Internet Engineering Task Force (IETF)
Request for Comments: 5906
Category: Informational
ISSN: 2070-1721

B. Haberman, Ed.
JHU/APL
D. Mills
U. Delaware

June 2010

Network Time Protocol Version 4: Autokey Specification

Abstract

This memo describes the Autokey security model for authenticating servers to clients using the Network Time Protocol (NTP) and public key cryptography. Its design is based on the premise that IPsec schemes cannot be adopted intact, since that would preclude stateless servers and severely compromise timekeeping accuracy. In addition, Public Key Infrastructure (PKI) schemes presume authenticated time values are always available to enforce certificate lifetimes; however, cryptographically verified timestamps require interaction between the timekeeping and authentication functions.

This memo includes the Autokey requirements analysis, design principles, and protocol specification. A detailed description of the protocol states, events, and transition functions is included. A prototype of the Autokey design based on this memo has been implemented, tested, and documented in the NTP version 4 (NTPv4) software distribution for the Unix, Windows, and Virtual Memory System (VMS) operating systems at http://www.ntp.org.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5906.

Table of Contents

1. 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 a client can verify the authenticity of a server using only public information. Survivability requires protection from faulty implementations, improper operation, and possibly malicious clogging and replay attacks.

This memo describes a cryptographically sound and efficient methodology for use in the Network Time Protocol (NTP) [RFC5905]. The various key agreement schemes [RFC4306][RFC2412][RFC2522] proposed 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, e.g., as implemented in the OpenSSL library, leads to the conclusion that any scheme requiring every NTP packet to carry a PKI digital signature would result in unacceptably poor timekeeping performance.

The Autokey protocol 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 implemented, tested, and deployed in the Internet of today. A detailed description of the security model, design principles, and implementation is presented in this memo.

This informational document describes the NTP extensions for Autokey as implemented in an NTPv4 software distribution available from http://www.ntp.org. This description is provided to offer a basis for future work and a reference for the software release. This document also describes the motivation for the extensions within the protocol.

2. NTP 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 that are valid only over designated time intervals; but, time intervals can be enforced only when participating servers and clients are reliably synchronized to UTC. In addition, the NTP subnet is

hierarchical by nature, so time and trust flow from the primary servers at the root through secondary servers to the clients at the leaves.

A client can claim authentic to dependent applications only if all servers on the path to the primary servers are bona fide authentic. In order to emphasize this requirement, in this memo, the notion of "authentic" is replaced by "proventic", an adjective 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 derivatives of authentic will be adopted for proventic as well. In NTP, each server authenticates the next-lower stratum servers and proventicates (authenticates by induction) the lowest stratum (primary) servers. Serious computer linguists would correctly interpret the proventic relation as the transitive closure of the authentic relation.

It is important to note that the notion of proventic does not necessarily imply the time is correct. An NTP client mobilizes a number of concurrent associations with different servers and uses a crafted agreement algorithm to pluck truechimers from the population possibly including falsetickers. A particular association is proventic if the server certificate and identity have been verified by the means described in this memo. However, the statement "the client is synchronized to proventic sources" means that the system clock has been set using the time values of one or more proventic associations and according to the NTP mitigation algorithms.

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) [RFC4303] and Authentication Header (AH) [RFC4302] for IPv4 and IPv6. Cryptographic algorithms that use these headers for various purposes include those developed for the PKI, including various message digest, digital signature, and key agreement algorithms. This memo takes no position on which message digest or digital signature algorithm is used. This is established by a profile for each community of users.

It will facilitate the discussion in this memo to refer to the reference implementation available at http://www.ntp.org. It includes Autokey as described in this memo and is available to the general public; however, it is not part of the specification itself. The cryptographic means used by the reference implementation and its user community are based on the OpenSSL cryptographic software library available at http://www.openssl.org, but other libraries with equivalent functionality could be used as well. It is important for

distribution and export purposes that the way in which these algorithms are used precludes encryption of any data other than incidental to the construction of digital signatures.

The fundamental assumption in NTP about the security model is that packets transmitted over the Internet can be intercepted by those other than the intended recipient, remanufactured in various ways, and replayed in whole or part. These packets can cause the client to believe or produce incorrect information, cause protocol operations to fail, interrupt network service, or consume precious network and processor resources.

In the case of NTP, the assumed goal of the intruder is to inject false time values, disrupt the protocol or clog the network, servers, or clients with spurious packets that exhaust resources and deny service to legitimate applications. The mission of the algorithms and protocols described in this memo is to detect and discard spurious packets sent by someone other than the intended sender or sent by the intended sender, but modified or replayed by an intruder.

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

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

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

3. In a wiretap attack, the intruder can intercept, modify, and replay a packet. However, it cannot permanently prevent onward transmission of the original packet; that is, it cannot break the wire, only tell lies and congest it. Except in the unlikely cases considered in Section 12, the modified packet cannot arrive at the victim before the original packet, nor does it have the server private keys or identity parameters.

4. In a man-in-the-middle or masquerade attack, the intruder is positioned between the server and client, so it can intercept, modify, and replay a packet and prevent onward transmission of the original packet. Except in unlikely cases considered in Section 12, the middleman does not have the server private keys.

The NTP security model assumes the following possible limitations.

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

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

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

4. Client security functions must involve only public values transmitted over the net. Private values must never be disclosed beyond the machine on which they were created, except in the case of a special trusted agent (TA) assigned for this purpose.

Unlike the Secure Shell (SSH) security model, where the client must be securely authenticated to the server, in NTP, the server must be securely authenticated 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 authenticate the client and others not.

3. Approach

The Autokey protocol described in this memo is designed to meet the following objectives. In-depth discussions on these objectives is in the web briefings and will not be elaborated in this memo. Note that here, and elsewhere in this memo, mention of broadcast mode means multicast mode as well, with exceptions as noted in the NTP software documentation [RFC5905].

1. It must interoperate with the existing NTP architecture model and protocol design. In particular, it must support the symmetric key scheme described in [RFC1305]. 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. An NTP packet is accepted for processing only when the required cryptographic values have been obtained and verified and the packet has passed all header sanity checks.

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

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

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

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

4. Autokey Cryptography

Autokey cryptography is based on the PKI algorithms commonly used in the Secure Shell and Secure Sockets Layer (SSL) applications. As in these applications, Autokey uses message digests to detect packet modification, digital signatures to verify credentials, and public certificates to provide traceable authority. 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.

Autokey, like many other remote procedure call (RPC) protocols, depends on message digests for basic authentication; however, it is important to understand that message digests are also used by NTP when Autokey is not available or not configured. Selection of the digest algorithm is a function of NTP configuration and is transparent to Autokey.

The protocol design and reference implementation support both 128-bit and 160-bit message digest algorithms, each with a 32-bit key ID. In order to retain backwards compatibility with NTPv3, the NTPv4 key ID space is partitioned in two subspaces at a pivot point of 65536. Symmetric key IDs have values less than the pivot and indefinite lifetime. Autokey key IDs have pseudo-random values equal to or greater than the pivot and are expunged immediately after use.

Both symmetric key and public key cryptography authenticate as shown in Figure 1. The server looks up the key associated with the key ID and calculates the message digest from the NTP header and extension fields together with the key value. The key ID and digest form the message authentication code (MAC) included with the message. The client does the same computation using its local copy of the key and compares the result with the digest in the MAC. If the values agree, the message is assumed authentic.

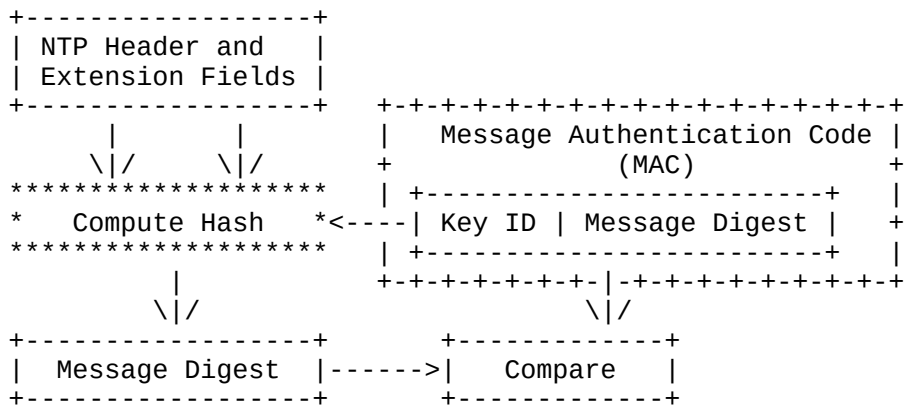


Figure 1: Message Authentication

Autokey uses specially contrived session keys, called autokeys, and a precomputed pseudo-random sequence of autokeys that are saved in the autokey list. The Autokey protocol operates separately for each association, so there may be several autokey sequences operating independently at the same time.

Figure 2: NTPv4 Autokey

An autokey is computed from four fields in network byte order as shown in Figure 2. The four values are hashed using the MD5 algorithm to produce the 128-bit autokey value, which in the reference implementation is stored along with the key ID in a cache used for symmetric keys as well as autokeys. Keys are retrieved from the cache by key ID using hash tables and a fast lookup algorithm.

For use with IPv4, the Src Address and Dst Address 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 addresses and key ID are public values visible in the packet, while the cookie can be a public value or shared private value, depending on the NTP mode.

The NTP packet format has been augmented to include one or more extension fields piggybacked between the original NTP header and the MAC. For packets without extension fields, the cookie is a shared private value. For packets with extension fields, the cookie has a default public value of zero, since these packets are validated independently using digital signatures.

There are some scenarios where the use of endpoint IP addresses may be difficult or impossible. These include configurations where network address translation (NAT) devices are in use or when addresses are changed during an association lifetime due to mobility constraints. For Autokey, the only restriction is that the address fields that are visible in the transmitted packet must be the same as those used to construct the autokey list and that these fields be the same as those visible in the received packet. (The use of alternative means, such as Autokey host names (discussed later) or hashes of these names may be a topic for future study.)

