1. Introduction

There are three descriptive web pages included in this document:

1. Autonomous Authentication. This is the root page describing the project. In the status and briefings tree, it has two siblings describing related projects which are not included here. There are a number of related status reports and briefings not included here.

2. Autokey Protocol. This page describes the Autokey security model, protocol and an overview of the protocol algorithms.

3. Identity Schemes. This page describes three cryptographic challenge-response identity schemes implemented for the Autokey protocol. These have specialized uses, such as in national time distribution services, timestamping services and hardened security compartments.

The NTP Version 4 software distribution contains an extensive set of document pages, two of which are included as appendices:

1. Authentication Options. This page documents the various options and modes available to the NTP daemon. There are a number of other documentation pages which discuss minor related issues such as broadcast/multicast dependencies, ephemeral associations, etc., but these pages are not included here.

2. ntp-keygen Program. This page documents the program used to generate encryption keys, signature keys, certificates and identity keys.

The detailed Autokey protocol specification is included as an appendix. This appendix itself includes other appendices describing packet formats, error checking and file formats.

1.1 Research Plan

We consider scenarios where sensors of one kind or another are deployed over some geographic area such as a battlefield or planetary surface. The intended application is for military intelligence, weather surveillance or planetary exploration. However, the sensors might be captured by an enemy or destroyed under a tank tread or by a volcanic eruption. We assume, although this is not required, that the sensors do not know in advance the coordinates of deployment or the orientation of antennas, etc. However, the sensors have onboard computing and storage resources, as well as wireless links with sufficient power and directivity to reach at least a fraction of the other sensors in the deployment. Furthermore, we assume the deployment, individually or collectively, has the means to preprocess the sensor data and communicate to a collection center or centers.

Among the issues of special importance in sensor networks are low detection/intercept probability and battery power conservation. In general, this requires the use of spread spectrum, directional antenna and power management systems. These considerations lead to designs with large spreading gains and relatively low data rates. In any case, the sensors must provide both for data collection and transmission, either directly or indirectly via a neighbor. Considerations of power management suggest that communications between the sensors as a network and the collection center(s) be delegated to only one or a few sensors, but the delegations can change from time to time as individuals die from a rocket or exhausted battery.

We assume the network of sensors must operate autonomously and without prior configuration. Once deployed, they must find each other, determine such things as antenna orientation, power level, code rate and code/spread parameters. They must also determine the network routing, synchronization and other functions necessary for the overall network operation. The use of a broadcast/multicast technology makes these functions simpler and more robust, but is in principle not necessary beyond the initial acquisition or repair stages.

Along with the requirement for autonomous configuration is the requirement for strong security. This assumes a security model with defined protocol and certificate schemes. In conventional wired networks this can be done using symmetric key cryptography or public key cryptography, each with its own advantages and disadvantages. In wireless networks and even more so in sensor networks, these technologies seriously challenge the resource limitations. For example, symmetric key cryptography complicates key distribution and does not scale well. On the other hand, public key cryptography requires significant processing and communication resources.

Our approach to a sensor network security strategy is derived from the autokey and autoconfigure technology developed for NTP and reported previously in management reports and the web. This technology provides autonomous, secure, network configuration and repair with no per-entity configuration. The autokey technology, which is based on a special protocol, provides a secure authentication trail from each entity via the network to previously secured entities. The autoconfigure technology, which is based on an add-drop heuristic, provides automatic network configuration with respect to defined metrics like transmission delay, bandwidth, etc. Both technologies can survive the occasional loss of an entity or addition of a new one.

One thing the autokey/autoconfigure technology lacks is a means to verify certificates which bind the identification values for each sensor to its public key. In the conventional model, certificates are loaded before deployment or obtained after deployment from some kind of directory service. However, the assumptions above preclude loading in advance. In addition, centralized directory services are vulnerable to attack and distributed services require consistency and access protocols that can clog the network resources.

1.2 Approach

In conventional security models an entity is presumed either secured or not and with no provision for some shadings between. However, a sensor network might be seriously challenged where data may be useful even if somehow its authenticity was not completely assured. For instance, the chain of command might be temporarily or permanently broken, yet various portions of the network might continue to believe the sensors that were previously secured. It might also happen that certificate trails cannot always be traced to the source because the network has fragmented.

As in PGP, this suggests two metrics, one ranking the level of trust for sensor reports and another ranking the level of trust for reports from that sensor about other sensors. In generalizing these notions, there might be a need for other related metrics or security indices as well. Just for the purposes of this proposal, we will call this approach the autotrust model. To support it, we need a strawman architecture, protocol and algorithms. A preliminary requirements list might include:

- The architecture must provide a security assessment in real time and without explicit messages to network services other than nearest autoconfigure neighbors.

- The protocol must support a dynamically changing network topology as autoconfigure adds new neighbors and drops old ones.
- The algorithms must not require the use of a centralized database. They must function when a database has fragmented in disconnected segments.
- The intrinsic network primitive in the trust model is signing certificates, which carries an increment in trust depending on the number of signatures and the trust of the signatories.

In a preliminary conceptualization of the autotrust scheme, consider the autoconfigure scheme adapted to autotrust. This scheme uses an add-drop heuristic with a single metric based on synchronization distance, which is generally equivalent to roundtrip delay. The scheme uses an expanding ring search to discover candidate entities, evaluate the metric and either drop them or add them to the current server population, but constrained by a maximum number of servers. In simple, the autoconfigure scheme operates as the common decentralized form of the Bellman-Ford routing algorithm, but with an add-drop neighbor router configuration scheme.

In our concept trust is based on certificates and what has been called a signing party. The sensors with direct links to collection entities might ask a trusted agent to sign their certificates, which are then stored at the sensor and retrieved by a suitably augmented autokey protocol. Autoconfigure neighbors then exchange and sign certificates all the while augmenting their own trust metrics by that of the signatories. A metric is assigned as a function of the protocol and eventually this metric percolates throughout the network. Should the network be degraded or partitioned, the trust metric is recalibrated as the segments reconfigure.

We have suggested two metrics above to represent the level of trust assigned each sensor, but there could be others as well. It is not a large hop of faith to extend the add-drop heuristic to include a multi-metric capability. Central to the success of this scheme is the algorithm that constructs the actual routing/discovery metric from an ensemble of submetrics including delay and possibly several trust indices. Since data transmission and trust transmission might form different routing trees, the add-drop population might include some neighbors for transmitting data and others for trusting sources.

1.3 Deliverables

The point of departure for this work is the final working version of the autokey and autoconfigure implementation for the Network Time Protocol (NTP). It is understood that the time synchronization function of this implementation is only ancillary and the software is used only as an implementation and evaluation vehicle for the proposed work.

The period of performance is from the contract date through the end of September, 2002. The staffing level includes the principal investigator, one graduate student and one undergraduate summer intern. We will need to replace two aging Sun workstations and expect to replace one or more cesium tubes for our three aging cesium oscillators. We will also need to install software and hardware upgrades for our Windows PCs. Finally, we will need to augment our laboratory router configuration to support needed features for virtual networking.

The deliverables will include a final report, web documentation and software suitable for testing and further development in a sensor network environment.

1.4 Statement of Work

3. Evaluate the physical and environmental environments for typical sensor networks, in particular limitations on power, spectrum and data rate.

4. Evaluate the current autokey and autoconfigure protocols with respect to the limitations in (1). Construct strawman scenarios showing the feasibility of possible adaptations.

5. Evaluate possible approaches for a fully hierarchical, distributed security management scheme suitable for sensor networks.

6. Select a likely candidate resulting from (2) and design a protocol suitable for testing in a local environment.

7. Implement the protocol to operate in the NTP test vehicle.

8. Test and evaluate the implementation first in a local network and later in the CAIRN testbed. Write a final report summarizing the lessons learned and test results. Evaluate the performance limitations with respect to the sensor network environment.

2. Autonomous Authentication

The missions considered in this project include autonomous networks that might be deployed from a reconnaissance vehicle over a battlefield or from a space probe over a planetary surface. Once deployed, the network must operate autonomously using an ad-hoc wireless infrastructure as servers are deployed or destroyed or the network is damaged or compromised and then repaired. In the traditional fog of war scenario, servers may be able to communicate directly only with nearby neighbors and in particular may be able to assess trust only intermittently and not always directly from a trusted source.

The goal of this project is to develop and test security protocols which resist accidental or malicious attacks on the servers deployed in the network. They must determine that received messages are authentic; that is, were actually sent by the intended source and not manufactured or modified by an intruder. In addition, they must verify the authenticity of any message using only public information and without requiring external management intervention.

The network is protected by a set of cryptographic values, some of which are instantiated before deployment and some of which are generated when needed after deployment. Probably the most important value is the group key which must be instantiated in each server before deployment. A server proves to another server that it is a legitimate group member if it can prove it knows this value. In addition to the group key, every sensor has a host key used to sign messages and certificates and one or more certificates signed by the host key. While the group key must persist for the lifetime of the group, or at least for the lifetime of the mission, the host key and certificates can be refreshed from time to time.

In our model a subset of servers is endowed by some means as trusted, either directly by command or indirectly by election in case the network becomes fragmented. The remaining servers must authenticate from the trusted servers, directly or indirectly, using only cryptographic values already instantiated. In other words, servers can rely on no help other than already available from other servers via the security protocol.

2.1 Brief Description of Work and Results

Our approach involves a cryptographically sound and efficient methodology for use in sensor networks, as well as other ubiquitous, distributed services deployed in the Internet. As demonstrated in the reports and briefings produced by this project, there is a place for Public-Key Infrastructure (PKI) schemes, but none of these schemes alone satisfies the requirements of a real-time network security model. The Photuris and ISAKMP schemes proposed by the IETF require per-association state variables, which contradicts the principles of the remote procedure call (RPC) paradigm in which servers keep no state for a possibly large population of clients. An evaluation of the PKI model and algorithms as implemented in the OpenSSL cryptographic library leads to the conclusion that any scheme requiring every real-time message to carry a PKI digital signature could be vulnerable to a clogging attack.

We have used the Network Time Protocol (NTP) software and the widely distributed NTP synchronization subnet in the Internet as a testbed for distributed protocol development and testing. Not only does the deployment, configuration and management of the NTP subnet have features in common with other distributed applications, but a synchronization service itself must be an intrinsic feature of the network infrastructure.

While NTP Version 3 contains provisions to authenticate individual servers using symmetric key cryptography, it contains no means for secure keys distribution. Public key cryptography provides for public key certificates that bind the server identification credentials to the associated keys. Using PKI key agreements and digital signatures with large client populations can cause significant performance degradations, especially in time critical applications such as NTP [11]. In addition, there are problems unique to NTP in the interaction between the authentication and synchronization functions, since reliable key management requires reliable lifetime control and good timekeeping, while secure timekeeping requires reliable key management.

