately, it queries the DNS and loads the canonical name, certificate and server public key.

Some time later the client generates a key list and sends a packet

(2) requesting the cookie as in client/server mode. The server immediately responds (3) with the cookie, signature and timestamp. The client checks the timestamp, verifies the signature and, if

successful, initializes the cookie and signature timestamp. The client retransmits packet (2) until receiving a valid timestamp and verified signature (3) or until a general reset occurs.

If an autokey response happens to be in one of the server packets (1, 3), the client can switch to multicast client mode and send no further packets. Otherwise, some time later the client sends a packet (4) requesting the autokey values. The server immediately responds (5) with the values. The client checks the timestamp, verifies the signature and, if successful, initializes the autokey values and signature timestamp and sets the authentic bit. The client retransmits packet (4) until receiving a valid timestamp and verified signature (5) or until a general reset occurs.

After successful verification, there is no further need for extension fields, unless the server regenerates the cookie or the server regenerates the key list and the Autokey response message happens to be lost. When this happens, the server fails to authenticate the packets (1). Eventually, a general reset occurs and the dance restarts from the beginning. 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 have been accumulated, the client discards the cookie and remaining keys on the key list, switches to multicast client mode and sets the clock.

5.3 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. The particular choice of 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.

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 host name and public key can be configured in advance. The other scenario is when one peer operates with a configured association and begins operation with another peer without a configured association and begins operation in symmetric-passive mode.

The following diagram shows the protocol dance in symmetric-active/passive mode. The exchange is similar in the symmetric-active/active mode, although the order can change depending on which peer starts the dance. There are four cryptographic values instantiated by the dance: the signature timestamp, cookie, autokey values and authentic bit.

```
active            passive
  |                  |
  |    NTP active    |
1 |----------------->| mobilize association; query DNS for
  |    public rsp    | canonical name, certificate and public key;
  |                  | verify public signature, compute and
  |    public req    | initialize agreed key
  |       ...        |
  |                  |
  |    NTP passive   |
2 |<-----------------|
  |    public rsp    |
  |    autokey req   |
  |       ...        |
  |                  |
  |    NTP active    |
3 |----------------->| verify autokey signature, initialize
  |    autokey rsp   | autokey key; sign public values
  |    autokey req   |
  |       ...        |
  |                  |
  |    NTP passive   |
4 |<-----------------|
  |    autokey rsp   |
  |       ...        |
  |                  |
  |    NTP active    |
  |----------------->| regular operation
  |       ...        |
  |                  |
  |    NTP passive   |
  |<-----------------|
  |                  |
  |     continue     |
  =  active/passive  =
```

The dance begins when the active peer on the left in the diagram sends a packet (1) to the passive peer on the right. Before sending the first packet, the active peer generates a key list using the default key (zero) and initializes the autokey values and signature along with the public agreement value and signature.

The first packet from the active peer includes its public value and signature along with a request for the public value of the passive peer. Upon arrival of this packet, the passive peer mobilizes an association and loads the canonical name and public key as described above. Alternately, it queries the DNS and loads the canonical name, certificate and public key of the active peer. The passive peer checks the timestamp, verifies the signature and, if successful, executes the agreements algorithm and initializes the cookie and signature timestamp. As the cookie affects the autokey values, the key list is regenerated with the cookie. The active peer retransmits packet (1) until receiving a valid public value (2) or until a general reset occurs.

Some time later the passive peer sends a packet (2) to the active peer including its public value and signature along with a request for the autokey values of the active peer. Upon arrival of this packet, the active peer checks the timestamp, verifies the signature and, if successful, executes the agreements algorithm and initializes the cookie and signature timestamp. As the cookie affects the autokey values, the key list is regenerated with the cookie. The passive peer retransmits packet (2) until receiving a valid autokey values (3) or until a general reset occurs.

Some time later the active peer sends a packet (3) to the passive peer including its autokey values and signature along with a request for the autokey values of the passive peer. Upon arrival of this packet, the passive peer checks the timestamp, verifies the signature and, if successful, initializes the autokey values and sets its authentic bit. The active peer retransmits packet (3) until receiving a valid autokey values (4) or until a general reset occurs.

Some time later the passive peer sends a packet (4) to the active peer including its autokey values and signature. Upon arrival of this packet, the active peer checks the timestamp, verifies the signature and, if successful, initializes the autokey values and sets the authentic bit.

After successful verification, 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. In any case, the request will be retransmitted at intervals until a general reset occurs.

6. Additional Protocols

While not mentioned in the above protocol descriptions, there are provisions to negotiate the algorithms and algorithm parameters, retrieve the public key and host name, and retrieve the agreement parameters and ancillary data using the defined requests summarized in Appendix A. Ordinarily, a client or peer requests the public key and host name in the first message from an association to a server or peer. The response includes the filestamp and is signed by the server using its private key. The signature and filestamp is useful to confirm the correct key generation and to verify correct procedure.

Each association requiring public key authentication cannot proceed until the response has been received.