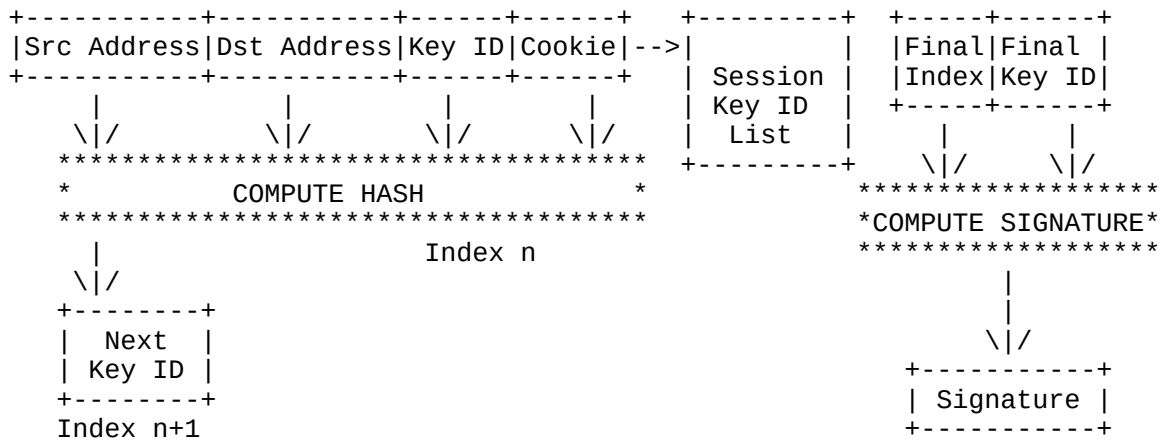


Figure 3: Constructing the Key List

Figure 3 shows how the autokey list and autokey values are computed. The key IDs used in the autokey list consist of a sequence starting with a random 32-bit nonce (autokey seed) greater than or equal to the pivot as the first key ID. The first autokey is computed as above using the given cookie and autokey seed and assigned index 0. The first 32 bits of the result in network byte order become the next key ID. The MD5 hash of the autokey is the key value saved in the key cache along with the key ID. The first 32 bits of the key become the key ID for the next autokey assigned index 1.

Operations continue to generate the entire list. It may happen that a newly generated key ID is less than the pivot or collides with another one already generated (birthday event). When this happens, which occurs 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, so for poll intervals above one hour, a new key list containing only a single entry is regenerated for every poll.

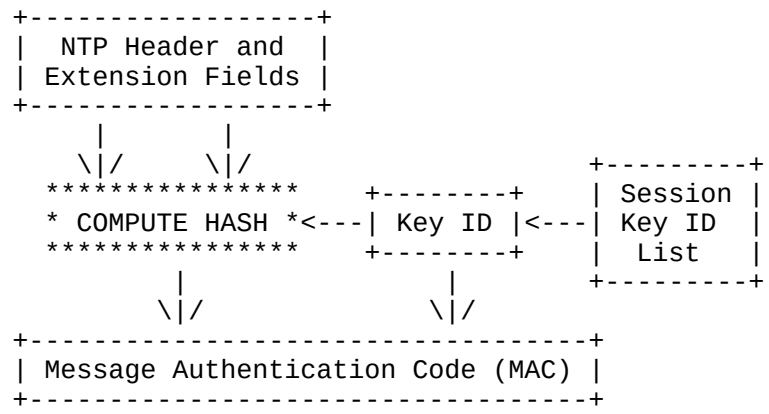


Figure 4: Transmitting Messages

The index of the last autokey in the list is saved along with the key ID for that entry, collectively called the autokey values. The autokey values are then signed for use later. The list is used in reverse order as shown in Figure 4, so that the first autokey used is the last one generated.

The Autokey protocol includes a message to retrieve the autokey values and verify the signature, so that subsequent packets can be validated using one or more hashes that eventually match the last 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 single packet is lost.

5. Autokey Protocol Overview

The Autokey protocol includes a number of request/response exchanges that must be completed in order. In each exchange, a client sends a request message with data and expects a server response message with data. Requests and responses are contained in extension fields, one request or response in each field, as described later. An NTP packet can contain one request message and one or more response messages. The following is a list of these messages.

- Parameter exchange. The request includes the client host name and status word; the response includes the server host name and status word. The status word specifies the digest/signature scheme to use and the identity schemes supported.

- Certificate exchange. The request includes the subject name of a certificate; the response consists of a signed certificate with that subject name. If the issuer name is not the same as the subject name, it has been signed by a host one step closer to a trusted host, so certificate retrieval continues for the issuer name. If it is trusted and self-signed, the trail concludes at the trusted host. If nontrusted and self-signed, the host certificate has not yet been signed, so the trail temporarily loops. Completion of this exchange lights the VAL bit as described below.

- Identity exchange. The certificate trail is generally not considered sufficient protection against man-in-the-middle attacks unless additional protection such as the proof-of-possession scheme described in [RFC2875] is available, but this is expensive and requires servers to retain state. Autokey can use one of the challenge/response identity schemes described in Appendix B. Completion of this exchange lights the IFF bit as described below.

- Cookie exchange. The request includes the public key of the server. The response includes the server cookie encrypted with this key. The client uses this value when constructing the key list. Completion of this exchange lights the COOK bit as described below.

- Autokey exchange. The request includes either no data or the autokey values in symmetric modes. The response includes the autokey values of the server. These values are used to verify the autokey sequence. Completion of this exchange lights the AUT bit as described below.

- Sign exchange. This exchange is executed only when the client has synchronized to a proventic source. The request includes the self-signed client certificate. The server acting as certification authority (CA) interprets the certificate as a X.509v3 certificate request. It extracts the subject, issuer, and extension fields, builds a new certificate with these data along with its own serial number and expiration time, then signs it using its own private key and includes it in the response. The client uses the signed certificate in its own role as server for dependent clients. Completion of this exchange lights the SIGN bit as described below.

- Leapseconds exchange. This exchange is executed only when the client has synchronized to a proventic source. This exchange occurs when the server has the leapseconds values, as indicated in the host status word. If so, the client requests the values and compares them with its own values, if available. If the server

values are newer than the client values, the client replaces its own with the server values. The client, acting as server, can now provide the most recent values to its dependent clients. In symmetric mode, this results in both peers having the newest values. Completion of this exchange lights the LPT bit as described below.

Once the certificates and identity have been validated, subsequent packets are validated by digital signatures and the autokey sequence. The association is now proventic with respect to the downstratum trusted host, but is not yet selectable to discipline the system clock. The associations accumulate time values, and the mitigation algorithms continue in the usual way. When these algorithms have culled the falsetickers and cluster outliers and at least three survivors remain, the system clock has been synchronized to a proventic source.

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

6. NTP Secure Groups

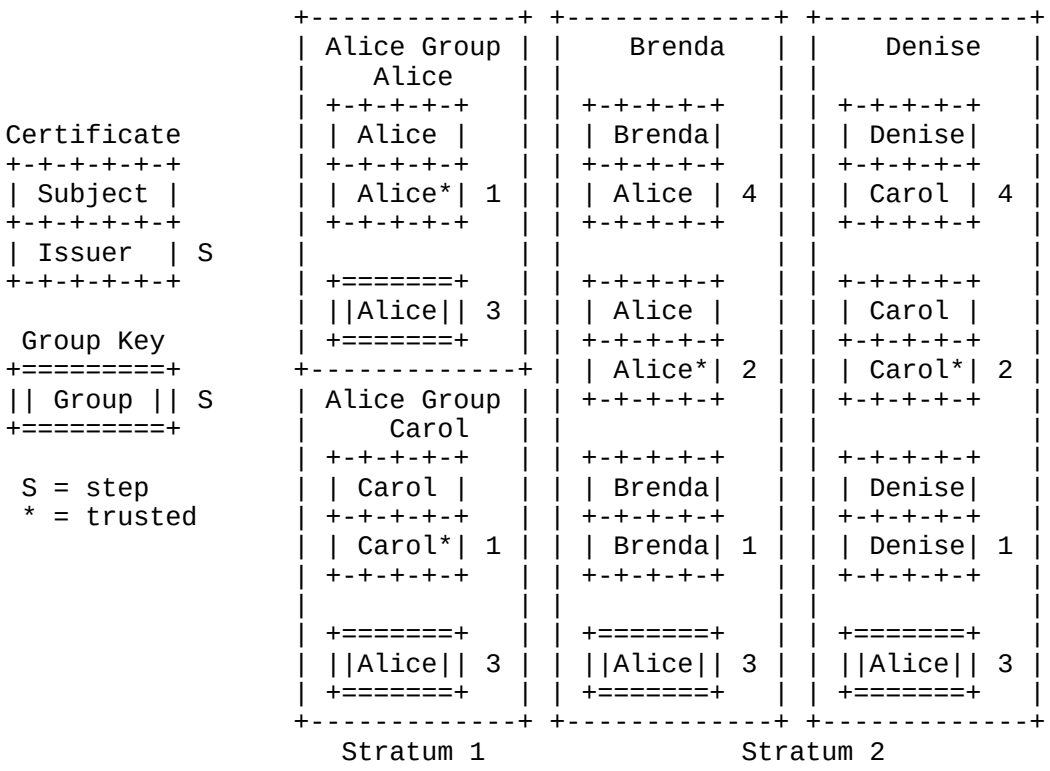
NTP secure groups are used to define cryptographic compartments and security hierarchies. A secure group consists of a number of hosts dynamically assembled as a forest with roots the trusted hosts (THs) at the lowest stratum of the group. The THs do not have to be, but often are, primary (stratum 1) servers. A trusted authority (TA), not necessarily a group host, generates private identity keys for servers and public identity keys for clients at the leaves of the forest. The TA deploys the server keys to the THs and other designated servers using secure means and posts the client keys on a public web site.

For Autokey purposes, all hosts belonging to a secure group have the same group name but different host names, not necessarily related to the DNS names. The group name is used in the subject and issuer fields of the TH certificates; the host name is used in these fields for other hosts. Thus, all host certificates are self-signed. During the use of the Autokey protocol, a client requests that the server sign its certificate and caches the result. A certificate

trail is constructed by each host, possibly via intermediate hosts and ending at a TH. Thus, each host along the trail retrieves the entire trail from its server(s) and provides this plus its own signed certificates to its clients.

Secure groups can be configured as hierarchies where a TH of one group can be a client of one or more other groups operating at a lower stratum. In one scenario, THs for groups RED and GREEN can be cryptographically distinct, but both be clients of group BLUE operating at a lower stratum. In another scenario, THs for group CYAN can be clients of multiple groups YELLOW and MAGENTA, both operating at a lower stratum. There are many other scenarios, but all must be configured to include only acyclic certificate trails.

In Figure 5, the Alice group consists of THs Alice, which is also the TA, and Carol. Dependent servers Brenda and Denise have configured Alice and Carol, respectively, as their time sources. Stratum 3 server Eileen has configured both Brenda and Denise as her time sources. Public certificates are identified by the subject and signed by the issuer. Note that the server group keys have been previously installed on Brenda and Denise and the client group keys installed on all machines.