A revised security model and authentication scheme called Autokey was proposed in earlier reports and papers cited at the end of this page. It has been evolved and refined over time now in its third generation after the original described in the technical report and Version 1 described in previous Internet Drafts. The protocol has been simplified and made more rugged and stable in the event of network or server disruptions. An outline of the security model is given below; additional details of the model and how the protocol operates is on the Autokey Protocol page

The Autokey security model is based on multiple overlapping security compartments or groups. Each group is assigned a group key by a trusted authority and is then deployed to all group members by secure means. Autokey uses conventional IPSEC certificate trails to provide secure server authentication, but this does not provide protection against masquerade, unless the server identity is verified by other means. Autokey includes a suite of identity verification schemes based in part on zero-knowledge proofs. There are five schemes now implemented to prove identity: (1) private certificates (PC), (2) trusted certificates (TC), (3) a modified Schnorr algorithm (IFF aka Identify Friendly or Foe), (4) a modified Guillou-Quisquater algorithm (GQ), and (5) a modified Mu-Varadharajan algorithm (MV). These are described on the Identity Schemes page.

The cryptographic data used by Autokey are generated by a utility program designed for this purpose. This program, called ntp-keygen in the NTP software distribution, generates several files. The lifetimes of all cryptographic values are carefully managed and frequently refreshed. Ordinarily, key lists are refreshed about once per hour and other public and private values are

refreshed about once per day. The protocol design is specially tailored to make a smooth transition when these values are refreshed and to avoid vulnerabilities due to clogging and replay attacks.

2.2 Leapseconds Table

The National Institute of Science and Technology (NIST) archives an ASCII file containing the epoch for all historic and pending occasions of leap second insertion since 1972. While not strictly a security function, the Autokey scheme provides means to securely retrieve the leapseconds table from a server or peer. At present, the only function provided is to fetch the leapseconds table via the network; the daemon itself makes no use of the values. The latest version of the nanokernel software for SunOS, Alpha, FreeBSD and Linux cited below retrieves the latest TAI offset via NTP and provides this on request to client applications.

2.3 Present Status

Autokey version 2 has been implemented in a wide range of machine architectures and operating systems. It has been tested under actual and simulated attack and recovery scenarios. The current public software distribution for NTPv4 includes Autokey and also a prototype version of the Manycast autonomous configuration scheme described on the companion Autonomous Configuration page. The distribution is available for download at www.ntp.org.

All five identity schemes described above have been implemented and tested. At present, the means to activate which one is used in practice lies in the parameters and keys selected during the key generation process. There remains some testing to explore modes of interoperating when different schemes are used by different clients and servers in the same NTP subnet.

2.4 Future Plans

The Autokey technology research and development process is basically mature, although refinements may be expected as the proof of concept phase continues with prototype testing in the Internet. We believe the technology is ready to exploit in other critical environments such as real sensor networks and critical mission command and control systems. However, what needs to be done first is to advance the standards track process.

The Internet draft on the Autokey protocol specification has been under major revision. The latest draft has been submitted to the IETF as a RFC for review. Upon approval, it will be circulated for comment and proposed as draft standard.

3. Autokey Protocol

The Autokey protocol is based on the public key infrastructure (PKI) algorithms of the OpenSSL library, which includes an assortment of message digest, digital signature and encryption schemes. As in NTPv3, NTPv4 supports symmetric key cryptography using keyed MD5 message digests to detect message modification and sequence numbers (actually timestamps) to avoid replay. In addition, NTPv4 supports timestamped digital signatures and X.509 certificates to verify the source as per common industry practices. It also supports several optional identity schemes based on cryptographic challenge-response algorithms.

What makes Autokey special is the way in which these algorithms are used to deflect intruder attacks while maintaining the integrity and accuracy of the time synchronization function. The detailed design is complicated by the need to provisionally authenticate under conditions when reliable time values have not yet been acquired. Only when the server identities have been confirmed, signatures verified and accurate time values obtained does the Autokey protocol declare success.

The NTP message format has been augmented to include one or more extension fields between the original NTP header and the message authenticator code (MAC). The Autokey protocol exchanges cryptographic values in a manner designed to resist clogging and replay attacks. It uses timestamped digital signatures to sign a session key and then a pseudo-random sequence to bind each session key to the preceding one and eventually to the signature. In this way the expensive signature computations are greatly reduced and removed from the critical code path for constructing accurate time values.

Each session key is hashed from the IPv4 or IPv6 source and destination addresses and key identifier, which are public values, and a cookie which can be a public value or hashed from a private value depending on the mode. The pseudo-random sequence is generated by repeated hashes of these values and saved in a key list. The server uses the key list in reverse order, so as a practical matter the next session key cannot be predicted from the previous one, but the client can verify it using the same hash as the server.

There are three Autokey protocol variants or dances in NTP, one for client/server mode, another for broadcast/multicast mode and a third for symmetric active/passive mode. The Association Management program documentation page provides additional details. For instance, in client/server mode the server keeps no state for each client, but uses a fast algorithm and a private value to regenerate the cookie upon arrival of a client message. A client sends its designated public key to the server, which generates the cookie and sends it to the client encrypted with this key. The client decrypts the cookie using its private key and generates the key list. Session keys from this list are used to generate message authentication codes (MAC) which are checked by the server for the request and by the client for the response. Operational details of this and the remaining modes are given in the Internet Draft cited at the end of this page.

3.1 Certificate Trails

A timestamped 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 security model assumes each client is able to verify the certificate trail to a trusted certificate authority (TA) [1][3], where each ascendant server must prove identity to the immediately descendant client by independent means, such as a credit card number or PIN. While Autokey supports this model by default, in a hierarchical ad-hoc network, especially with server discovery schemes like Manycast, proving identity at each rest stop on the trail must be an intrinsic capability of Autokey itself.

Our model is that every member of a closed group, such as might be operated by a timestamping service, be in possession of a secret group key. This could take the form of a private certificate or one or another identification schemes described in the literature and below. Certificate trails and

identification schemes are at the heart of the NTP security model preventing masquerade and middleman attacks. The Autokey protocol operates to hike the trails and run the identity schemes.

A NTP secure group consists of a number of hosts dynamically assembled as a forest with roots the trusted hosts at the lowest stratum of the group. The trusted hosts do not have to be, but often are, primary (stratum 1) servers. A TA, not necessarily a group host, generates private and public identity values and deploys selected values to the group members using secure means.

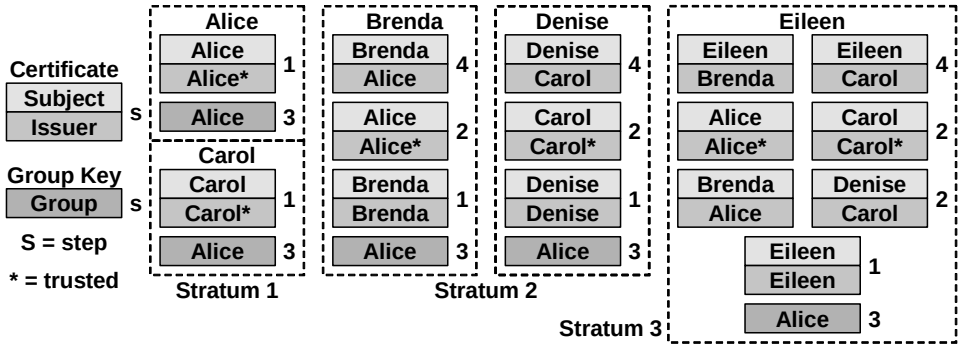

Figure 1. NTP Secure Group

In the above figure the Alice group consists of trusted hosts 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. The certificates are identified by the subject and signed by the issuer. Note that the group key has previously been generated by Alice and deployed by secure means to all group members.

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. At startup each server loads its self-signed certificate from a local file. By convention the lowest stratum server certificates are marked trusted in a X.509 extension field. As Alice and Carol have trusted certificates, they need do nothing further to validate the time. It could be that the trusted hosts depend on servers in other groups; this scenario is discussed later.Brenda, Denise and Eileen start with the Autokey Parameter Exchange, which establishes the server name, signature scheme and identity scheme for each configured server. They continue with the Certificate Exchange, which loads server certificates recursively until a self-signed trusted certificate is found. Brenda and Denise immediately find self-signed trusted certificates for Alice, but Eileen will loop because neither Brenda nor Denise have their own certificates signed by either Alice or Carol.

2. Brenda and Denise continue with the Identity Exchange, which uses one of the identity schemes described below to verify each has the group key previously deployed by Alice. If this succeeds, each continues in step 4.

3. Brenda and Denise present their certificates to Alice for signature. If this succeeds, either or both Brenda and Denise can now provide these signed certificates to Eileen, which may be looping in step 2. When Eileen receives them, she can now follow the trail in either Brenda or Denise to the trusted certificates for Alice and Carol. Once this is done, Eileen can execute the Identity Exchange and Signature Exchange just as Brenda and Denise.

3.2 Secure Groups

The NTP security model is based on multiple overlapping security compartments or groups. The example above illustrates how groups can be used to construct closed compartments, depending on how the identity credentials are deployed. The rules can be summarized:

1. Each host holds a private group key generated by a trusted authority (TA).
2. A host is trusted if it operates at the lowest stratum in the group and has a trusted, self-signed certificate.
3. A host uses the identity scheme to prove to another host it has the same group key.
4. A client verifies group membership if the server has the same key and has an unbroken certificate trail to a trusted host.

Each compartment is assigned a group key by the TA, which is then deployed to all group members by secure means. For various reasons it may be convenient for a server to hold keys for more than one group. For example, The figure below shows three secure groups Alice, Helen and Carol.

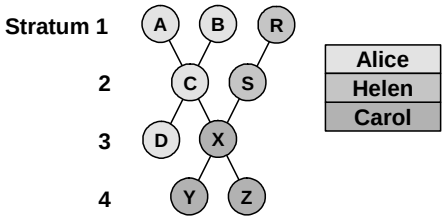

Figure 2. Nested Secure Groups

Hosts A, B, C and D belong to the Alice group, hosts R, S to the Helen group and hosts X, Y and Z to the Carol group. While not strictly necessary, hosts A, B and R are stratum 1 and presumed trusted, but the TA generating the group keys could be one of them or another not shown.

In most identity schemes there are two kinds of group keys, server and client. The intent of the design is to provide security separation, so that servers cannot masquerade as TAs and clients cannot masquerade as servers. Assume for example that Alice and Helen belong to national standards laboratories and their group keys are used to confirm identity between members of each group. Carol is a prominent corporation receiving standards products via broadcast satellite and requiring cryptographic authentication.

Perhaps under contract, host X belonging to the Carol group has rented client keys for both Alice and Helen and has server keys for Carol. The Autokey protocol operates as previously described for each group separately while preserving security separation. Host X prove identity in Carol to clients Y and Z, but cannot prove to anybody that he belongs to either Alice or Helen.

Ordinarily, it would not be desirable to reveal the group key in server keys and forbidden to reveal it in client keys. This can be avoided using the MV identity scheme described later. It allows the same broadcast transmission to be authenticated by more than one key, one used internally by the laboratories (Alice and/or Helen) and the other handed out to clients like Carol. In the MV scheme these keys can be separately activated upon subscription and deactivated if the subscriber fails to pay the bill.