The NIST provides a table showing the epoch for all occasions of leap second insertions since 1972. The table is maintained in a file called pub/leap-seconds and available for anonymous FTP download. While not strictly a security function, the table can be retrieved from an NTP server using the Autokey protocol if the feature is enabled. If enabled and the leap table is not available, a request is included in the next Autokey message. The response includes the original filestamp generated by the NIST and is signed and timestamped. Note that the table will be requested by all associations, either configured or not; but, none of the associations can proceed until one of them has received the response. After this, the table can be provided on request to other clients and servers.

If any associations are operating in symmetric modes, the agreement parameters are required to complete the protocol. If the parameters are needed and not currently available, they are requested in the next message. The response includes the original filestamp and is signed as before. Note that the parameters will be requested by all associations needing them, either configured or not; but, none of the associations can proceed until one of them has received the response. After this, the parameters can be provided on request to other clients and servers.

7 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 parameters is unlikely. However, vulnerabilities remain in the way the actual cryptographic data, including the cookie and autokey values, are computed and used.

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 successfulcryptanalysis 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 authentic 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.

7.1 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 synchronized to an authentic source. 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 synchronized to an authentic source, 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 synchronized to an authentic source, 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 multicast 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 multicast 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 multicast 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.

8 Present Status

The Autokey scheme has been implemented in the public software distribution for NTP Version 4 and has been tested in all machines of either endian persuasion and both 32- and 64-bit architectures. 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.

9 Future Plans

The analysis, design and implementation of the Autokey scheme is basically mature; however, There are two remaining implementation issues. One has to do with the Unix sockets semantics used for multicast. The problem is how to set the source address when more than one interface is present. Since the Autokey scheme hashes the IP addresses, as well as the NTP header, it is necessary that the

correct address be known before the hash can be computed. In the
present implementation the address is not known until the first

packet arrives, which considerably complicates the protocol. Probably nothing short of a complete rewrite of the I/O code will fix this.

The other issue is support for Secure DNS services, especially the retrieval of public certificates. A complicating factor is the existing banal state of the configuration and resolver code in the NTP daemon. 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 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.

Public Key (1)

This extension field is reserved for future use as an algorithm and algorithm parameter offer/select exchange. The command code is reserved.

Association ID (2)

This message is sent by a multicast server as an unsolicited response only; there is no corresponding request message of this type. 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     |       2       |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Association ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Association ID field contains the association ID of the server. This response is included in every packet sent by a multicast server, except when a new key list is generated. There is no timestamp or signature associated with this message.

Cookie (3)

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 an authenticated 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.

Autokey (4)

A multicast 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 an authenticated 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.

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 an authenticated 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 an authenticated 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 an authenticated 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.

TAI Leap Second 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 leap second 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                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Leap Second Table Length                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                        Leap Second Table                      =
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Signature Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
=                         Signature                             =
|                                                               |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The NTP extension field format consists of a table with one entry in NTP seconds for each leap second insertion and in the order from the most recent insertion to the first. Since UTC led TAI by ten seconds at the first insertion and each insertion since then adds one second, the current UTC-TAI offset is simply the sum of these values.

Since the leap second table is a property of the server and not any particular association, the association ID field for the request and response is 0. The Leap Second 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 leap second 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 an authenticated 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 Leap Second 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.

In the reference implementation, Autokey key management is 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. From these data the program generates four files, one containing random DES/MD5 private keys, which are not used in the Autokey scheme, another containing the RSA private key, a third the RSA public key, and a fourth the agreement parameters. In addition, the leap second 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 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 stored in shared directory /usr/local/etc, which is where the daemon looks for them by default. The private RSA key file should be stored 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 parameter file and, if enabled by a configuration file bit, it attempts to load the leap second table file. 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. The set of 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 leap second table can be obtained from other servers during operation. These data completely define the security community and the servers configured for each client. In multicast 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 will be automated using public certificates retrieved from secure directory services.

Where security considerations permit and the public key and parameter data can be retrieved directly from the server, key and parameter refreshment can be easily automated. Each server and client runs a

shell script perhaps once per month to generate new key and parameter files, update the links and then restarts the daemon. The daemon will load the necessary files and then restart the protocol with each of its servers or peers, refreshing public keys and parameter files during the process.
Clients of the daemon will not be able to authenticate following daemon restart, of course, but the protocol design is such that they will eventually time out, restart the protocol and retrieve the latest data.

The parameters and leap second table files are a special case, since the expectation is that all servers and clients in the network have the same versions. Therefore, the scripts should provide for automatic, secure transfer of these files to all the lowest-stratum servers in the security compartment.

Unlike ssh, 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 are required for each interface address with a distinct name. In the NTP design the canonical host name, as returned by the gethostname() library function, represents all interface addresses. Since at least in some host configurations the canonical 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.

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

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

3. Kent, S., and R. Atkinson. IP Encapsulating security payload (ESP). Request for Comments RFC-2406, Internet Engineering Task Force, November 1998.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.

9. 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.

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

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

12. Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997

11. Author's Address

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

Full Copyright Statement