Certificate
Subject
Issuer S
Group Key
Group S
S = step
* = trusted
Alice Group Alice
Alice
Alice* 1
Alice 3
Alice Group Carol
Carol
Carol* 1
Alice 3
Brenda
Alice 4
Alice
Alice* 2
Brenda 1
Alice 3
Denise
Carol 4
Carol
Carol* 2
Denise 1
Alice 3
Stratum 1
Stratum 2

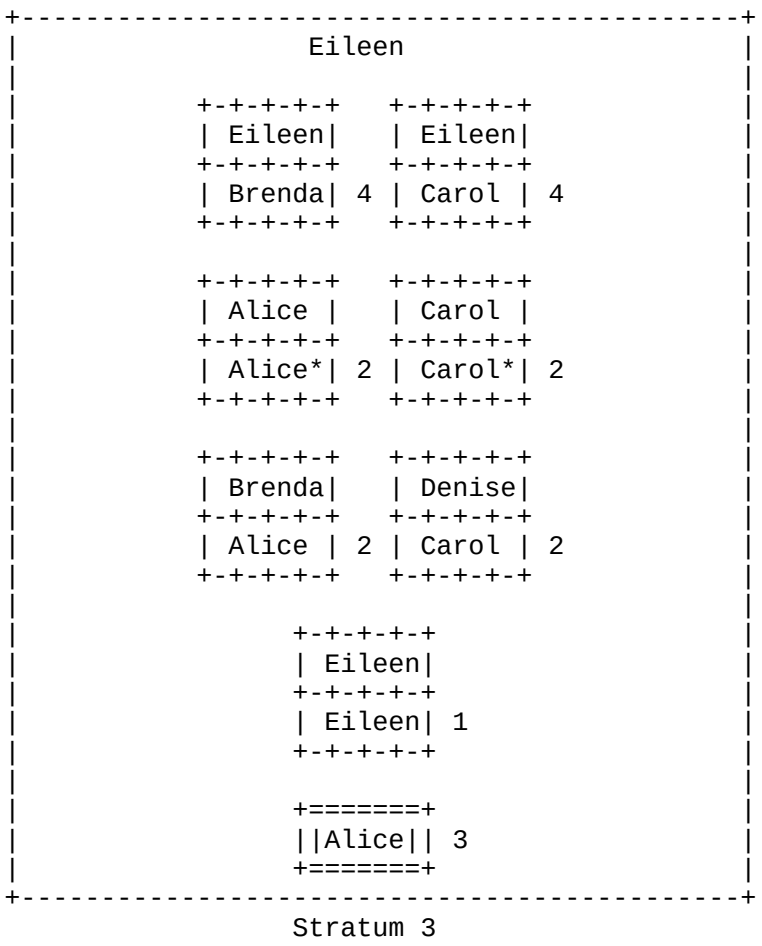


Figure 5: NTP Secure Groups

The steps in hiking the certificate trails and verifying identity are as follows. Note the step number in the description matches the step number in the figure.

1. The girls start by loading the host key, sign key, self-signed certificate, and group key. Each client and server acting as a client starts the Autokey protocol by retrieving the server host name and digest/signature. This is done using the ASSOC exchange described later.

2. They continue to load certificates recursively until a self-signed trusted certificate is found. Brenda and Denise immediately find trusted certificates for Alice and Carol,

respectively, but Eileen will loop because neither Brenda nor Denise have their own certificates signed by either Alice or Carol. This is done using the CERT exchange described later.

3. Brenda and Denise continue with the selected identity schemes to verify that Alice and Carol have the correct group key previously generated by Alice. This is done using one of the identity schemes IFF, GQ, or MV, described later. If this succeeds, each continues in step 4.

4. Brenda and Denise present their certificates for signature using the SIGN exchange described later. If this succeeds, either one of or both Brenda and Denise can now provide these signed certificates to Eileen, which may be looping in step 2. Eileen can now verify the trail via either Brenda or Denise to the trusted certificates for Alice and Carol. Once this is done, Eileen can complete the protocol just as Brenda and Denise did.

For various reasons, it may be convenient for a server to have client keys for more than one group. For example, Figure 6 shows three secure groups Alice, Helen, and Carol arranged in a hierarchy. Hosts A, B, C, and D belong to Alice with A and B as her THs. Hosts R and S belong to Helen with R as her TH. Hosts X and Y belong to Carol with X as her TH. Note that the TH for a group is always the lowest stratum and that the hosts of the combined groups form an acyclic graph. Note also that the certificate trail for each group terminates on a TH for that group.

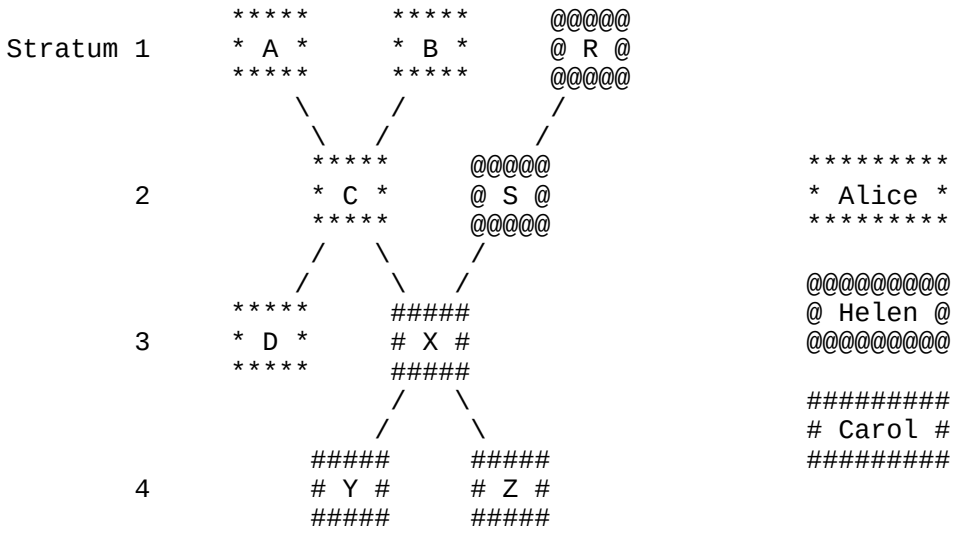


Figure 6: Hierarchical Overlapping Groups

The intent of the scenario is to provide security separation, so that servers cannot masquerade as clients in other groups and clients cannot masquerade as servers. Assume, for example, that Alice and Helen belong to national standards laboratories and their server keys are used to confirm identity between members of each group. Carol is a prominent corporation receiving standards products and requiring cryptographic authentication. Perhaps under contract, host X belonging to Carol has client keys for both Alice and Helen and server keys for Carol. The Autokey protocol operates for each group separately while preserving security separation. Host X can prove identity in Carol to clients Y and Z, but cannot prove to anybody that it belongs to either Alice or Helen.

7. Identity Schemes

A digital signature scheme provides secure server authentication, but it does not provide protection against masquerade, unless the server identity is verified by other means. The PKI model requires a server to prove identity to the client by a certificate trail, but independent means such as a driver's license are required for a CA to sign the server certificate. While Autokey supports this model by default, in a hierarchical ad hoc network, especially with server discovery schemes like NTP manycast, proving identity at each rest stop on the trail must be an intrinsic capability of Autokey itself.

While the identity scheme described in [RFC2875] is based on a ubiquitous Diffie-Hellman infrastructure, it is expensive to generate and use when compared to others described in Appendix B. In principle, an ordinary public key scheme could be devised for this purpose, but the most stringent Autokey design requires that every challenge, even if duplicated, results in a different acceptable response.

1. The scheme must have a relatively long lifetime, certainly longer than a typical certificate, and have no specific lifetime or expiration date. At the time the scheme is used, the host has not yet synchronized to a proventic source, so the scheme cannot depend on time.

2. As the scheme can be used many times where the data might be exposed to potential intruders, the data must be either nonces or encrypted nonces.

3. The scheme should allow designated servers to prove identity to designated clients, but not allow clients acting as servers to prove identity to dependent clients.

4. To the greatest extent possible, the scheme should represent a zero-knowledge proof; that is, the client should be able to verify that the server has the correct group key, but without knowing the key itself.

There are five schemes now implemented in the NTPv4 reference implementation to prove identity: (1) private certificate (PC), (2) trusted certificate (TC), (3) a modified Schnorr algorithm (IFF aka Identify Friendly or Foe), (4) a modified Guillou-Quisquater (GQ) algorithm, and (5) a modified Mu-Varadharajan (MV) algorithm. Not all of these provide the same level of protection and one, TC, provides no protection but is included for comparison. The following is a brief summary description of each; details are given in Appendix B.

The PC scheme involves a private certificate as group key. The certificate is distributed to all other group members by secure means and is never revealed outside the group. In effect, the private certificate is used as a symmetric key. This scheme is used primarily for testing and development and is not recommended for regular use and is not considered further in this memo.

All other schemes involve a conventional certificate trail as described in [RFC5280]. This is the default scheme when an identity scheme is not required. While the remaining identity schemes incorporate TC, it is not by itself considered further in this memo.

The three remaining schemes IFF, GQ, and MV involve a cryptographically strong challenge-response exchange where an intruder cannot deduce the server key, even after repeated observations of multiple exchanges. In addition, the MV scheme is properly described as a zero-knowledge proof, because the client can verify the server has the correct group key without either the server or client knowing its value. These schemes start when the client sends a nonce to the server, which then rolls its own nonce, performs a mathematical operation and sends the results to the client. The client performs another mathematical operation and verifies the results are correct.

8. Timestamps and Filestamps

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

timestamp in the form of the NTP seconds when it was created. If the system clock is synchronized to a proventic source, a signature is produced with a valid (nonzero) timestamp. Otherwise, there is no signature and the timestamp is invalid (zero). The protocol detects and discards extension fields with old or duplicate timestamps, before any values are used or signatures are verified.

Signatures are computed only when cryptographic values are created or modified, which is by design not very often. Extension fields carrying these signatures are copied to messages as needed, but the signatures are not recomputed. There are three signature types:

1. Cookie signature/timestamp. The cookie is signed when created by the server and sent to the client.

2. Autokey signature/timestamp. The autokey values are signed when the key list is created.

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

The most recent timestamp received of each type is saved for comparison. Once a signature with a valid timestamp has been received, messages with invalid timestamps or earlier valid timestamps of the same type are discarded before the signature is verified. This is most important in broadcast mode, which could be vulnerable to a clogging attack without this test.

All cryptographic values used by the protocol are time sensitive and are regularly refreshed. In particular, files containing cryptographic values used by signature and encryption algorithms are regenerated from time to time. It is the intent that file regenerations occur without specific advance warning and without requiring prior distribution of the file contents. While cryptographic data files are not specifically signed, every file is associated with a filestamp showing the NTP seconds at the creation epoch.

Filestamps and timestamps can be compared in any combination and use the same conventions. It is necessary to compare them from time to time to determine which are earlier or later. Since these quantities have a granularity only to the second, such comparisons are ambiguous if the values are in the same second.

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

9. Autokey Operations

The NTP protocol has three principal modes of operation: client/server, symmetric, and broadcast and each has its own Autokey program, or dance. Autokey choreography is designed to be non-intrusive and to require no additional packets other than for regular NTP operations. The NTP and Autokey protocols operate simultaneously and independently. When the dance is complete, subsequent packets are validated by the autokey sequence and thus considered proventic as well. Autokey assumes NTP clients poll servers at a relatively low rate, such as once per minute or slower. In particular, it assumes that a request sent at one poll opportunity will normally result in a response before the next poll opportunity; however, the protocol is robust against a missed or duplicate response.

The server dance was suggested by Steve Kent over lunch some time ago, but considerably modified since that meal. The server keeps no state for each client, but uses a fast algorithm and a 32-bit random private value (server seed) to regenerate the cookie upon arrival of a client packet. The cookie is calculated as the first 32 bits of the autokey computed from the client and server addresses, key ID zero, and the server seed as cookie. The cookie is used for the actual autokey calculation by both the client and server and is thus specific to each client separately.