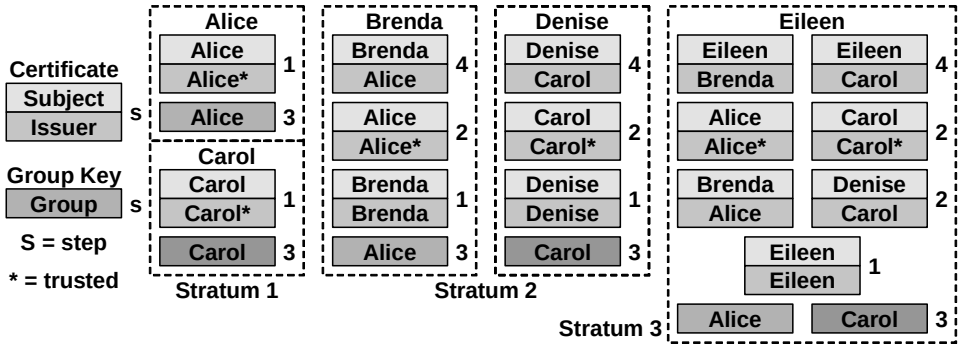

Figure 3. Multiple Secure Groups

The figure above shows operational details where more than one group is involved, in this case Carol and Alice. As in the previous example, Brenda has configured Alice as her time source and Denise has configured Carol as her time source. Alice and Carol have server keys; Brenda and Denise have server and client keys only for their respective groups. Eileen has client keys for both Alice and Carol. The protocol operates as previously described to verify Alice to Brenda and Carol to Denise.

The interesting case is Eileen, who may verify identity either via Brenda or Denise or both. To do that she uses the client keys of both Alice and Carol. But, Eileen doesn't know which of the two keys to use until hiking the certificate trail to find the trusted certificate of either Alice or Carol and then loading the associated local key. This scenario can of course become even more complex as the number of servers and depth of the tree increase. The bottom line is that every host must have the client keys for all the lowest-stratum trusted hosts it is ever likely to find.

3.3 Identity Schemes

While the identity scheme described in RFC-2875 is based on a ubiquitous Diffie-Hellman infrastructure, it is expensive to generate and use when compared to others described here. There are five schemes now implemented in Autokey to prove identity: (1) private certificates (PC), (2) trusted certificates (TC), (3) a modified Schnorr algorithm (IFF aka Identify Friendly or Foe), (4) a modified Guillou-Quisquater algorithm (GQ), and (5) a modified Mu-Varadharajan algorithm

(MV). Following is a summary description of each; details are given on the Identity Schemes page.

Figure 4. PC Identity Scheme

The PC scheme shown above involves the use of a private certificate as group key. A certificate is designated private by a X509 Version 3 extension field when generated by utility routines in the NTP software distribution. The certificate is distributed to all other group members by secure means and is never revealed inside or outside the group. A client is marked trusted in the Parameter Exchange and authentic when the first signature is verified. This scheme is cryptographically strong as long as the private certificate is protected; however, it can be very awkward to refresh the keys or certificate, since new values must be securely distributed to a possibly large population and activated simultaneously.

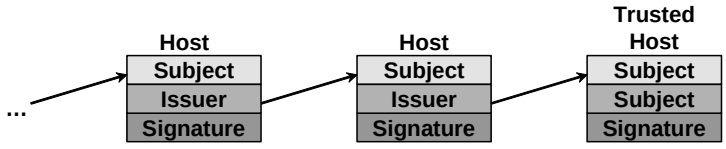

Figure 5. Certificate Trail

All other schemes involve a conventional certificate trail as shown above. As described in RFC-2510, 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 by a X509 Version 3 extension field when generated by utility routines in the NTP software distribution. A host obtains the certificates of all other hosts along the trail leading to a trusted host by the Autokey protocol, then requests the immediately ascendant host to sign its certificate. Subsequently, these certificates are provided to descendent hosts by the Autokey protocol. In this scheme keys and certificates can be refreshed at any time, but a masquerade vulnerability remains unless a request to sign a client certificate is validated by some means such as reverse-DNS. If no specific identity scheme is specified in the Identification Exchange, this is the default TC scheme.

The three remaining schemes IFF, GQ and MV involve a cryptographically strong challenge-response exchange where an intruder cannot learn the group key, even after repeated observations of multiple exchanges. In addition, the IFF and GQ are properly described as zero-knowledge proofs, because the client can verify the server has the group key without the client knowing its value.

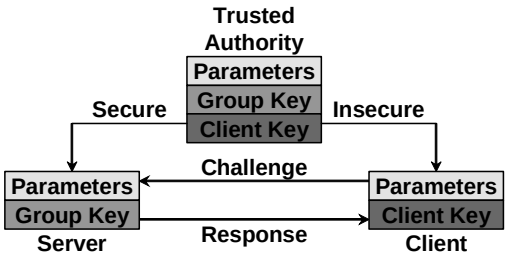

Figure 6. IFF Identity Scheme

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 along with a message digest to the client. The client performs another mathematical operation and verifies the results match the message digest. The IFF scheme shown above is used when the certificate is generated by a third party, such as a commercial service and in general has the same refreshment and distribution problems as PC. However, this scheme has the advantage that the group key is not known to the clients.

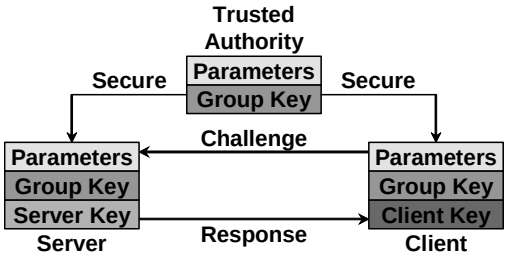

Figure 7. GQ Identity Scheme

On the other hand, when certificates are generated by routines in the NTP distribution, the GQ scheme shown above may be a better choice. In this scheme the server further obscures the secret group key using a public/private key pair which can be refreshed at any time. The public member is conveyed in the certificate by a X509 Version 3 extension field which changes for each regeneration of key pair and certificate.

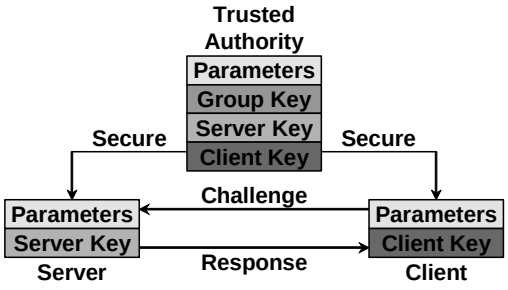

Figure 8. MV Identity Scheme

The MV scheme shown above is perhaps the most interesting and flexible of the three challenge/response schemes. It can be used 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 TA generates an intricate cryptosystem involving public and private

encryption keys, together with a number of activation keys and associated private client decryption keys. The activation keys are used by the TA to activate and revoke individual client decryption keys without changing the decryption keys themselves.

The TA provides the server with a private encryption key and public decryption key. The server adjusts the keys by a nonce for each plaintext encryption, so they appear different on each use. The encrypted ciphertext and adjusted public decryption key are provided in the client message. The client computes the decryption key from its private decryption key and the public decryption key in the message.

3.4 Key Management

The cryptographic data used by Autokey are generated by the ntp-keygen utility program included in the NTP software distribution. This program generates several files, containing MD5 symmetric keys, RSA and DSA public keys, identity group keys and self signed X.509 Version 3 certificates. The certificate format and contents conform to RFC-3280, although with some liberty in the interpretation of extension fields. During generation, a private/public key pair is chosen along with a compatible message digest algorithm. During operation, a client can obtain this and any other certificate held by the server. The client can also request a server acting as a certificate authority to sign and return a certificate.

The lifetimes of all cryptographic values are carefully managed and frequently refreshed. While public keys and certificates have lifetimes that expire only when manually revoked, random session keys have a lifetime specified at the time of generation. Ordinarily, key lists are regenerated about once per hour and other public and private values are refreshed about once per day. Appropriate scripts running from a Unix cron job about once per month can automatically refresh public/private key pairs and certificates without operator intervention. The protocol design is specially tailored to make a smooth transition when these values are refreshed and to avoid vulnerabilities due to clogging and replay attacks.

4. Identity Schemes

This page describes three challenge-response identity schemes based on Schnorr (IFF), Guillou-Quisquater (GQ) and Mu-Varadharajan (MV) cryptosystems. Each scheme involves generating parameters specific to the scheme, together with a secret group key and other public and private values used by the scheme. In order to simplify implementation, each scheme uses existing structures in the OpenSSL library, including those for RSA and DSA cryptography. As these structures are sometimes used in ways far different than their original purpose, they are called cuckoo structures in the descriptions that follow.

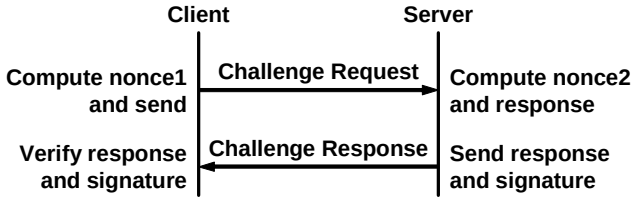

Figure 9. Client-Server Message Exchange

All three schemes operate a challenge-response protocol where client Alice asks server Bob to prove identity relative to a secret group key *b* provided by a trusted authority (TA). As shown in the figure above, client Alice rolls random nonce *r* and sends to server Bob. Bob rolls random nonce *k*, performs some mathematical function and returns the value along with the hash of some private value to Alice. Alice performs another mathematical function and verifies the result matches the hash in Bob's message.

Each scheme is intended for specific use. There are two kinds of keys, server and client. Servers can be clients of other servers, but clients cannot be servers for dependent clients. In general, the goals of the schemes are that clients cannot masquerade as servers and servers cannot masquerade as TAs, but they differ somewhat on how to achieve these goals. To the extent that identity can be verified without revealing the group key, the schemes are properly described as zero-knowledge proofs.

The IFF scheme is intended for servers operated by national laboratories. The servers use a private group key and provide the client key on request. The servers share the same group key, but it is not necessary that they protect each other from masquerade. The clients do not know the group key, so cannot masquerade as legitimate servers. The GQ scheme is intended for exceptionally hostile scenarios where it is necessary to change the client key at relatively frequent intervals. The servers and clients share the group key, but the exchange is further obscured by the client key, which is on the server certificate. The client key can be changed frequently while retaining the same parameters and group key.

The MV scheme is intended for the most challenging scenarios where it is necessary to protect against both TA and server masquerade. The private values used by the TA to generate the cryptosystem are not available to the servers and the private values used by the servers to encrypt data are not available to the clients. Thus, a client cannot masquerade as a server and a server cannot masquerade as the TA. However, a client can verify a server has the correct group key even though neither the client nor server know the group key, nor can either manufacture a client key acceptable to any other client. A further feature of this scheme is that the TA can collaborate with the servers to revoke client keys.

4.1 Schnorr (IFF) Cryptosystem

The Schnorr (IFF) identity scheme can be used when certificates are generated by utilities other than the ntp-keygen program in the NTP software distribution. Certificates can be generated by the OpenSSL library or an external public certificate authority, but conveying an arbitrary public value in a certificate extension field might not be possible. The TA generates IFF parameters and keys and distributes them by secure means to all servers, then removes the group key and redistributes these data to dependent clients. Without the group key a client cannot masquerade as a legitimate server.

The IFF values hide in a DSA cuckoo structure which uses the same parameters. The values are used by an identity scheme based on DSA cryptography and described in [12] and [13] p. 285. 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.