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

In the broadcast dance, clients normally do not send packets to the server, except when first starting up. At that time, the client runs the server dance to verify the server credentials and calibrate the propagation delay. The dance requires the association ID of the particular server association, since there can be more than one operating in the same server. For this purpose, the server packet includes the association ID in every response message sent and, when sending the first packet after generating a new key list, it sends the autokey values as well. After obtaining and verifying the autokey values, no extension fields are necessary and the client verifies further server packets using the autokey sequence.

The symmetric dance is similar to the server dance and requires only a small amount of state between the arrival of a request and departure of the response. The key list for each direction is generated separately by each peer and used independently, but each is generated with the same cookie. The cookie is conveyed in a way similar to the server dance, except that the cookie is a simple nonce. There exists a possible race condition where each peer sends a cookie request before receiving the cookie response from the other peer. In this case, each peer winds up with two values, one it generated and one the other peer generated. The ambiguity is resolved simply by computing the working cookie as the EXOR of the two values.

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

10. Autokey Protocol Messages

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 optional data. To avoid deadlocks, any number of responses can be included in a packet, but only one request can be. A response is generated for every request, even if the requestor is not synchronized to a proventic source, but most contain meaningful data only if the responder is synchronized to a proventic source. Some requests and most responses carry timestamped signatures. The signature covers the entire extension field, including the timestamp

and filestamp, where applicable. Only if the packet has correct format, length, and message digest are cycles spent to verify the signature.

There are currently eight Autokey requests and eight corresponding responses. The NTP packet format is described in [RFC5905] and the extension field format used for these messages is illustrated in Figure 7.

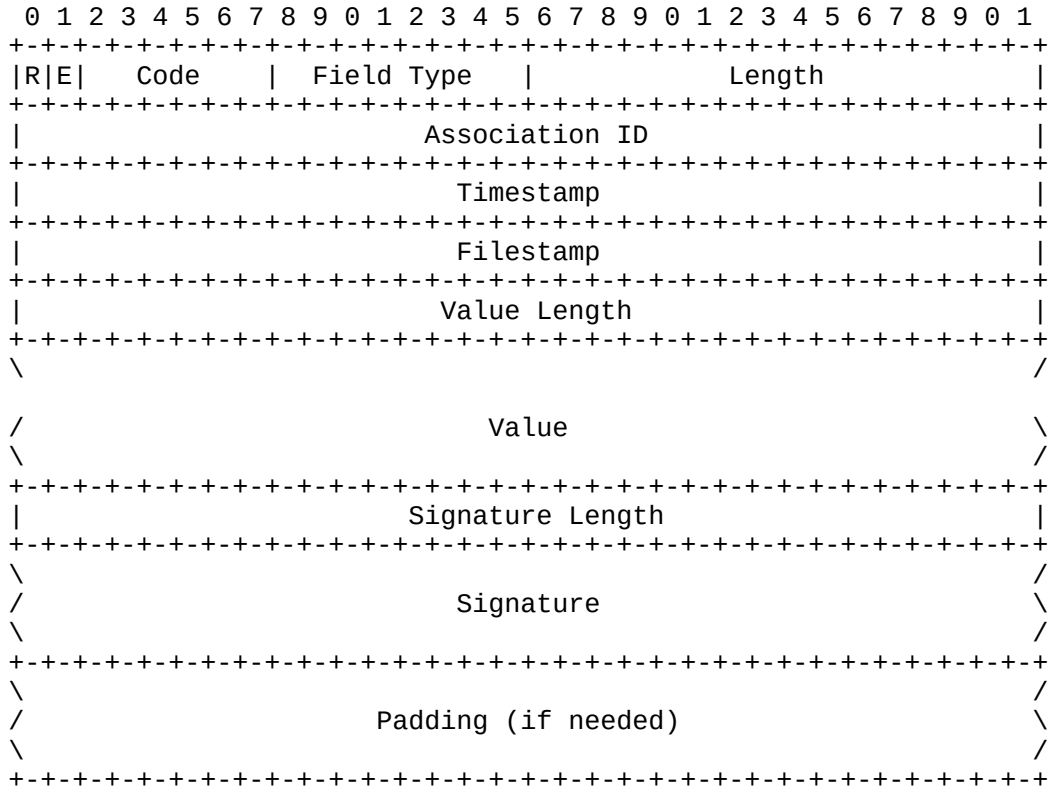


Figure 7: NTPv4 Extension Field Format

While each extension field is zero-padded to a 4-octet (word) boundary, the entire extension is not word-aligned. The Length field covers the entire extension field, including the Length and Padding fields. While the minimum field length is 8 octets, a maximum field length remains to be established. The reference implementation discards any packet with a field length more than 1024 octets.

One or more extension fields follow the NTP packet header and the last followed by the MAC. The extension field parser initializes a pointer to the first octet beyond the NTP packet header and calculates the number of octets remaining to the end of the packet. If the remaining length is 20 (128-bit digest plus 4-octet key ID) or 22 (160-bit digest plus 4-octet key ID), the remaining data are the MAC and parsing is complete. If the remaining length is greater than 22, an extension field is present. If the remaining length is less than 8 or not a multiple of 4, a format error has occurred and the packet is discarded; otherwise, the parser increments the pointer by the extension field length and then uses the same rules as above to determine whether a MAC is present or another extension field.

In Autokey the 8-bit Field Type field is interpreted as the version number, currently 2. For future versions, values 1-7 have been reserved for Autokey; other values may be assigned for other applications. The 6-bit Code field specifies the request or response operation. There are two flag bits: bit 0 is the Response Flag (R) and bit 1 is the Error Flag (E); the Reserved field is unused and should be set to 0. The remaining fields will be described later.

In the most common protocol operations, a client sends a request to a server with an operation code specified in the Code field and both the R bit and E bit dim. The server returns a response with the same operation code in the Code field and lights the R bit. The server can also light the E bit in case of error. Note that it is not necessarily a protocol error to send an unsolicited response with no matching request. If the R bit is dim, the client sets the Association ID field to the client association ID, which the server returns for verification. If the two values do not match, the response is discarded as if never sent. If the R bit is lit, the Association ID field is set to the server association ID obtained in the initial protocol exchange. If the Association ID field does not match any mobilized association ID, the request is discarded as if never sent.

In some cases, not all fields may be present. For requests, until a client has synchronized to a proventic source, signatures are not valid. In such cases, the Timestamp field and Signature Length field (which specifies the length of the Signature) are zero and the Signature field is absent. Some request and error response messages carry no value or signature fields, so in these messages only the first two words (8 octets) are present.

The Timestamp and Filestamp words carry the seconds field of an NTP timestamp. The timestamp establishes the signature epoch of the data field in the message, while the filestamp establishes the generation epoch of the file that ultimately produced the data that is signed.

A signature and timestamp are valid only when the signing host is synchronized to a proventic source; otherwise, the timestamp is zero. A cryptographic data file can only be generated if a signature is possible; otherwise, the filestamp is zero, except in the ASSOC response message, where it contains the server status word.

As in all other TCP/IP protocol designs, all data are sent in network byte order. Unless specified otherwise in the descriptions to follow, the data referred to are stored in the Value field. The Value Length field specifies the length of the data in the Value field.

10.1. No-Operation

A No-operation request (Code 0) does nothing except return an empty response, which can be used as a crypto-ping.

10.2. Association Message (ASSOC)

An Association Message (Code 1) is used in the parameter exchange to obtain the host name and status word. The request contains the client status word in the Filestamp field and the Autokey host name in the Value field. The response contains the server status word in the Filestamp field and the Autokey host name in the Value field. The Autokey host name is not necessarily the DNS host name. A valid response lights the ENAB bit and possibly others in the association status word.

When multiple identity schemes are supported, the host status word determines which ones are available. In server and symmetric modes, the response status word contains bits corresponding to the supported schemes. In all modes, the scheme is selected based on the client identity parameters that are loaded at startup.

10.3. Certificate Message (CERT)

A Certificate Message (Code 2) is used in the certificate exchange to obtain a certificate by subject name. The request contains the subject name; the response contains the certificate encoded in X.509 format with ASN.1 syntax as described in Appendix H.

If the subject name in the response does not match the issuer name, the exchange continues with the issuer name replacing the subject name in the request. The exchange continues until a trusted, self-signed certificate is found and lights the CERT bit in the association status word.

10.4. Cookie Message (COOKIE)

The Cookie Message (Code 3) is used in server and symmetric modes to obtain the server cookie. The request contains the host public key encoded with ASN.1 syntax as described in Appendix H. The response contains the cookie encrypted by the public key in the request. A valid response lights the COOKIE bit in the association status word.

10.5. Autokey Message (AUTO)

The Autokey Message (Code 4) is used to obtain the autokey values. The request contains no value for a client or the autokey values for a symmetric peer. The response contains two 32-bit words, the first is the final key ID, while the second is the index of the final key ID. A valid response lights the AUTO bit in the association status word.

10.6. Leapseconds Values Message (LEAP)

The Leapseconds Values Message (Code 5) is used to obtain the leapseconds values as parsed from the leapseconds table from the National Institute of Standards and Technology (NIST). The request contains no values. The response contains three 32-bit integers: first the NTP seconds of the latest leap event followed by the NTP seconds when the latest NIST table expires and then the TAI offset following the leap event. A valid response lights the LEAP bit in the association status word.

10.7. Sign Message (SIGN)

The Sign Message (Code 6) requests that the server sign and return a certificate presented in the request. The request contains the client certificate encoded in X.509 format with ASN.1 syntax as described in Appendix H. The response contains the client certificate signed by the server private key. A valid response lights the SIGN bit in the association status word.

10.8. Identity Messages (IFF, GQ, MV)

The Identity Messages (Code 7 (IFF), 8 (GQ), or 9 (MV)) contains the client challenge, usually a 160- or 512-bit nonce. The response contains the result of the mathematical operation defined in Appendix B. The Response is encoded in ASN.1 syntax as described in Appendix H. A valid response lights the VRFY bit in the association status word.

11. Autokey State Machine

This section describes the formal model of the Autokey state machine, its state variables and the state transition functions.

11.1. Status Word

The server implements a host status word, while each client implements an association status word. These words have the format and content shown in Figure 8. The low-order 16 bits of the status word define the state of the Autokey dance, while the high-order 16 bits specify the Numerical Identifier (NID) as generated by the OpenSSL library of the OID for one of the message digest/signature encryption schemes defined in [RFC3279]. The NID values for the digest/signature algorithms defined in RFC 3279 are as follows:

Algorithm	OID	NID
pkcs-1	1.2.840.113549.1.1	2
md2	1.2.840.113549.2.2	3
md5	1.2.840.113549.2.5	4
rsaEncryption	1.2.840.113549.1.1.1	6
md2WithRSAEncryption	1.2.840.113549.1.1.2	7
md5WithRSAEncryption	1.2.840.113549.1.1.4	8
id-sha1	1.3.14.3.2.26	64
sha-1WithRSAEncryption	1.2.840.113549.1.1.5	65
id-dsa-wth-sha1	1.2.840.10040.4.3	113
id-dsa	1.2.840.10040.4.1	116

Bits 24-31 are reserved for server use, while bits 16-23 are reserved for client use. In the host portion, bits 24-27 specify the available identity schemes, while bits 28-31 specify the server capabilities. There are two additional bits implemented separately.

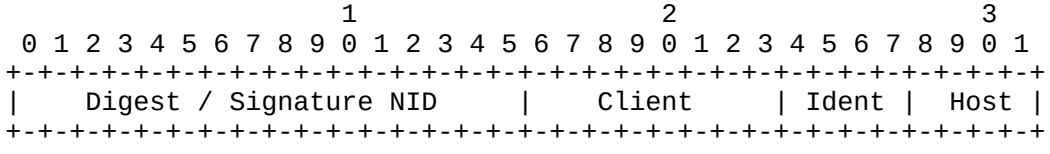


Figure 8: Status Word

The host status word is included in the ASSOC request and response messages. The client copies this word to the association status word and then lights additional bits as the dance proceeds. Once enabled, these bits ordinarily never become dark unless a general reset occurs and the protocol is restarted from the beginning.

The host status bits are defined as follows:

- ENAB (31) is lit if the server implements the Autokey protocol.
- LVAL (30) is lit if the server has installed leapseconds values, either from the NIST leapseconds file or from another server.
- Bits (28-29) are reserved - always dark.
- Bits 24-27 select which server identity schemes are available. While specific coding for various schemes is yet to be determined, the schemes available in the reference implementation and described in Appendix B include the following:
  - none - Trusted Certificate (TC) Scheme (default).
  - PC (27) Private Certificate Scheme.
  - IFF (26) Schnorr aka Identify-Friendly-or-Foe Scheme.
  - GQ (25) Guillard-Quisquater Scheme.
  - MV (24) Mu-Varadharajan Scheme.
- The PC scheme is exclusive of any other scheme. Otherwise, the IFF, GQ, and MV bits can be enabled in any combination.

The association status bits are defined as follows:

- CERT (23): Lit when the trusted host certificate and public key are validated.
- VRFY (22): Lit when the trusted host identity credentials are confirmed.
- PROV (21): Lit when the server signature is verified using its public key and identity credentials. Also called the proventic bit elsewhere in this memo. When enabled, signed values in subsequent messages are presumed proventic.

- COOK (20): Lit when the cookie is received and validated. When lit, key lists with nonzero cookies are generated; when dim, the cookie is zero.

- AUTO (19): Lit when the autokey values are received and validated. When lit, clients can validate packets without extension fields according to the autokey sequence.

- SIGN (18): Lit when the host certificate is signed by the server.

- LEAP (17): Lit when the leapseconds values are received and validated.

- Bit 16: Reserved - always dark.

There are three additional bits: LIST, SYNC, and PEER not included in the association status word. LIST is lit when the key list is regenerated and dim when the autokey values have been transmitted. This is necessary to avoid livelock under some conditions. SYNC is lit when the client has synchronized to a proventic source and never dim after that. PEER is lit when the server has synchronized, as indicated in the NTP header, and never dim after that.

11.2. Host State Variables

The following is a list of host state variables.

Host Name:	The name of the host, by default the string returned by the Unix gethostname() library function. In the reference implementation, this is a configurable value.
Host Status Word:	This word is initialized when the host first starts up. The format is described above.
Host Key:	The RSA public/private key pair used to encrypt/decrypt cookies. This is also the default sign key.
Sign Key:	The RSA or Digital Signature Algorithm (DSA) public/private key pair used to encrypt/decrypt signatures when the host key is not used for this purpose.
Sign Digest:	The message digest algorithm used to compute the message digest before encryption.

IFF Parameters: The parameters used in the optional IFF identity scheme described in Appendix B.

GQ Parameters: The parameters used in the optional GQ identity scheme described in Appendix B.

MV Parameters: The parameters used in the optional MV identity scheme described in Appendix B.

Server Seed: The private value hashed with the IP addresses and key identifier to construct the cookie.

CIS: Certificate Information Structure. This structure includes certain information fields from an X.509v3 certificate, together with the certificate itself. The fields extracted include the subject and issuer names, subject public key and message digest algorithm (pointers), and the beginning and end of the valid period in NTP seconds.

The certificate itself is stored as an extension field in network byte order so it can be copied intact to the message. The structure is signed using the sign key and carries the public values timestamp at signature time and the filestamp of the original certificate file. The structure is used by the CERT response message and SIGN request and response messages.

A flags field in the CIS determines the status of the certificate. The field is encoded as follows:

* TRUST (0x01) - The certificate has been signed by a trusted issuer. If the certificate is self-signed and contains "trustRoot" in the Extended Key Usage field, this bit is lit when the CIS is constructed.

* SIGN (0x02) - The certificate signature has been verified. If the certificate is self-signed and verified using the contained public key, this bit is lit when the CIS is constructed.

* VALID (0x04) - The certificate is valid and can be used to verify signatures. This bit is lit when a trusted certificate has been found on a valid certificate trail.

* PRIV (0x08) - The certificate is private and not to be revealed. If the certificate is self-signed and contains "Private" in the Extended Key Usage field, this bit is lit when the CIS is constructed.

* ERROR (0x80) - The certificate is defective and not to be used in any way.

Certificate List: CIS structures are stored on the certificate list in order of arrival, with the most recently received CIS placed first on the list. The list is initialized with the CIS for the host certificate, which is read from the host certificate file. Additional CIS entries are added to the list as certificates are obtained from the servers during the certificate exchange. CIS entries are discarded if overtaken by newer ones.

The following values are stored as an extension field structure in network byte order so they can be copied intact to the message. They are used to send some Autokey requests and responses. All but the Host Name Values structure are signed using the sign key and all carry the public values timestamp at signature time.

Host Name Values: This is used to send ASSOC request and response messages. It contains the host status word and host name.

Public Key Values: This is used to send the COOKIE request message. It contains the public encryption key used for the COOKIE response message.

Leapseconds Values: This is used to send the LEAP response message. It contains the leapseconds values in the LEAP message description.

11.3. Client State Variables (all modes)

The following is a list of state variables used by the various dances in all modes.

Association ID:	The association ID used in responses. It is assigned when the association is mobilized.
Association Status Word:	The status word copied from the ASSOC response; subsequently modified by the state machine.
Subject Name:	The server host name copied from the ASSOC response.
Issuer Name:	The host name signing the certificate. It is extracted from the current server certificate upon arrival and used to request the next host on the certificate trail.
Server Public Key:	The public key used to decrypt signatures. It is extracted from the server host certificate.
Server Message Digest:	The digest/signature scheme determined in the parameter exchange.
Group Key:	A set of values used by the identity exchange. It identifies the cryptographic compartment shared by the server and client.
Receive Cookie Values:	The cookie returned in a COOKIE response, together with its timestamp and filestamp.
Receive Autokey Values:	The autokey values returned in an AUTO response, together with its timestamp and filestamp.
Send Autokey Values:	The autokey values with signature and timestamps.

Key List:	A sequence of key IDs starting with the autokey seed and each pointing to the next. It is computed, timestamped, and signed at the next poll opportunity when the key list becomes empty.
Current Key Number:	The index of the entry on the Key List to be used at the next poll opportunity.

11.4. Protocol State Transitions

The protocol state machine is very simple but robust. The state is determined by the client status word bits defined above. The state transitions of the three dances are shown below. The capitalized truth values represent the client status bits. All bits are initialized as dark and are lit upon the arrival of a specific response message as detailed above.

11.4.1. Server Dance

The server dance begins when the client sends an ASSOC request to the server. The clock is updated when PREV is lit and the dance ends when LEAP is lit. In this dance, the autokey values are not used, so an autokey exchange is not necessary. Note that the SIGN and LEAP requests are not issued until the client has synchronized to a proventic source. Subsequent packets without extension fields are validated by the autokey sequence. This example and others assumes the IFF identity scheme has been selected in the parameter exchange.

```
1       while (1) {
2               wait_for_next_poll;
3               make_NTP_header;
4               if (response_ready)
5                       send_response;
6               if (!ENB)             /* parameter exchange */
7                       ASSOC_request;
8               else if (!CERT)       /* certificate exchange */
9                       CERT_request(Host_Name);
10              else if (!IFF)        /* identity exchange */
11                      IFF_challenge;
12              else if (!COOK)       /* cookie exchange */
13                      COOKIE_request;
14              else if (!SYNC)       /* wait for synchronization */
15                      continue;
16              else if (!SIGN)       /* sign exchange */
17                      SIGN_request(Host_Certificate);
18              else if (!LEAP)       /* leapsecond values exchange */
19                      LEAP_request;
20              send packet;
21      }
```

Figure 9: Server Dance

If the server refreshes the private seed, the cookie becomes invalid. The server responds to an invalid cookie with a crypto-NAK message, which causes the client to restart the protocol from the beginning.

11.4.2. Broadcast Dance

The broadcast dance is similar to the server dance with the cookie exchange replaced by the autokey values exchange. The broadcast dance begins when the client receives a broadcast packet including an ASSOC response with the server association ID. This mobilizes a client association in order to proventicate the source and calibrate the propagation delay. The dance ends when the LEAP bit is lit, after which the client sends no further packets. Normally, the broadcast server includes an ASSOC response in each transmitted packet. However, when the server generates a new key list, it includes an AUTO response instead.

In the broadcast dance, extension fields are used with every packet, so the cookie is always zero and no cookie exchange is necessary. As in the server dance, the clock is updated when PREV is lit and the

dance ends when LEAP is lit. Note that the SIGN and LEAP requests are not issued until the client has synchronized to a proventic source. Subsequent packets without extension fields are validated by the autokey sequence.

```
1       while (1) {
2               wait_for_next_poll;
3               make_NTP_header;
4               if (response_ready)
5                       send_response;
6               if (!ENB)               /* parameters exchange */
7                       ASSOC_request;
8               else if (!CERT)         /* certificate exchange */
9                       CERT_request(Host_Name);
10              else if (!IFF)          /* identity exchange */
11                      IFF_challenge;
12              else if (!AUT)          /* autokey values exchange */
13                      AUTO_request;
14              else if (!SYNC)         /* wait for synchronization */
15                      continue;
16              else if (!SIGN)         /* sign exchange */
17                      SIGN_request(Host_Certificate);
18              else if (!LEAP)         /* leapsecond values exchange */
19                      LEAP_request;
20              send NTP_packet;
21      }
```

Figure 10: Broadcast Dance

If a packet is lost and the autokey sequence is broken, the client hashes the current autokey until either it matches the previous autokey or the number of hashes exceeds the count given in the autokey values. If the latter, the client sends an AUTO request to retrieve the autokey values. If the client receives a crypto-NAK during the dance, or if the association ID changes, the client restarts the protocol from the beginning.