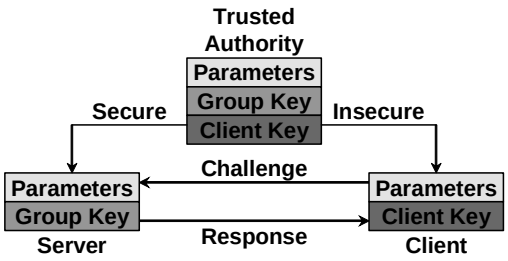

Figure 10. IFF Protocol

The TA generates a DSA parameter structure for use as IFF parameters. The IFF parameters are identical to the DSA parameters, so the OpenSSL library DSA parameter generation routine can be used directly. The DSA parameter structure is written to a file as a DSA private key encoded in PEM. Unused structure members are set to one.

I

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

Table 1. IFF Cuckoo 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 $y = k + br \bmod q$ and $x = g^k \bmod p$, then sends (y, hash(x)) to Alice.

3. Alice computes $z = g^y v^r \bmod p$ and verifies hash(z) equals hash(x).

4.2 Guillou-Quisquater (GQ) Cryptosystem

The Guillou-Quisquater (GQ) identity scheme is useful when certificates are generated by the ntp-keygen utility in the NTP distribution. The utility inserts the client key in an X.509 extension field when the certificate is generated. The client key is used when computing the response to a challenge. The TA generates the GQ parameters and keys and distributes them by secure means to all group members.

The GQ values hide in a RSA cuckoo structure which uses the same parameters. The values are used in an identity scheme based on RSA cryptography and described in [2] and [13] p. 300 (with errors). The 512-bit public modulus $n = pq$, where p and q are secret large primes. The TA rolls

random group key b ($0 < b < n$) and distributes (n, b) to all group members using secure means. The private server key and public client key are constructed later.

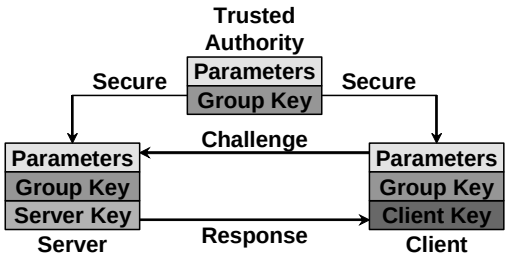

Figure 11. GQ Protocol

When generating new certificates, the server rolls new random private server key u ($0 < u < n$) and public client key its inverse obscured by the group key $v = (u^{-1})^b \bmod n$. These values replace the private and public keys normally generated by the RSA scheme. In addition, the public client key is conveyed in a X.509 certificate extension. The updated GQ structure is written as a RSA private key encoded in PEM. Unused structure members are set to one.

I

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

Table 2. GQ Cuckoo 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 $y = ku^r \bmod n$ and $x = k^b \bmod n$, then sends (y, hash(x)) to Alice.

3. Alice computes $z = v^r y^b \bmod n$ and verifies hash(z) equals hash(x).

4.3 Mu-Varadharajan (MV) Cryptosystem

The Mu-Varadharajan (MV) scheme was originally intended to encrypt broadcast transmissions to receivers which do not transmit. There is one encryption key for the broadcaster and a separate decryption key for each receiver. It operates something like a pay-per-view satellite broadcasting system where the session key is encrypted by the broadcaster and the decryption keys are held in a tamper proof set-top box. We don't use it this way, but read on.

In the MV scheme the TA constructs an intricate cryptosystem involving a number of activation keys known only to the TA. The TA decides which keys to activate and provides to the servers a private encryption key E and public decryption keys $\bar{g}$ and $\hat{g}$ 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 private decryption keys $\bar{x}_j$ and $\hat{x}_j$, which do not need to be changed if the TA activates or deactivates this key. The clients have no further information and, in particular, cannot masquerade as a server or TA.

The MV values hide in a DSA cuckoo structure which uses the same parameters, but generated in a different way. The values are used in an encryption scheme similar to El Gamal cryptography and a polynomial formed from the expansion of product terms $\prod_{0<j\le n}(x-x_j)$, as described in [10]. The paper has significant errors and serious omissions.

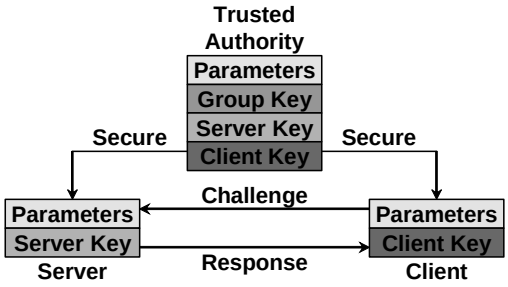

Figure 12. MV Protocol

The TA writes the server parameters, private encryption key and public decryption keys for all servers as a DSA private key encoded in PEM.

I

MV	DSA	Item	Include
p	p	modulus	all
q	q	modulus	server
E	g	private encrypt	server
$\bar{g}$	priv_key	public decrypt	server
$\hat{g}$	pub_key	public decrypt	server

Table 3. MV Server Cuckoo Structure

The TA writes the client parameters and private decryption keys for each client as a DSA private key encoded in PEM. It is used only by the designated recipient(s) who pay a suitably outrageous fee for its use. Unused structure members are set to one.

I

MV	DSA	Item	Include
p	p	modulus	all
$\bar{x}_j$	priv_key	private decrypt	client
$\hat{x}_j$	pub_key	private decrypt	client

Table 4. MV Client Cuckoo Structure

The devil is in the details. Let q be the product of n distinct primes s'_j ($j = 1...n$), where each s'_j, also called an activation key, has m significant bits. Let prime $p = 2q + 1$, so that q and each s'_j divide $p - 1$ and p has $M = nm + 1$ significant bits. Let g be a generator of the multiplicative group Z_p*; that is, gcd(g, $p - 1$) = 1 and $g^q = 1 \bmod p$. We do modular arithmetic over Z_q and then project into Z_p* as powers of g. Sometimes we have to compute an inverse b^{-1} of random b in Z_q, but for that purpose we require gcd(b, q) = 1. We expect M to be in the 500-bit range and n relatively small, like 30. The TA uses a nasty probabilistic algorithm to generate the cryptosystem.

1. Generate the m-bit primes s'_j $(0 < j \le n)$, which may have to be replaced later. As a practical matter, it is tough to find more than 30 distinct primes for $M \approx 512$ or 60 primes for $M \approx 1024$. The latter can take several hundred iterations and several minutes on a Sun Blade 1000.

2. Compute modulus $q = \prod_{0 < j \le n} s'_j$, then modulus $p = 2q + 1$. If p is composite, the TA replaces one of the primes with a new distinct prime and tries again. Note that q will hardly be a secret since p is revealed to servers and clients. However, factoring q to find the primes should be adequately hard, as this is the same problem considered hard in RSA. Question: is it as hard to find n small prime factors totalling M bits as it is to find two large prime factors totalling M bits? Remember, the bad guy doesn't know n.

3. Associate with each s'_j an element s_j such that $s_j s'_j = s'_j \bmod q$. One way to find an s_j is the quotient $s_j = \frac{q + s'_j}{s'_j}$. The student should prove the remainder is always zero.

4. Compute the generator g of Z_p using a random roll such that $\gcd(g, p - 1) = 1$ and $g^q = 1 \bmod p$. If not, roll again.

Once the cryptosystem parameters have been determined, the TA sets up a specific instance of the scheme as follows.

1. Roll n random roots x_j $(0 < x_j < q)$ for a polynomial of order n. While it may not be strictly necessary, Make sure each root has no factors in common with q.

2. Expand the n product terms $\prod_{0 < j \le n} (x - x_j)$ to form $n + 1$ coefficients $a_i \bmod q$ $(0 \le i \le n)$ in powers of x using a fast method contributed by C. Boncelet.

3. Generate $g_i = g^{a_i} \bmod p$ for all i and the generator g. Verify $\prod_{0 \le i \le n,\, 0 < j \le n} g_i^{a_i x_j^{\,i}} = 1 \bmod p$ for all i, j. Note the $a_i x_j^{\,i}$ exponent is computed mod q, but the g_i is computed mod p. Also note the expression given in the paper cited is incorrect.

4. Make master encryption key $A = \prod_{0 < i \le n,\, 0 < j < n} g_i^{x_j} \bmod p$. Keep it around for awhile, since it is expensive to compute.

5. Roll private random group key b ($0 < b < q$), where gcd(b, q) = 1 to guarantee the inverse exists, then compute $b^{-1} \bmod q$. If b is changed, all keys must be recomputed.

6. Make private client keys $\bar{x}_j = b^{-1} \sum_{0 < i \le n,\, i \ne j} x_i^{\,n} \bmod q$ and $\hat{x}_j = s_j x_j^{\,n} \bmod q$ for all j. Note that the keys for the jth client involve only s_j, but not s'_j or s. The TA sends (p, $\bar{x}_j$, $\hat{x}_j$) to the jth client(s) using secure means.

7. The activation key is initially q by construction. The TA revokes client j by dividing q by s'_j. The quotient becomes the activation key s. Note we always have to revoke one key; otherwise, the plaintext and cryptotext would be identical. The TA computes $E = A^s$, $\bar{g} = \bar{x}^s \bmod p$, $\hat{g} = \hat{x}^{sb} \bmod p$ and sends (p, E, $\bar{g}$, $\hat{g}$) to the servers using secure means.

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 $E' = E^k \bmod p$ and public decryption key $\bar{g}' = \bar{g}^k \bmod p$ and $\hat{g}' = \hat{g}^k \bmod p$. He encrypts $x = E'r$ and sends (hash(x), $\bar{g}'$, $\hat{g}'$) to Alice.

3. Alice computes the session decryption key $E'^{-1} = \bar{g}'^{\hat{x}_j} \hat{g}'^{\bar{x}_j} \bmod p$, recovers the encryption key $E' = (E'^{-1})^{-1} \bmod p$, encrypts $z = E'r \bmod p$, then verifies that hash(z) = hash(x).

5. References and Bibliography

Note: All reports and papers by D.L. Mills can be found on the web in PostScript and PDF format at www.eecis.udel.edu/~mills.

1. Adams, C., S. Farrell. Internet X.509 public key infrastructure certificate management protocols. Network Working Group Request for Comments RFC-2510, Entrust Technologies, March 1999, 30 pp.

2. Guillou, L.C., and J.-J. Quisquatar. A “paradoxical” identity-based signature scheme resulting from zero-knowledge. Proc. *CRYPTO 88 Advanced in Cryptology,* Springer-Verlag, 1990, 216-231.

3. Housley, R., et al. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. Network Working Group Request for Comments RFC-3280, RSA Laboratories, April 2002, 129 pp.

4. Mills, D.L. Public-Key cryptography for the Network Time Protocol. Internet Draft draft-ietf-stime-ntpauth-04.txt, University of Delaware, July 2002, UNFORMATTED DRAFT.

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

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

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

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

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

10. Mu, Y., and V. Varadharajan. Robust and secure broadcasting. *Proc. INDOCRYPT 2001, LNCS 2247*, Springer Verlag, 2001, 223-231.

11. Prafullchandra, H., and J. Schaad. Diffie-Hellman proof-of-possession algorithms. Network Working Group Request for Comments RFC-2875, Critical Path, Inc., July 2000, 23 pp.

12. Schnorr, C.P. Efficient signature generation for smart cards. *J. Cryptology 4*, 3 (1991), 161-174.

13. Stinson, D.R. *Cryptography - Theory and Practice*. CRC Press, Boca Raton, FA, 1995, ISBN 0-8493-8521-0.

A. Program Manual Page: Authentication Support

Authentication support allows the NTP client to verify that the server is in fact known and trusted and not an intruder intending accidentally or on purpose to masquerade as that server. The NTPv3 specification RFC-1305 defines a scheme which provides cryptographic authentication of received NTP packets. Originally, this was done using the Data Encryption Standard (DES) algorithm operating in Cipher Block Chaining (CBC) mode, commonly called DES-CBC. Subsequently, this was replaced by the RSA Message Digest 5 (MD5) algorithm using a private key, commonly called keyed-MD5. Either algorithm computes a message digest, or one-way hash, which can be used to verify the server has the correct private key and key identifier.

NTPv4 retains the NTPv3 scheme, properly described as symmetric key cryptography and, in addition, provides a new Autokey scheme based on public key cryptography. Public key cryptography is generally considered more secure than symmetric key cryptography, since the security is based on a private value which is generated by each server and never revealed. With Autokey all key distribution and management functions involve only public values, which considerably simplifies key distribution and storage. Public key management is based on X.509 certificates, which can be provided by commercial services or produced by utility programs in the OpenSSL software library or the NTPv4 distribution.

While the algorithms for symmetric key cryptography are included in the NTPv4 distribution, public key cryptography requires the OpenSSL software library to be installed before building the NTP distribution. Directions for doing that are on the Building and Installing the Distribution page.

Authentication is configured separately for each association using the key or autokey subcommand on the peer, server, broadcast and manycastclient configuration commands as described in the Configuration Options page. The authentication options described below specify the locations of the key files, if other than default, which symmetric keys are trusted and the interval between various operations, if other than default.

Authentication is always enabled, although ineffective if not configured as described below. If a NTP packet arrives including a message authentication code (MAC), it is accepted only if it passes all cryptographic checks. The checks require correct key ID, key value and message digest. If the packet has been modified in any way or replayed by an intruder, it will fail one or more of these checks and be discarded. Furthermore, the Autokey scheme requires a preliminary protocol exchange to obtain the server certificate, verify its credentials and initialize the protocol

The auth flag controls whether new associations or remote configuration commands require cryptographic authentication. This flag can be set or reset by the enable and disable commands and also by remote configuration commands sent by a ntpdc program running on another machine. If this flag is enabled, which is the default case, new broadcast/manycast client and symmetric passive associations and remote configuration commands must be cryptographically authenticated using either symmetric key or public key cryptography. If this flag is disabled, these operations are effective even if not cryptographic authenticated. It should be understood that operating with the auth flag disabled invites a significant vulnerability where a rogue hacker can masquerade as a falseticker and seriously disrupt system timekeeping. It is important to note that this flag has no purpose other than to allow or disallow a new association in response to new broadcast and sym-

metric active messages and remote configuration commands and, in particular, the flag has no effect on the authentication process itself.

An attractive alternative where multicast support is available is manycast mode, in which clients periodically troll for servers as described in the Automatic NTP Configuration Options page. Either symmetric key or public key cryptographic authentication can be used in this mode. The principle advantage of manycast mode is that potential servers need not be configured in advance, since the client finds them during regular operation, and the configuration files for all clients can be identical.

The security model and protocol schemes for both symmetric key and public key cryptography are summarized below; further details are in the briefings, papers and reports at the NTP project page linked from www.ntp.org.

A.1 Symmetric Key Cryptography

The original RFC-1305 specification allows any one of possibly 65,534 keys, each distinguished by a 32-bit key identifier, to authenticate an association. The servers and clients involved must agree on the key and key identifier to authenticate NTP packets. Keys and related information are specified in a key file, usually called ntp.keys, which must be distributed and stored using secure means beyond the scope of the NTP protocol itself. Besides the keys used for ordinary NTP associations, additional keys can be used as passwords for the ntpq and ntpdc utility programs.

When ntpd is first started, it reads the key file specified in the keys configuration command and installs the keys in the key cache. However, individual keys must be activated with the trusted command before use. This allows, for instance, the installation of possibly several batches of keys and then activating or deactivating each batch remotely using ntpdc. This also provides a revocation capability that can be used if a key becomes compromised. The requestkey command selects the key used as the password for the ntpdc utility, while the controlkey command selects the key used as the password for the ntpq utility.

A.2 Public Key Cryptography

NTPv4 supports the original NTPv3 symmetric key scheme described in RFC-1305 and in addition the Autokey protocol, which is based on public key cryptography. The Autokey Version 2 protocol described on the Autokey Protocol page verifies packet integrity using MD5 message digests and verifies the source with digital signatures and any of several digest/signature schemes. Optional identity schemes described on the Identity Schemes page and based on cryptographic challenge/response algorithms are also available. Using all of these schemes provides strong security against replay with or without modification, spoofing, masquerade and most forms of clogging attacks.

The cryptographic means necessary for all Autokey operations is provided by the OpenSSL software library. This library is available from http://www.openssl.org and can be installed using the procedures outlined in the Building and Installing the Distribution page. Once installed, the configure and build process automatically detects the library and links the library routines required.

The Autokey protocol has several modes of operation corresponding to the various NTP modes supported. Most modes use a special cookie which can be computed independently by the client

and server, but encrypted in transmission. All modes use in addition a variant of the S-KEY scheme, in which a pseudo-random key list is generated and used in reverse order. These schemes are described along with an executive summary, current status, briefing slides and reading list on the Autonomous Authentication page.

The specific cryptographic environment used by Autokey servers and clients is determined by a set of files and soft links generated by the ntp-keygen program. This includes a required host key file, required certificate file and optional sign key file, leapsecond file and identity scheme files. The digest/signature scheme is specified in the X.509 certificate along with the matching sign key. There are several schemes available in the OpenSSL software library, each identified by a specific string such as md5WithRSAEncryption, which stands for the MD5 message digest with RSA encryption scheme. The current NTP distribution supports all the schemes in the OpenSSL library, including those based on RSA and DSA digital signatures.

NTP secure groups can be used to define cryptographic compartments and security hierarchies. It is important that every host in the group be able to construct a certificate trail to one or more trusted hosts in the same group. Each group host runs the Autokey protocol to obtain the certificates for all hosts along the trail to one or more trusted hosts. This requires the configuration file in all hosts to be engineered so that, even under anticipated failure conditions, the NTP subnet will form such that every group host can find a trail to at least one trusted host.

A.3 Operation

A specific combination of authentication scheme (none, symmetric key, public key) and identity scheme is called a cryptotype, although not all combinations are compatible. There may be management configurations where the clients, servers and peers may not all support the same cryptotypes. A secure NTPv4 subnet can be configured in many ways while keeping in mind the principles explained above and in this section. Note however that some cryptotype combinations may successfully interoperate with each other, but may not represent good security practice.

The cryptotype of an association is determined at the time of mobilization, either at configuration time or some time later when a message of appropriate cryptotype arrives. When mobilized by a server or peer configuration command and no key or autokey subcommands are present, the association is not authenticated; if the key subcommand is present, the association is authenticated using the symmetric key ID specified; if the autokey subcommand is present, the association is authenticated using Autokey.

When multiple identity schemes are supported in the Autokey protocol, the first message exchange determines which one is used. The client request message contains bits corresponding to which schemes it has available. The server response message contains bits corresponding to which schemes it has available. Both server and client match the received bits with their own and select a common scheme.

Following the principle that time is a public value, a server responds to any client packet that matches its cryptotype capabilities. Thus, a server receiving an unauthenticated packet will respond with an unauthenticated packet, while the same server receiving a packet of a cryptotype it supports will respond with packets of that cryptotype. However, unconfigured broadcast or manycast client associations or symmetric passive associations will not be mobilized unless the

server supports a cryptotype compatible with the first packet received. By default, unauthenticated associations will not be mobilized unless overridden in a decidedly dangerous way.

Some examples may help to reduce confusion. Client Alice has no specific cryptotype selected. Server Bob has both a symmetric key file and minimal Autokey files. Alice's unauthenticated messages arrive at Bob, who replies with unauthenticated messages. Cathy has a copy of Bob's symmetric key file and has selected key ID 4 in messages to Bob. Bob verifies the message with his key ID 4. If it's the same key and the message is verified, Bob sends Cathy a reply authenticated with that key. If verification fails, Bob sends Cathy a thing called a crypto-NAK, which tells her something broke. She can see the evidence using the ntpq program.

Denise has rolled her own host key and certificate. She also uses one of the identity schemes as Bob. She sends the first Autokey message to Bob and they both dance the protocol authentication and identity steps. If all comes out okay, Denise and Bob continue as described above.

It should be clear from the above that Bob can support all the girls at the same time, as long as he has compatible authentication and identity credentials. Now, Bob can act just like the girls in his own choice of servers; he can run multiple configured associations with multiple different servers (or the same server, although that might not be useful). But, wise security policy might preclude some cryptotype combinations; for instance, running an identity scheme with one server and no authentication with another might not be wise.

A.4 Key Management

The cryptographic values used by the Autokey protocol are incorporated as a set of files generated by the ntp-keygen utility program, including symmetric key, host key and public certificate files, as well as sign key, identity parameters and leapseconds files. Alternatively, host and sign keys and certificate files can be generated by the OpenSSL utilities and certificates can be imported from public certificate authorities. Note that symmetric keys are necessary for the ntpq and ntpdc utility programs. The remaining files are necessary only for the Autokey protocol.

Certificates imported from OpenSSL or public certificate authorities have certian limitations. The certificate should be in ASN.1 syntax, X.509 Version 3 format and encoded in PEM, which is the same format used by OpenSSL. The overall length of the certificate encoded in ASN.1 must not exceed 1024 bytes. The subject distinguished name field (CN) is the fully qualified name of the host on which it is used; the remaining subject fields are ignored. The certificate extension fields must not contain either a subject key identifier or a issuer key identifier field; however, an extended key usage field for a trusted host must contain the value trustRoot;. Other extension fields are ignored.

A.5 Authentication Commands

autokey [logsec]

Specifies the interval between regenerations of the session key list used with the Autokey protocol. Note that the size of the key list for each association depends on this interval and the current poll interval. The default value is 12 (4096 s or about 1.1 hours). For poll intervals above the specified interval, a session key list with a single entry will be regenerated for every message sent.

controlkey key

Specifies the key identifier to use with the ntpq utility, which uses the standard protocol defined in RFC-1305. The key argument is the key identifier for a trusted key, where the value can be in the range 1 to 65,534, inclusive.

crypto [cert file] [leap file] [randfile file] [host file] [sign file] [gq file] [gqpar file] [iffpar file] [mvpar file] [pw password]

This command requires the OpenSSL library. It activates public key cryptography, selects the message digest and signature encryption scheme and loads the required private and public values described above. If one or more files are left unspecified, the default names are used as described above. Unless the complete path and name of the file are specified, the location of a file is relative to the keys directory specified in the keysdir command or default /usr/local/etc. Following are the subcommands: <dl>

cert file

Specifies the location of the required host public certificate file. This overrides the link ntpkey_cert_hostname in the keys directory.

gqpar file

Specifies the location of the optional GQ parameters file. This overrides the link ntpkey_gq_hostname in the keys directory.

host file

Specifies the location of the required host key file. This overrides the link ntpkey_key_hostname in the keys directory.

iffpar file

Specifies the location of the optional IFF parameters file.This overrides the link ntpkey_iff_hostname in the keys directory.

leap file

Specifies the location of the optional leapsecond file. This overrides the link ntpkey_leap in the keys directory.

mvpar file

Specifies the location of the optional MV parameters file. This overrides the link ntpkey_mv_hostname in the keys directory.

pw password

Specifies the password to decrypt files containing private keys and identity parameters. This is required only if these files have been encrypted.

randfile file

Specifies the location of the random seed file used by the OpenSSL library. The defaults are described in the main text above.

sign file

Specifies the location of the optional sign key file. This overrides the link ntpkey_sign_hostname in the keys directory. If this file is not found, the host key is also the sign key.

keys keyfile

Specifies the complete path and location of the MD5 key file containing the keys and key identifiers used by ntpd, ntpq and ntpdc when operating with symmetric key cryptography. This is the same operation as the -k command line option.

keysdir path

This command specifies the default directory path for cryptographic keys, parameters and certificates. The default is /usr/local/etc/.

requestkey key

Specifies the key identifier to use with the ntpdc utility program, which uses a proprietary protocol specific to this implementation of ntpd. The key argument is a key identifier for the trusted key, where the value can be in the range 1 to 65,534, inclusive.

revoke [logsec]

Specifies the interval between re-randomization of certain cryptographic values used by the Autokey scheme, as a power of 2 in seconds. These values need to be updated frequently in order to deflect brute-force attacks on the algorithms of the scheme; however, updating some values is a relatively expensive operation. The default interval is 16 (65,536 s or about 18 hours). For poll intervals above the specified interval, the values will be updated for every message sent.

trustedkey key [...]