11.4.3. Symmetric Dance

The symmetric dance is intricately choreographed. It begins when the active peer sends an ASSOC request to the passive peer. The passive peer mobilizes an association and both peers step a three-way dance where each peer completes a parameter exchange with the other. Until one of the peers has synchronized to a proventic source (which could be the other peer) and can sign messages, the other peer loops waiting for a valid timestamp in the ensuing CERT response.

```
1       while (1) {
2               wait_for_next_poll;
3               make_NTP_header;
4               if (!ENB)             /* parameters exchange */
5                       ASSOC_request;
6               else if (!CERT)       /* certificate exchange */
7                       CERT_request(Host_Name);
8               else if (!IFF)        /* identity exchange */
9                       IFF_challenge;
10              else if (!COOK && PEER) /* cookie exchange */
11                      COOKIE_request);
12              else if (!AUTO)       /* autokey values exchange */
13                      AUTO_request;
14              else if (LIST)        /* autokey values response */
15                      AUTO_response;
16              else if (!SYNC)       /* wait for synchronization */
17                      continue;
18              else if (!SIGN)       /* sign exchange */
19                      SIGN_request;
20              else if (!LEAP)       /* leapsecond values exchange */
21                      LEAP_request;
22              send NTP_packet;
23      }
```

Figure 11: Symmetric Dance

Once a peer has synchronized to a proventic source, it includes timestamped signatures in its messages. The other peer, which has been stalled waiting for valid timestamps, now mates the dance. It retrieves the now nonzero cookie using a cookie exchange and then the updated autokey values using an autokey exchange.

As in the broadcast dance, if a packet is lost and the autokey sequence broken, the peer hashes the current autokey until either it matches the previous autokey or the number of hashes exceeds the count given in the autokey values. If the latter, the client sends an AUTO request to retrieve the autokey values. If the peer receives a crypto-NAK during the dance, or if the association ID changes, the peer restarts the protocol from the beginning.

11.5. Error Recovery

The Autokey protocol state machine includes provisions for various kinds of error conditions that can arise due to missing files, corrupted data, protocol violations, and packet loss or misorder, not to mention hostile intrusion. This section describes how the protocol responds to reachability and timeout events that can occur due to such errors.

A persistent NTP association is mobilized by an entry in the configuration file, while an ephemeral association is mobilized upon the arrival of a broadcast or symmetric active packet with no matching association. Subsequently, a general reset reinitializes all association variables to the initial state when first mobilized. In addition, if the association is ephemeral, the association is demobilized and all resources acquired are returned to the system.

Every NTP association has two variables that maintain the liveness state of the protocol, the 8-bit reach register and the unreach counter defined in [RFC5905]. At every poll interval, the reach register is shifted left, the low order bit is dimmed and the high order bit is lost. At the same time, the unreach counter is incremented by one. If an arriving packet passes all authentication and sanity checks, the rightmost bit of the reach register is lit and the unreach counter is set to zero. If any bit in the reach register is lit, the server is reachable; otherwise, it is unreachable.

When the first poll is sent from an association, the reach register and unreach counter are set to zero. If the unreach counter reaches 16, the poll interval is doubled. In addition, if association is persistent, it is demobilized. 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 packet sent at each poll interval until a valid response is received or a general reset occurs, in which case the protocol restarts from the beginning. A general reset also occurs for an association when an unrecoverable protocol error occurs. A general reset occurs for all associations when the system clock is first synchronized or the clock is stepped or when the server seed is refreshed.

There are special cases designed to quickly respond to broken associations, such as when a server restarts or refreshes keys. Since the client cookie is invalidated, the server rejects the next client request and returns a crypto-NAK packet. Since the crypto-NAK has no MAC, the problem for the client is to determine whether it is legitimate or the result of intruder mischief. In order to reduce the vulnerability in such cases, the crypto-NAK, as well as all responses, is believed only if the result of a previous packet sent by the client and not a replay, as confirmed by the NTP on-wire protocol. While this defense can be easily circumvented by a man-in-the-middle, it does deflect other kinds of intruder warfare.

There are a number of situations where some event happens that causes the remaining autokeys on the key list to become invalid. When one of these situations happens, the key list and associated autokeys in

the key cache are purged. A new key list, signature, and timestamp are generated when the next NTP message is sent, assuming there is one. The following is a list of these situations:

1. When the cookie value changes for any reason.

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

3. 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.

12. Security Considerations

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

12.1. Protocol Vulnerability

While the protocol has not been subjected to a formal analysis, a few preliminary assertions can be made. In the client/server and symmetric dances, the underlying NTP on-wire protocol is resistant to lost, duplicate, and bogus packets, even if the clock is not synchronized, so the protocol is not vulnerable to a wiretapper attack. The on-wire protocol is resistant to replays of both the client request packet and the server reply packet. A man-in-the-middle attack, even if it could simulate a valid cookie, could not prove identity.

In the broadcast dance, the client begins with a volley in client/server mode to obtain the autokey values and signature, so has the same protection as in that mode. When continuing in receive-only mode, a wiretapper cannot produce a key list with valid signed autokey values. If it replays an old packet, the client will reject it by the timestamp check. The most it can do is manufacture a future packet causing clients to repeat the autokey hash operations until exceeding the maximum key number. If this happens the broadcast client temporarily reverts to client mode to refresh the autokey values.

By assumption, a man-in-the-middle attacker that intercepts a packet cannot break the wire or delay an intercepted packet. If this assumption is removed, the middleman could intercept a broadcast packet and replace the data and message digest without detection by the clients.

As mentioned previously in this memo, the TC identity scheme is vulnerable to a man-in-the-middle attack where an intruder could create a bogus certificate trail. To foil this kind of attack, either the PC, IFF, GQ, or MV identity schemes must be used.

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

12.2. Clogging Vulnerability

A self-induced clogging incident cannot happen, since signatures are computed only when the data have changed and the data do not change very often. 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 one of them is updated during a dance or the server seed is refreshed, which happens about once per day.

There are two clogging vulnerabilities exposed in the protocol design: an encryption attack where the intruder hopes to clog the victim server with needless cryptographic calculations, and a decryption attack where the intruder attempts to clog the victim client with needless cryptographic calculations. Autokey uses public key cryptography and the algorithms that perform these functions consume significant resources.

In client/server and peer dances, an encryption hazard exists when a wiretapper replays prior cookie request messages at speed. There is no obvious way to deflect such attacks, as the server retains no state between requests. Replays of cookie request or response messages are detected and discarded by the client on-wire protocol.

In broadcast mode, a decryption hazard exists when a wiretapper replays autokey response messages at speed. Once synchronized to a proventic source, a legitimate extension field with timestamp the

same as or earlier than the most recently received of that type is immediately discarded. This foils a man-in-the-middle cut-and-paste attack using an earlier response, for example. A legitimate extension field with timestamp in the future is unlikely, as that would require predicting the autokey sequence. However, this causes the client to refresh and verify the autokey values and signature.

A determined attacker can destabilize the on-wire protocol or an Autokey dance in various ways by replaying old messages before the client or peer has synchronized for the first time. For instance, replaying an old symmetric mode message before the peers have synchronize will prevent the peers from ever synchronizing. Replaying out of order Autokey messages in any mode during a dance could prevent the dance from ever completing. There is nothing new in these kinds of attack; a similar vulnerability even exists in TCP.

13. IANA Consideration

The IANA has added the following entries to the NTP Extensions Field Types registry:

Field Type	Meaning
0x0002	No-Operation Request
0x8002	No-Operation Response
0xC002	No-Operation Error Response
0x0102	Association Message Request
0x8102	Association Message Response
0xC102	Association Message Error Response
0x0202	Certificate Message Request
0x8202	Certificate Message Response
0xC202	Certificate Message Error Response
0x0302	Cookie Message Request
0x8302	Cookie Message Response
0xC302	Cookie Message Error Response
0x0402	Autokey Message Request
0x8402	Autokey Message Response
0xC402	Autokey Message Error Response
0x0502	Leapseconds Value Message Request
0x8502	Leapseconds Value Message Response
0xC502	Leapseconds Value Message Error Response
0x0602	Sign Message Request
0x8602	Sign Message Response
0xC602	Sign Message Error Response
0x0702	IFF Identity Message Request
0x8702	IFF Identity Message Response
0xC702	IFF Identity Message Error Response
0x0802	GQ Identity Message Request
0x8802	GQ Identity Message Response
0xC802	GQ Identity Message Error Response
0x0902	MV Identity Message Request
0x8902	MV Identity Message Response
0xC902	MV Identity Message Error Response

14. References

14.1. Normative References

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

14.2. Informative References

[DASBUCH] Mills, D., "Computer Network Time Synchronization - the Network Time Protocol", 2006.

[GUILLOU] Guillou, L. and J. Quisquatar, "A "paradoxical" identity-based signature scheme resulting from zero-knowledge", 1990.

[MV] Mu, Y. and V. Varadharajan, "Robust and secure broadcasting", 2001.

[RFC1305] Mills, D., "Network Time Protocol (Version 3) Specification, Implementation", RFC 1305, March 1992.

[RFC2412] Orman, H., "The OAKLEY Key Determination Protocol", RFC 2412, November 1998.

[RFC2522] Karn, P. and W. Simpson, "Photuris: Session-Key Management Protocol", RFC 2522, March 1999.

[RFC2875] Prafullchandra, H. and J. Schaad, "Diffie-Hellman Proof-of-Possession Algorithms", RFC 2875, July 2000.

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, April 2002.

[RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, September 2005.

[RFC4302] Kent, S., "IP Authentication Header", RFC 4302, December 2005.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

[RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[SCHNORR] Schnorr, C., "Efficient signature generation for smart cards", 1991.

[STINSON] Stinson, D., "Cryptography - Theory and Practice", 1995.

Appendix A. Timestamps, Filestamps, and Partial Ordering

When the host starts, it reads the host key and host certificate files, which are required for continued operation. It also reads the sign key and leapseconds values, when available. When reading these files, the host checks the file formats and filestamps for validity; for instance, all filestamps must be later than the time the UTC timescale was established in 1972 and the certificate filestamp must not be earlier than its associated sign key filestamp. At the time the files are read, the host is not synchronized, so it cannot determine whether the filestamps are bogus other than by using these simple checks. It must not produce filestamps or timestamps until synchronized to a proventic source.

In the following, the relation A --> B is Lamport's "happens before" relation, which is true if event A happens before event B. When timestamps are compared to timestamps, the relation is false if A <--> B; that is, false if the events are simultaneous. For timestamps compared to filestamps and filestamps compared to filestamps, the relation is true if A <--> B. Note that the current time plays no part in these assertions except in (6) below; however, the NTP protocol itself ensures a correct partial ordering for all current time values.

The following assertions apply to all relevant responses:

1. The client saves the most recent timestamp T0 and filestamp F0 for the respective signature type. For every received message carrying timestamp T1 and filestamp F1, the message is discarded unless T0 --> T1 and F0 --> F1. The requirement that T0 --> T1 is the primary defense against replays of old messages.

2. For timestamp T and filestamp F, F --> T; that is, the filestamp must happen before the timestamp. If not, this could be due to a file generation error or a significant error in the system clock time.

3. For sign key filestamp S, certificate filestamp C, cookie timestamp D and autokey timestamp A, S --> C --> D --> A; that is, the autokey must be generated after the cookie, the cookie after the certificate, and the certificate after the sign key.

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

5. A certificate for subject S signed by issuer I and with filestamp C1 obsoletes, but does not necessarily invalidate, another certificate with the same subject and issuer but with filestamp C0, where C0 --> C1.

6. A certificate with begin time B and end time E is invalid and cannot be used to verify signatures if t --> B or E --> t, where t is the current proventic time. Note that the public key previously extracted from the certificate continues to be valid for an indefinite time. This raises the interesting possibility where a truechimer server with expired certificate or a falseticker with valid certificate are not detected until the client has synchronized to a proventic source.

Appendix B. Identity Schemes

There are five identity schemes in the NTPv4 reference implementation: (1) private certificate (PC), (2) trusted certificate (TC), (3) a modified Schnorr algorithm (IFF - Identify Friend or Foe), (4) a modified Guillou-Quisquater (GQ) algorithm, and (5) a modified Mu-Varadharajan (MV) algorithm.

The PC scheme is intended for testing and development and not recommended for general use. The TC scheme uses a certificate trail, but not an identity scheme. The IFF, GQ, and MV identity schemes use a cryptographically strong challenge-response exchange where an intruder cannot learn the group key, even after repeated observations of multiple exchanges. These schemes begin when the client sends a nonce to the server, which then rolls its own nonce, performs a mathematical operation and sends the results to the client. The client performs a second mathematical operation to prove the server has the same group key as the client.

Appendix C. Private Certificate (PC) Scheme

The PC scheme shown in Figure 12 uses a private certificate as the group key.

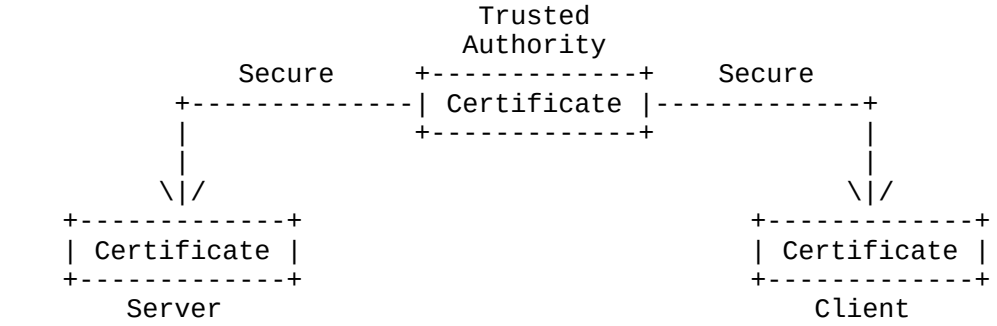


Figure 12: Private Certificate (PC) Identity Scheme

A certificate is designated private when the X.509v3 Extended Key Usage extension field is present and contains "Private". The private certificate is distributed to all other group members by secret means, so in fact becomes a symmetric key. Private certificates are also trusted, so there is no need for a certificate trail or identity scheme.

Appendix D. Trusted Certificate (TC) Scheme

All other schemes involve a conventional certificate trail as shown in Figure 13.

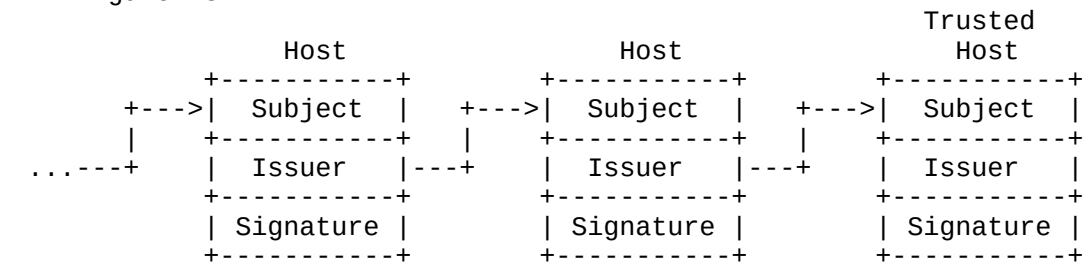


Figure 13: Trusted Certificate (TC) Identity Scheme

As described in RFC 4210 [RFC4210], each certificate is signed by an issuer one step closer to the trusted host, which has a self-signed trusted certificate. A certificate is designated trusted when an X.509v3 Extended Key Usage extension field is present and contains "trustRoot". If no identity scheme is specified in the parameter exchange, this is the default scheme.

Appendix E. Schnorr (IFF) Identity Scheme

The IFF scheme is useful when the group key is concealed, so that client keys need not be protected. The primary disadvantage is that when the server key is refreshed all hosts must update the client key. The scheme shown in Figure 14 involves a set of public parameters and a group key including both private and public components. The public component is the client key.

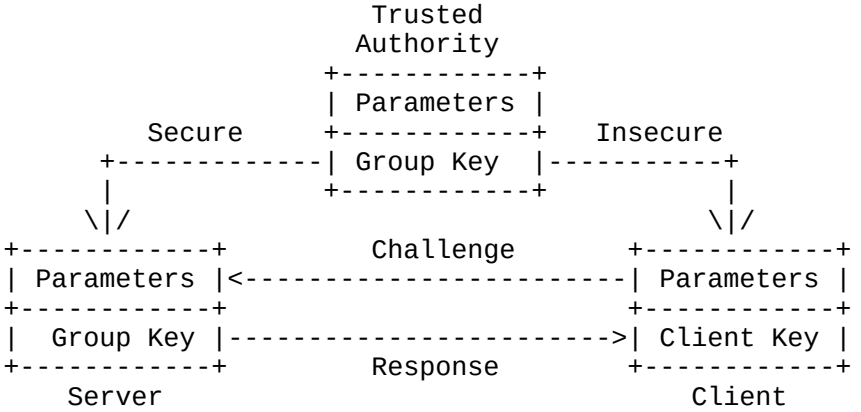


Figure 14: Schnorr (IFF) Identity Scheme

By happy coincidence, the mathematical principles on which IFF is based are similar to DSA. The scheme is a modification an algorithm described in [SCHNORR] and [STINSON] (p. 285). The parameters are generated by routines in the OpenSSL library, but only the moduli p, q and generator g are used. The p is a 512-bit prime, g a generator of the multiplicative group Z_p* and q a 160-bit prime that divides (p-1) and is a qth root of 1 mod p; that is, g^q = 1 mod p. The TA rolls a private random group key b (0 < b < q), then computes public client key v = g^(q-b) mod p. The TA distributes (p, q, g, b) to all servers using secure means and (p, q, g, v) to all clients not necessarily using secure means.

The TA hides IFF parameters and keys in an OpenSSL DSA cuckoo structure. The IFF parameters are identical to the DSA parameters, so the OpenSSL library can be used directly. The structure shown in Figure 15 is written to a file as a DSA private key encoded in PEM. Unused structure members are set to one.

IFF	DSA	Item	Include
p	p	modulus	all
q	q	modulus	all
g	g	generator	all
b	priv_key	group key	server
v	pub_key	client key	client

Figure 15: IFF Identity Scheme Structure

Alice challenges Bob to confirm identity using the following protocol exchange.

1. Alice rolls random r (0 < r < q) and sends to Bob.

2. Bob rolls random k (0 < k < q), computes y = k + br mod q and x = g^k mod p, then sends (y, hash(x)) to Alice.

3. Alice computes z = g^y * v^r mod p and verifies hash(z) equals hash(x).

If the hashes match, Alice knows that Bob has the group key b. Besides making the response shorter, the hash makes it effectively impossible for an intruder to solve for b by observing a number of these messages. The signed response binds this knowledge to Bob’s private key and the public key previously received in his certificate.

Appendix F. Guillard-Quisquater (GQ) Identity Scheme

The GQ scheme is useful when the server key must be refreshed from time to time without changing the group key. The NTP utility programs include the GQ client key in the X.509v3 Subject Key Identifier extension field. The primary disadvantage of the scheme is that the group key must be protected in both the server and client. A secondary disadvantage is that when a server key is refreshed, old extension fields no longer work. The scheme shown in Figure 16 involves a set of public parameters and a group key used to generate private server keys and client keys.

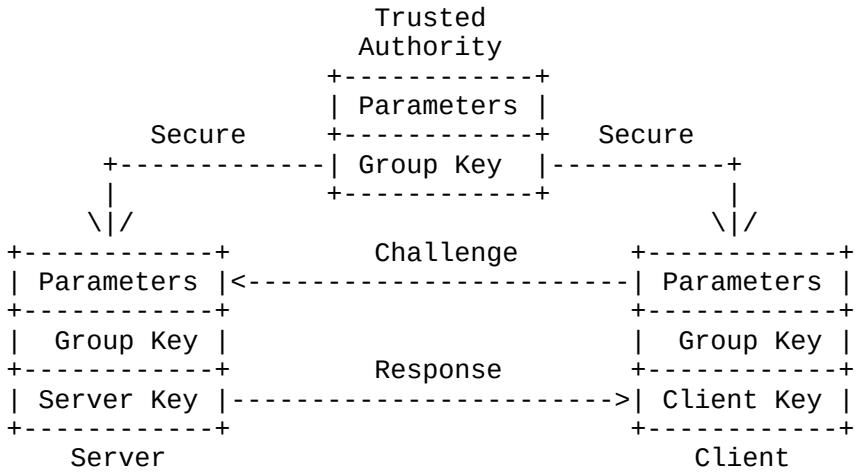


Figure 16: Schnorr (IFF) Identity Scheme

By happy coincidence, the mathematical principles on which GQ is based are similar to RSA. The scheme is a modification of an algorithm described in [GUILLOU] and [STINSON] (p. 300) (with errors). The parameters are generated by routines in the OpenSSL library, but only the moduli p and q are used. The 512-bit public modulus is n=pq, where p and q are secret large primes. The TA rolls random large prime b (0 < b < n) and distributes (n, b) to all group servers and clients using secure means, since an intruder in possession of these values could impersonate a legitimate server. The private server key and public client key are constructed later.

The TA hides GQ parameters and keys in an OpenSSL RSA cuckoo structure. The GQ parameters are identical to the RSA parameters, so the OpenSSL library can be used directly. When generating a certificate, the server rolls random server key u (0 < u < n) and client key its inverse obscured by the group key v = (u^-1)^b mod n. These values replace the private and public keys normally generated by the RSA scheme. The client key is conveyed in a X.509 certificate extension. The updated GQ structure shown in Figure 17 is written as an RSA private key encoded in PEM. Unused structure members are set to one.

GQ	RSA	Item	Include
n	n	modulus	all
b	e	group key	all
u	p	server key	server
v	q	client key	client

Figure 17: GQ Identity Scheme Structure

Alice challenges Bob to confirm identity using the following exchange.