Specifies the key identifiers which are trusted for the purposes of authenticating peers with symmetric key cryptography, as well as keys used by the ntpq and ntpdc programs. The authentication procedures require that both the local and remote servers share the same key and key identifier for this purpose, although different keys can be used with different servers. The key arguments are 32-bit unsigned integers with values from 1 to 65,534.

A.6 Error Codes

The following error codes are reported via the NTP control and monitoring protocol trap mechanism.

101 (bad field format or length)
The packet has invalid version, length or format.

102 (bad timestamp)
The packet timestamp is the same or older than the most recent received. This could be due to a replay or a server clock time step.

103 (bad filestamp)
The packet filestamp is the same or older than the most recent received. This could be due to a replay or a key file generation error.

104 (bad or missing public key)
The public key is missing, has incorrect format or is an unsupported type.

105 (unsupported digest type)
The server requires an unsupported digest/signature scheme.

106 (mismatched digest types)
Not used.

107 (bad signature length)
The signature length does not match the current public key.

108 (signature not verified)
The message fails the signature check. It could be bogus or signed by a different private key.

109 (certificate not verified)
The certificate is invalid or signed with the wrong key.

110 (certificate not verified)
The certificate is not yet valid or has expired or the signature could not be verified.

111 (bad or missing cookie)
The cookie is missing, corrupted or bogus.

112 (bad or missing leapseconds table)
The leapseconds table is missing, corrupted or bogus.

113 (bad or missing certificate)
The certificate is missing, corrupted or bogus.

114 (bad or missing identity)
The identity key is missing, corrupt or bogus.

A.7 Files

See the ntp-keygen page.

A.8 Leapseconds Table

The NIST provides a file documenting the epoch for all historic occasions of leap second insertion since 1972. The leapsecond table shows each epoch of insertion along with the offset of International Atomic Time (TAI) with respect to Coordinated Universal Time (UTC), as disseminated by NTP. The table can be obtained directly from NIST national time servers using ftp as the ASCII file pub/leap-seconds.

While not strictly a security function, the Autokey protocol provides means to securely retrieve the leapsecond table from a server or peer. Servers load the leapsecond table directly from the file specified in the crypto command, with default ntpkey_leap, while clients can obtain the table indi-

rectly from the servers using the Autokey protocol. Once loaded, the table can be provided on request to other clients and servers.

B. Program Manual Page: ntp-keygen Program

B.1 Synopsis

ntp-keygen [-deGgHIMnPT] [-c [RSA-MD2 | RSA-MD5 | RSA-SHA | RSA-SHA1 | RSA-MDC2 | RSA-RIPEMD160 | DSA-SHA | DSA-SHA1]] [-i name] [-p password] [-S [RSA | DSA]] [-s name] [-v nkeys]

B.2 Description

This program generates cryptographic data files used by the NTPv4 authentication and identification schemes. It generates MD5 key files used in symmetric key cryptography. In addition, if the OpenSSL software library has been installed, it generates keys, certificate and identity files used in public key cryptography. These files are used for cookie encryption, digital signature and challenge/response identification algorithms compatible with the Internet standard security infrastructure.

All files are in PEM-encoded printable ASCII format, so they can be embedded as MIME attachments in mail to other sites and certificate authorities. By default, files are not encrypted. The -p password option specifies the write password and -q password option the read password for previously encrypted files. The ntp-keygen program prompts for the password if it reads an encrypted file and the password is missing or incorrect. If an encrypted file is read successfully and no write password is specified, the read password is used as the write password by default.

The ntpd configuration command crypto pw password specifies the read password for previously encrypted files. The daemon expires on the spot if the password is missing or incorrect. For convenience, if a file has been previously encrypted, the default read password is the name of the host running the program. If the previous write password is specified as the host name, these files can be read by that host with no explicit password.

File names begin with the prefix ntpkey_ and end with the postfix _hostname.filestamp , where hostname is the owner name, usually the string returned by the Unix gethostname() routine, and filestamp is the NTP seconds when the file was generated, in decimal digits. This both guarantees uniqueness and simplifies maintenance procedures, since all files can be quickly removed by a rm ntpkey* command or all files generated at a specific time can be removed by a rm * filestamp command. To further reduce the risk of misconfiguration, the first two lines of a file contain the file name and generation date and time as comments.

All files are installed by default in the keys directory /usr/local/etc, which is normally in a shared filesystem in NFS-mounted networks. The actual location of the keys directory and each file can be overridden by configuration commands, but this is not recommended. Normally, the files for each host are generated by that host and used only by that host, although exceptions exist as noted later on this page.

Normally, files containing private values, including the host key, sign key and identification parameters, are permitted root read/write-only; while others containing public values are permitted world readable. Alternatively, files containing private values can be encrypted and these files permitted world readable, which simplifies maintenance in shared file systems. Since uniqueness

is insured by the hostname and file name extensions, the files for a NFS server and dependent clients can all be installed in the same shared directory.

The recommended practice is to keep the file name extensions when installing a file and to install a soft link from the generic names specified elsewhere on this page to the generated files. This allows new file generations to be activated simply by changing the link. If a link is present, ntpd follows it to the file name to extract the filestamp. If a link is not present, ntpd extracts the filestamp from the file itself. This allows clients to verify that the file and generation times are always current. The ntp-keygen program uses the same timestamp extension for all files generated at one time, so each generation is distinct and can be readily recognized in monitoring data.

B.3 Running the program

The safest way to run the ntp-keygen program is logged in directly as root. The recommended procedure is change to the keys directory, usually /ust/local/etc, then run the program. When run for the first time, or if all ntpkey files have been removed, the program generates a RSA host key file and matching RSA-MD5 certificate file, which is all that is necessary in many cases. The program also generates soft links from the generic names to the respective files. If run again, the program uses the same host key file, but generates a new certificate file and link.

The host key is used to encrypt the cookie when required and so must be RSA type. By default, the host key is also the sign key used to encrypt signatures. When necessary, a different sign key can be specified and this can be either RSA or DSA type. By default, the message digest type is MD5, but any combination of sign key type and message digest type supported by the OpenSSL library can be specified, including those using the MD2, MD5, SHA, SHA1, MDC2 and RIPE160 message digest algorithms. However, the scheme specified in the certificate must be compatible with the sign key. Certificates using any digest algorithm are compatible with RSA sign keys; however, only SHA and SHA1 certificates are compatible with DSA sign keys.

Private/public key files and certificates are compatible with other OpenSSL applications and very likely other libraries as well. Certificates or certificate requests derived from them should be compatible with extant industry practice, although some users might find the interpretation of X509v3 extension fields somewhat liberal. However, the identification parameter files, although encoded as the other files, are probably not compatible with anything other than Autokey.

Running the program as other than root and using the Unix su command to assume root may not work properly, since by default the OpenSSL library looks for the random seed file .rnd in the user home directory. However, there should be only one .rnd, most conveniently in the root directory, so it is convenient to define the $RANDFILE environment variable used by the OpenSSL library as the path to /.rnd.

Installing the keys as root might not work in NFS-mounted shared file systems, as NFS clients may not be able to write to the shared keys directory, even as root. In this case, NFS clients can specify the files in another directory such as /etc using the keysdir command. There is no need for one client to read the keys and certificates of other clients or servers, as these data are obtained automatically by the Autokey protocol.

Ordinarily, cryptographic files are generated by the host that uses them, but it is possible for a trusted agent (TA) to generate these files for other hosts; however, in such cases files should

always be encrypted. The subject name and trusted name default to the hostname of the host generating the files, but can be changed by command line options. It is convenient to designate the owner name and trusted name as the subject and issuer fields, respectively, of the certificate. The owner name is also used for the host and sign key files, while the trusted name is used for the identity files.

B.4 Trusted Hosts and Groups

Each cryptographic configuration involves selection of a signature scheme and identification scheme, called a cryptotype, as explained in the Authentication Options page. The default cryptotype uses RSA encryption, MD5 message digest and TC identification. First, configure a NTP subnet including one or more low-stratum trusted hosts from which all other hosts derive synchronization directly or indirectly. Trusted hosts have trusted certificates; all other hosts have nontrusted certificates. These hosts will automatically and dynamically build authoritative certificate trails to one or more trusted hosts. A trusted group is the set of all hosts that have, directly or indirectly, a certificate trail ending at a trusted host. The trail is defined by static configuration file entries or dynamic means described on the Automatic NTP Configuration Options page.

On each trusted host as root, change to the keys directory. To insure a fresh fileset, remove all ntpkey files. Then run ntp-keygen -T to generate keys and a trusted certificate. On all other hosts do the same, but leave off the -T flag to generate keys and nontrusted certificates. When complete, start the NTP daemons beginning at the lowest stratum and working up the tree. It may take some time for Autokey to instantiate the certificate trails throughout the subnet, but setting up the environment is completely automatic.

If it is necessary to use a different sign key or different digest/signature scheme than the default, run ntp-keygen with the -S type option, where type is either RSA or DSA. The most often need to do this is when a DSA-signed certificate is used. If it is necessary to use a different certificate scheme than the default, run ntp-keygen with the -c scheme option and selected scheme as needed. If ntp-keygen is run again without these options, it generates a new certificate using the same scheme and sign key.

After setting up the environment it is advisable to update certificates from time to time, if only to extend the validity interval. Simply run ntp-keygen with the same flags as before to generate new certificates using existing keys. However, if the host or sign key is changed, ntpd should be restarted. When ntpd is restarted, it loads any new files and restarts the protocol. Other dependent hosts will continue as usual until signatures are refreshed, at which time the protocol is restarted.

B.5 Identity Schemes

As mentioned on the Autonomous Authentication page, the default TC identity scheme is vulnerable to a middleman attack. However, there are more secure identity schemes available, including PC, IFF, GQ and MV described on the Identification Schemes page. These schemes are based on a TA, one or more trusted hosts and some number of nontrusted hosts. Trusted hosts prove identity using values provided by the TA, while the remaining hosts prove identity using values provided by a trusted host and certificate trails that end on that host. The name of a trusted host is also the name of its sugroup and also the subject and issuer name on its trusted certificate. The TA is not necessarily a trusted host in this sense, but often is.

In some schemes there are separate keys for servers and clients. A server can also be a client of another server, but a client can never be a server for another client. In general, trusted hosts and nontrusted hosts that operate as both server and client have parameter files that contain both server and client keys. Hosts that operate only as clients have key files that contain only client keys.

The PC scheme supports only one trusted host in the group. On trusted host alice run ntp-keygen -P -p password to generate the host key file ntpkey_RSAkey_ alice.filestamp and trusted private certificate file ntpkey_RSA-MD5_cert_ alice.filestamp . Copy both files to all group hosts; they replace the files which would be generated in other schemes. On each host bob install a soft link from the generic name ntpkey_host_ bob to the host key file and soft link ntpkey_cert_ bob to the private certificate file. Note the generic links are on bob , but point to files generated by trusted host alice . In this scheme it is not possible to refresh either the keys or certificates without copying them to all other hosts in the group.

For the IFF scheme proceed as in the TC scheme to generate keys and certificates for all group hosts, then for every trusted host in the group, generate the IFF parameter file. On trusted host alice run ntp-keygen -T -I -p password to produce her parameter file ntpkey_IFFpar_ alice.filestamp , which includes both server and client keys. Copy this file to all group hosts that operate as both servers and clients and install a soft link from the generic ntpkey_iff_ alice to this file. If there are no hosts restricted to operate only as clients, there is nothing further to do. As the IFF scheme is independent of keys and certificates, these files can be refreshed as needed.

If a rogue client has the parameter file, it could masquerade as a legitimate server and present a middleman threat. To eliminate this threat, the client keys can be extracted from the parameter file and distributed to all restricted clients. After generating the parameter file, on alice run ntp-keygen -e and pipe the output to a file or mail program. Copy or mail this file to all restricted clients. On these clients install a soft link from the generic ntpkey_iff_ alice to this file. To further protect the integrity of the keys, each file can be encrypted with a secret password.

For the GQ scheme proceed as in the TC scheme to generate keys and certificates for all group hosts, then for every trusted host in the group, generate the IFF parameter file. On trusted host alice run ntp-keygen -T -G -p password to produce her parameter file ntpkey_GQpar_ alice.filestamp , which includes both server and client keys. Copy this file to all group hosts and install a soft link from the generic ntpkey_gq_ alice to this file. In addition, on each host bob install a soft link from generic ntpkey_gq_ bob to this file. As the GQ scheme updates the GQ parameters file and certificate at the same time, keys and certificates can be regenerated as needed.

For the MV scheme, proceed as in the TC scheme to generate keys and certificates for all group hosts. For illustration assume trish is the TA, alice one of several trusted hosts and bob one of her clients. On TA trish run ntp-keygen -V n -p password , where n is the number of revokable keys (typically 5) to produce the parameter file ntpkeys_MVpar_ trish.filestamp and client key files ntpkeys_MVkey d _ trish.filestamp where d is the key number (0 < d < n). Copy the parameter file to alice and install a soft link from the generic ntpkey_mv_ alice to this file. Copy one of the client key files to alice for later distribution to her clients. It doesn’t matter which client key file goes to alice , since they all work the same way. Alice copies the client key file to all of her cliens. On client bob install a soft link from generic ntpkey_mvkey_ bob to the client key file. As the MV scheme is independent of keys and certificates, these files can be refreshed as needed.

B.6 Command Line Options

-c [RSA-MD2 | RSA-MD5 | RSA-SHA | RSA-SHA1 | RSA-MDC2 | RSA-RIPEMD160 | DSA-SHA | DSA-SHA1]

Select certificate message digest/signature encryption scheme. Note that RSA schemes must be used with a RSA sign key and DSA schemes must be used with a DSA sign key. The default without this option is RSA-MD5.

-d

Enable debugging. This option displays the cryptographic data produced in eye-friendly billboards.

-e

Write the IFF client keys to the standard output. This is intended for automatic key distribution by mail.-G

Generate parameters and keys for the GQ identification scheme, obsoleting any that may exist.

-g

Generate keys for the GQ identification scheme using the existing GQ parameters. If the GQ parameters do not yet exist, create them first.

-H

Generate new host keys, obsoleting any that may exist.

-I

Generate parameters for the IFF identification scheme, obsoleting any that may exist.

-i name

Set the suject name to name . This is used as the subject field in certificates and in the file name for host and sign keys.

-M

Generate MD5 keys, obsoleting any that may exist.

-P

Generate a private certificate. By default, the program generates public certificates.

-p password

Encrypt generated files containing private data with password and the DES-CBC algorithm.

-q

Set the password for reading files to password .

-S [RSA | DSA]

Generate a new sign key of the designated type, obsoleting any that may exist. By default, the program uses the host key as the sign key.

-s name

Set the issuer name to name . This is used for the issuer field in certificates and in the file name for identity files.-T

Generate a trusted certificate. By default, the program generates a non-trusted certificate.

-V nkeys

Generate parameters and keys for the Mu-Varadharajan (MV) identification scheme.

B.7 Random Seed File

All cryptographically sound key generation schemes must have means to randomize the entropy seed used to initialize the internal pseudo-random number generator used by the library routines. The OpenSSL library uses a designated random seed file for this purpose. The file must be available when starting the NTP daemon and ntp-keygen program. If a site supports OpenSSL or its companion OpenSSH, it is very likely that means to do this are already available.

It is important to understand that entropy must be evolved for each generation, for otherwise the random number sequence would be predictable. Various means dependent on external events, such as keystroke intervals, can be used to do this and some systems have built-in entropy sources. Suitable means are described in the OpenSSL software documentation, but are outside the scope of this page.

The entropy seed used by the OpenSSL library is contained in a file, usually called .rnd, which must be available when starting the NTP daemon or the ntp-keygen program. The NTP daemon will first look for the file using the path specified by the randfile subcommand of the crypto configuration command. If not specified in this way, or when starting the ntp-keygen program, the OpenSSL library will look for the file using the path specified by the RANDFILE environment variable in the user home directory, whether root or some other user. If the RANDFILE environment variable is not present, the library will look for the .rnd file in the user home directory. If the file is not available or cannot be written, the daemon exits with a message to the system log and the program exits with a suitable error message.

B.8 Cryptographic Data Files

All other file formats begin with two lines. The first contains the file name, including the generated host name and filestamp. The second contains the datestamp in conventional Unix date format. Lines beginning with # are considered comments and ignored by the ntp-keygen program and ntpd daemon. Cryptographic values are encoded first using ASN.1 rules, then encrypted if necessary, and finally written PEM-encoded printable ASCII format preceded and followed by MIME content identifier lines.

The format of the symmetric keys file is somewhat different than the other files in the interest of backward compatibility. Since DES-CBC is deprecated in NTPv4, the only key format of interest is MD5 alphanumeric strings. Following the header the keys are entered one per line in the format

keyno type key

where keyno is a positive integer in the range 1-65,535, type is the string MD5 defining the key format and key is the key itself, which is a printable ASCII string 16 characters or less in length. Each character is chosen from the 93 printable characters in the range 0x21 through 0x7f excluding space and the '#' character.

Note that the keys used by the ntpq and ntpdc programs are checked against passwords requested by the programs and entered by hand, so it is generally appropriate to specify these keys in human readable ASCII format.

The ntp-keygen program generates a MD5 symmetric keys file ntpkey_MD5key_ hostname.filestamp . Since the file contains private shared keys, it should be visible only to root and distributed by secure means to other subnet hosts. The NTP daemon loads the file ntp.keys, so ntp-keygen installs a soft link from this name to the generated file. Subsequently, similar soft links must be installed by manual or automated means on the other subnet hosts. While this file is not used with the Autokey Version 2 protocol, it is needed to authenticate some remote configuration commands used by the ntpq and ntpdc utilities.

B.9 Bugs

It can take quite a while to generate some cryptographic values, from one to several minutes with modern architectures such as UltraSPARC and up to tens of minutes to an hour with older architectures such as SPARC IPC.

C. Autokey Protocol Specification

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

The Network Time Protocol (NTP) contains provisions to cryptographically authenticate individual servers as described in the most recent protocol NTP Version 3 (NTPv3) specification [11]; however, that specification does not provide a scheme for the distribution of cryptographic keys, nor does it provide for the retrieval of cryptographic media that reliably bind the server identification credentials with the associated private keys and related public values. However, conventional key agreement and digital signatures with large client populations can cause significant performance degradations, especially in time critical applications such as NTP. In addition, there are problems unique to NTP in the interaction between the authentication and synchronization functions, since each requires the other.

This document describes a cryptographically sound and efficient methodology for use in NTP and similar distributed protocols. As demonstrated in the reports and briefings cited in the references at the end of this document, there is a place for PKI and related schemes, but none of these schemes alone satisfies the requirements of the NTP security model. The various key agreement schemes [8], [13], [5] proposed by the IETF require per-association state variables, which contradicts the principles of the remote procedure call (RPC) paradigm in which servers keep no state for a possibly large client population. An evaluation of the PKI model and algorithms as implemented in the 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.

A revised security model and authentication scheme called Autokey was proposed in earlier reports [10], [9]. It is based on a combination of PKI and a pseudo-random sequence generated by repeated hashes of a cryptographic value involving both public and private components. This scheme has been tested and evaluated in a local environment and in the CAIRN experiment network funded by DARPA. A detailed description of the security model, design principles and implementation experience is presented in this document.

Additional information about NTP, including executive summaries, briefings and bibliography can be found on the NTP project page linked from www.ntp.org. The NTPv4 reference implementation for Unix and Windows, including sources and documentation in HTML, is available from the NTP repository at the same site. All of the features described in this document, including support for both IPv4 and IPv6 address families, are included in the current development version at that repository. The reference implementation is not intended to become part of any standard that

may be evolved from this document, but to serve as an example of how the procedures described in this document can be implemented in a practical way.

C.1 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 which are valid only over designated time intervals; but, time intervals can be enforced only when participating servers and clients are reliably synchronized to UTC. Second, the NTP subnet is hierarchical by nature, so time and trust flow from the primary servers at the root through secondary servers to the clients at the leaves.

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

t is important to note that the notion of proventic does not necessarily imply the time is correct. A 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 document. 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 client associations and according to the NTP mitigation algorithms. While a certificate authority (CA) must satisfy this requirement when signing a certificate request, the certificate itself can be stored in public directories and retrieved over unsecured network paths.

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) [7] and Authentication Header (AH) [6] for IPv4 and IPv6. Cryptographic algorithms that use these headers for various purposes include those developed for the PKI, including MD5 message digests, RSA digital signatures and several variations of Diffie-Hellman key agreements. The fundamental assumption in the security model is that packets transmitted over the Internet can be intercepted by other than the intended receiver, remanufactured in various ways and replayed in whole or part. These packets can cause the client to believe or produce incorrect information, cause protocol operations to fail, interrupt network service or consume precious 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 document is to detect and discard spurious packets sent by other than the intended sender or sent by the intended sender, but modified or replayed by an intruder. The cryptographic means of the

reference implementation are based on the OpenSSL cryptographic software library available at www.openssl.org, but other libraries with equivalent functionality could be used as well. It is important for distribution and export purposes that the way in which these algorithms are used precludes encryption of any data other than incidental to the construction of digital signatures.