1. Alice rolls random r (0 < r < n) and sends to Bob.

2. Bob rolls random k (0 < k < n) and computes y = ku^r mod n and x = k^b mod n, then sends (y, hash(x)) to Alice.

3. Alice computes z = (v^r)*(y^b) mod n and verifies hash(z) equals hash(x).

If the hashes match, Alice knows that Bob has the corresponding server key u. Besides making the response shorter, the hash makes it effectively impossible for an intruder to solve for u by observing a number of these messages. The signed response binds this knowledge to Bob's private key and the client key previously received in his certificate.

Appendix G. Mu-Varadharajan (MV) Identity Scheme

The MV scheme is perhaps the most interesting and flexible of the three challenge/response schemes, but is devilishly complicated. It is most useful when a small number of servers provide synchronization to a large client population where there might be considerable risk of compromise between and among the servers and clients. The client population can be partitioned into a modest number of subgroups, each associated with an individual client key.

The TA generates an intricate cryptosystem involving encryption and decryption keys, together with a number of activation keys and associated client keys. The TA can activate and revoke individual client keys without changing the client keys themselves. The TA provides to the servers an encryption key E, and partial decryption keys g-bar and g-hat which depend on the activated keys. The servers

have no additional information and, in particular, cannot masquerade as a TA. In addition, the TA provides to each client j individual partial decryption keys x-bar_j and x-hat_j, which do not need to be changed if the TA activates or deactivates any client key. The clients have no further information and, in particular, cannot masquerade as a server or TA.

The scheme uses an encryption algorithm similar to El Gamal cryptography and a polynomial formed from the expansion of product terms (x-x_1)(x-x_2)(x-x_3)...(x-x_n), as described in [MV]. The paper has significant errors and serious omissions. The cryptosystem is constructed so that, for every encryption key E its inverse is (g-bar^x-hat_j)(g-hat^x-bar_j) mod p for every j. This remains true if both quantities are raised to the power k mod p. The difficulty in finding E is equivalent to the discrete log problem.

The scheme is shown in Figure 18. The TA generates the parameters, group key, server keys, and client keys, one for each client, all of which must be protected to prevent theft of service. Note that only the TA has the group key, which is not known to either the servers or clients. In this sense, the MV scheme is a zero-knowledge proof.

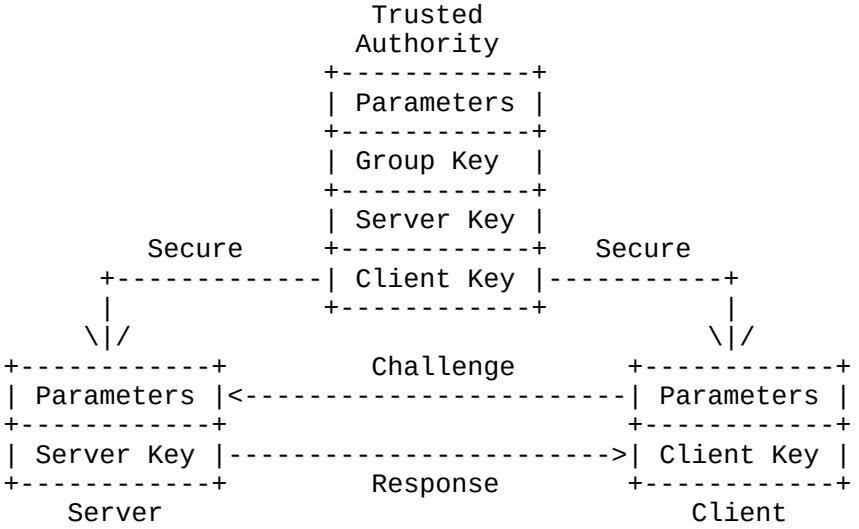


Figure 18: Mu-Varadharajan (MV) Identity Scheme

The TA hides MV parameters and keys in OpenSSL DSA cuckoo structures. The MV parameters are identical to the DSA parameters, so the OpenSSL library can be used directly. The structure shown in the figures below are written to files as a the fkey encoded in PEM. Unused structure members are set to one. The Figure 19 shows the data

structure used by the servers, while Figure 20 shows the client data structure associated with each activation key.

MV	DSA	Item	Include
p	p	modulus	all
q	q	modulus	server
E	g	private encrypt	server
g-bar	priv_key	public decrypt	server
g-hat	pub_key	public decrypt	server

Figure 19: MV Scheme Server Structure

MV	DSA	Item	Include
p	p	modulus	all
x-bar_j	priv_key	public decrypt	client
x-hat_j	pub_key	public decrypt	client

Figure 20: MV Scheme Client Structure

The devil is in the details, which are beyond the scope of this memo. The steps in generating the cryptosystem activating the keys and generating the partial decryption keys are in [DASBUCH] (page 170 ff).

Alice challenges Bob to confirm identity using the following exchange.

1. Alice rolls random r (0 < r < q) and sends to Bob.

2. Bob rolls random k (0 < k < q) and computes the session encryption key E-prime = E^k mod p and partial decryption keys g-bar-prime = g-bar^k mod p and g-hat-prime = g-hat^k mod p. He encrypts x = E-prime * r mod p and sends (x, g-bar-prime, g-hat-prime) to Alice.

3. Alice computes the session decryption key E^-1 = (g-bar-prime)^x-hat_j (g-hat-prime)^x-bar_j mod p and verifies that r = E^-1 x.

Appendix H. ASN.1 Encoding Rules

Certain value fields in request and response messages contain data encoded in ASN.1 distinguished encoding rules (DER). The BNF grammar for each encoding rule is given below along with the OpenSSL routine used for the encoding in the reference implementation. The object identifiers for the encryption algorithms and message digest/signature encryption schemes are specified in [RFC3279]. The particular algorithms required for conformance are not specified in this memo.

Appendix I. COOKIE Request, IFF Response, GQ Response, MV Response

The value field of the COOKIE request message contains a sequence of two integers (n, e) encoded by the i2d_RSAPublicKey() routine in the OpenSSL distribution. In the request, n is the RSA modulus in bits and e is the public exponent.

```
RSAPublicKey ::= SEQUENCE {
        n ::= INTEGER,
        e ::= INTEGER
}
```

The IFF and GQ responses contain a sequence of two integers (r, s) encoded by the i2d_DSA_SIG() routine in the OpenSSL distribution. In the responses, r is the challenge response and s is the hash of the private value.

```
DSAPublicKey ::= SEQUENCE {
        r ::= INTEGER,
        s ::= INTEGER
}
```

The MV response contains a sequence of three integers (p, q, g) encoded by the i2d_DSAparams() routine in the OpenSSL library. In the response, p is the hash of the encrypted challenge value and (q, g) is the client portion of the decryption key.

```
DSAparameters ::= SEQUENCE {
        p ::= INTEGER,
        q ::= INTEGER,
        g ::= INTEGER
}
```

Appendix J. Certificates

Certificate extension fields are used to convey information used by the identity schemes. While the semantics of these fields generally conform with conventional usage, there are subtle variations. The fields used by Autokey version 2 include:

- Basic Constraints. This field defines the basic functions of the certificate. It contains the string "critical,CA:TRUE", which means the field must be interpreted and the associated private key can be used to sign other certificates. While included for compatibility, Autokey makes no use of this field.

- Key Usage. This field defines the intended use of the public key contained in the certificate. It contains the string "digitalSignature,keyCertSign", which means the contained public key can be used to verify signatures on data and other certificates. While included for compatibility, Autokey makes no use of this field.

- Extended Key Usage. This field further refines the intended use of the public key contained in the certificate and is present only in self-signed certificates. It contains the string "Private" if the certificate is designated private or the string "trustRoot" if it is designated trusted. A private certificate is always trusted.

- Subject Key Identifier. This field contains the client identity key used in the GQ identity scheme. It is present only if the GQ scheme is in use.

The value field contains an X.509v3 certificate encoded by the i2d_X509() routine in the OpenSSL distribution. The encoding follows the rules stated in [RFC5280], including the use of X.509v3 extension fields.

```
Certificate ::= SEQUENCE {
        tbsCertificate                  TBSCertificate,
        signatureAlgorithm              AlgorithmIdentifier,
        signatureValue                  BIT STRING
}
```

The signatureAlgorithm is the object identifier of the message digest/signature encryption scheme used to sign the certificate. The signatureValue is computed by the certificate issuer using this algorithm and the issuer private key.

```
TBSCertificate ::= SEQUENCE {
        version                         EXPLICIT v3(2),
        serialNumber                    CertificateSerialNumber,
        signature                       AlgorithmIdentifier,
        issuer                          Name,
        validity                        Validity,
        subject                         Name,
        subjectPublicKeyInfo            SubjectPublicKeyInfo,
        extensions                      EXPLICIT Extensions OPTIONAL
}
```

The serialNumber is an integer guaranteed to be unique for the generating host. The reference implementation uses the NTP seconds when the certificate was generated. The signature is the object identifier of the message digest/signature encryption scheme used to sign the certificate. It must be identical to the signatureAlgorithm.

```
CertificateSerialNumber
SET { ::= INTEGER
        Validity ::= SEQUENCE {
                notBefore               UTCTime,
                notAfter                UTCTime
        }
}
```

The notBefore and notAfter define the period of validity as defined in Appendix B.

```
SubjectPublicKeyInfo ::= SEQUENCE {
        algorithm                       AlgorithmIdentifier,
        subjectPublicKey                BIT STRING
}
```

The AlgorithmIdentifier specifies the encryption algorithm for the subject public key. The subjectPublicKey is the public key of the subject.

```
Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension
Extension ::= SEQUENCE {
        extnID                          OBJECT IDENTIFIER,
        critical                        BOOLEAN DEFAULT FALSE,
        extnValue                       OCTET STRING
}

SET {
        Name ::= SEQUENCE {
                OBJECT IDENTIFIER       commonName
                PrintableString         HostName
        }
}
```

For trusted host certificates, the subject and issuer HostName is the NTP name of the group, while for all other host certificates the subject and issuer HostName is the NTP name of the host. In the reference implementation, if these names are not explicitly specified, they default to the string returned by the Unix gethostname() routine (trailing NUL removed). For other than self-signed certificates, the issuer HostName is the unique DNS name of the host signing the certificate.

It should be noted that the Autokey protocol itself has no provisions to revoke certificates. The reference implementation is purposely restarted about once a week, leading to the regeneration of the certificate and a restart of the Autokey protocol. This restart is not enforced for the Autokey protocol but rather for NTP functionality reasons.

Each group host operates with only one certificate at a time and constructs a trail by induction. Since the group configuration must form an acyclic graph, with roots at the trusted hosts, it does not matter which, of possibly several, signed certificates is used. The reference implementation chooses a single certificate and operates with only that certificate until the protocol is restarted.

Authors' Addresses

Brian Haberman (editor)
The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, MD 20723-6099
US

Phone: +1 443 778 1319
EMail: brian@innovationslab.net

Dr. David L. Mills
University of Delaware
Newark, DE 19716
US

Phone: +1 302 831 8247
EMail: mills@udel.edu