There are a number of defense mechanisms already built in the NTP architecture, protocol and algorithms. The fundamental timestamp exchange scheme is inherently resistant to spoof 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 mischiefs.

The NTP security model assumes the following possible threats. Further discussion is in [9] and in the briefings at the NTP project page, but beyond the scope of this document.

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 unlikely cases considered in Appendix G, the modified packet cannot arrive at the victim before the original packet.

4. In a middleman 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 Appendix G, the middleman does not have the server private keys or identity parameters.

The NTP security model assumes the following possible limitations. Further discussion is in [9] and in the briefings at the NTP project page, but beyond the scope of this document.

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. All proventication functions must involve only public values transmitted over the net with the single exception of encrypted signatures and cookies intended only to authenticate the source. Private values must never be disclosed beyond the machine on which they were created.

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

6. Error checking must be at the enhanced paranoid level, as network terrorists may be able to craft errored packets that consume excessive cycles with needless result. While this document includes an informal vulnerability analysis and error protection paradigm, a formal model based on communicating finite-state machine analysis remains to be developed.

Unlike the Secure Shell 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 proventicate the client and others not.

However, the NTP security model specifically assumes that access control is performed by means external to the protocol and that all time values and cryptographic values are public, so there is no need to associate each interface with different cryptographic values. To do so would create the possibility of a two-faced clock, which is ordinarily considered a Byzantine hazard. In other words, there is one set of private secrets for the host, not one for each interface. In the NTP design the host name, as returned by the Unix gethostname() library function, represents all interface addresses. Since at least in some host configurations the host name may not be identifiable in a DNS query, the name must be either configured in advance or obtained directly from the server using the Autokey protocol.

C.2 Approach

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

1. It must interoperate with the existing NTP architecture model and protocol design. In particular, it must support the symmetric key scheme described in [11]. As a practical matter, the reference implementation must use the same internal key management system, including the use of 32-bit key IDs and existing mechanisms to store, activate and revoke keys.

2. It must provide for the independent collection of cryptographic values and time values. A NTP packet is accepted for processing only when the required cryptographic values have been obtained and verified and the NTP header has passed all sanity checks.

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

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

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

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

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

8. The reference implementation must contain provisions to generate cryptographic key files specific to each client and server.

C.3 Autokey Cryptography

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

NTPv3 and NTPv4 symmetric key cryptography use keyed-MD5 message digests with a 128-bit private key and 32-bit key ID. In order to retain backward 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 below. The server looks up the key associated

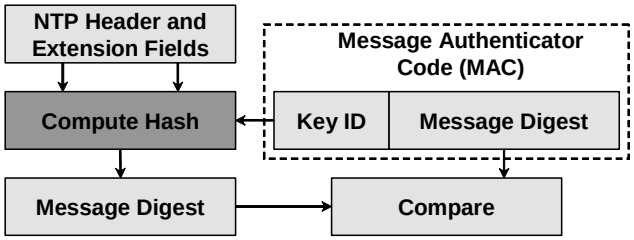

Figure 13. Receiving Messages

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.

There are three Autokey protocol variants corresponding to each of the three NTP modes: server, symmetric and broadcast. All three variants make use of specially contrived session keys, called autokeys, and a precomputed pseudo-random sequence of autokeys with the key IDs saved in a key list. As in the original NTPv3 authentication scheme, the Autokey protocol operates separately for each association, so there may be several autokey sequences operating independently at the same time.

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

Figure 14. NTPv4 Autokey

The four values are hashed by the MD5 message digest 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 Source IP and Dest IP fields contain 32 bits; for use with IPv6, these fields contain 128 bits. In either case the Key ID and Cookie fields contain 32 bits. Thus, an IPv4 autokey has four 32-bit words, while an IPv6 autokey has ten 32-bit words. The source and destination IP addresses and key ID are public values visible in the packet, while the cookie can be a public value or shared private value, depending on the mode.

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

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 visible in the transmitted packet must be the same as those used to construct the autokey sequence and key list and that these fields be the same as those visible in the received packet.

Provisions are included in the reference implementation to handle cases when these addresses change, as possible in mobile IP. For scenarios where the endpoint IP addresses are not available, an optional public identification value could be used instead of the addresses. Examples include the Interplanetary Internet, where bundles are identified by name rather than address. Specific provisions are for further study.

The figure below shows how the autokey list and autokey values are computed. The key list con-

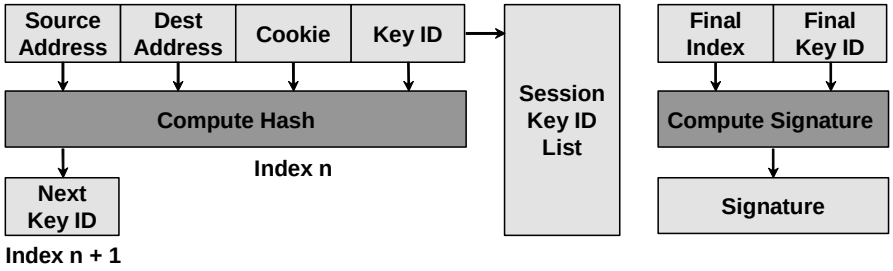

Figure 15. Constructing Key List

sists of a sequence of key IDs starting with a random 32-bit nonce (autokey seed) equal to or greater than the pivot as the first key ID. The first autokey is computed as above using the given cookie and the first 32 bits of the result in network byte order become the next key ID. 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.

The index of the last key ID in the list is saved along with the next key ID for that entry, collectively called the autokey values. The autokey values are then signed. The list is used in reverse order as in the figure below, so that the first autokey used is the last one generated. The Autokey

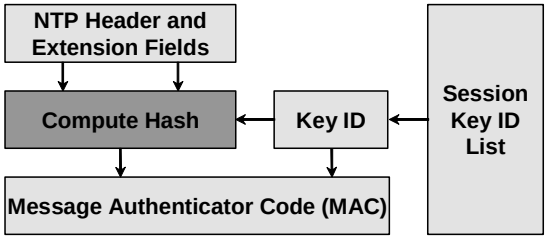

Figure 16. Transmitting Messages

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

C.4 Autokey Operations

The Autokey protocol has three variations, called dances, corresponding to the NTP server, symmetric and broadcast modes. 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, a key ID of 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 previous Autokey versions the cookie was transmitted in clear on the assumption it was not useful to a wiretapper other than to launch an ineffective replay attack. However, a middleman could intercept the cookie and manufacture bogus messages acceptable to the client. In order to reduce the risk of such an attack, the Autokey Version 2 server encrypts the cookie using a public key supplied by the client. While requiring additional processor resources for the encryption, this makes it effectively impossible to spoof a cookie or masquerade as the server.

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

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

In broadcast dance clients normally do not send packets to the server, except when first starting up to verify credentials and calibrate the propagation delay. At the same time the client runs the broadcast dance to obtain the autokey values. 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, the client verifies further server packets using the autokey sequence.

The symmetric dance is similar to the server dance and keeps only a small amount of state between the arrival of a packet and departure of the reply. 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 random value. There exists a possible race condition where each peer sends a cookie request message before receiving the cookie response from the other peer. In this case, each peer winds up with two values, one it generated and one the other peer generated. The ambiguity is resolved simply by computing the working cookie as the EXOR of the two values.

Autokey choreography includes one or more exchanges, each with a specific purpose, that must be completed in order. The client obtains the server host name, digest/signature scheme and identity scheme in the parameter exchange. It recursively obtains and verifies certificates on the trail leading to a trusted certificate in the certificate exchange and verifies the server identity in the

identity exchange. In the values exchange the client obtains the cookie and autokey values, depending on the particular dance. Finally, the client presents its self-signed certificate to the server for signature in the sign exchange.

The ultimate security of Autokey is based on digitally signed certificates and a certificate infrastructure compatible with [1] and [4]. The Autokey protocol builds the certificate trail from the primary servers, which presumably have trusted self-signed certificates, recursively by stratum. Each stratum $n + 2$ server obtains the certificate of a stratum n server, presumably signed by a stratum $n - 1$ server, and then the stratum $n + 1$ server presents its own self-signed certificate for signature by the stratum n server. As the NTP subnet forms from the primary servers at the root outward to the leaves, each server accumulates non-duplicative certificates for all associations and for all trails. In typical NTP subnets, this results in a good deal of useful redundancy and cross checking and making it even harder for a terrorist to subvert.

In order to prevent masquerade, it is necessary for the stratum n server to prove identity to the stratum $n + 1$ server when signing its certificate. In many applications a number of servers share a single security compartment, so it is only necessary that each server verifies identity to the group. Although no specific identity scheme is specified in this document, Appendix H describes a number of them based on cryptographic challenge-response algorithms. The reference implementation includes all of them with provision to add more if required.

Once the certificates and identity have been validated, subsequent packets are validated by digital signatures or autokey sequences. These packets are presumed to contain valid time values; however, unless the system clock has already been set by some other proventic means, it is not known whether these values actually represent a truechime or falsetick source. As the protocol evolves, the NTP associations continue to accumulate time values until a majority clique is available to synchronize the system clock. At this point the NTP intersection algorithm culls the falsetickers from the population and the remaining truechimers are allowed to discipline the clock.

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

Once the Autokey values have been instantiated, the dances are 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 insures that stale values never propagate beyond a clock step. At intervals of about one day the reference implementation purges all associations, refreshes all signatures, garbage collects expired certificates and refreshes the server seed.

C.5 Public Key Signatures and Timestamps

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

In order to foil such attacks, every signed extension field carries a timestamp in the form of the NTP seconds at the signature epoch. The signature spans the entire extension field including the timestamp. If the Autokey protocol has verified a proventic source and the NTP algorithms have validated the time values, the system clock can be synchronized and signatures will then carry a nonzero (valid) timestamp. Otherwise the system clock is unsynchronized and signatures carry a zero (invalid) timestamp. The protocol detects and discards replayed extension fields with old or duplicate timestamps, as well as fabricated extension fields with bogus timestamps, before any values are used or signatures verified.

There are three signature types currently defined:

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

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

3. Public values signature/timestamp: All public key, certificate and leap second table values are signed at the time of generation, which occurs when the system clock is first synchronized to a proventic source, when the values have changed and about once per day after that, even if these values have not changed. During protocol operations, each of these values and associated signatures and timestamps are returned in the associated request or response message. While there are in fact several public value signatures, depending on the number of entries on the certificate list, the values are all signed at the same time, so there is only one public value timestamp.

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

All cryptographic values used by the protocol are time sensitive and are regularly refreshed. In particular, files containing cryptographic basis values used by signature and encryption algorithms are regenerated from time to time. It is the intent that file regenerations occur without specific advance warning and without requiring prior distribution of the file contents. While

cryptographic data files are not specifically signed, every file is associated with a filestamp in the form of the NTP seconds at the creation epoch. It is not the intent in this document to specify file formats or names or encoding rules; however, whatever conventions are used must support a NTP filestamp in one form or another. Additional details specific to the reference implementation are in Appendix I.

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

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 insure new data are consistent. As the data are forwarded from server to client, the filestamps are preserved, including those for certificate and leap seconds files. Packets with older filestamps are discarded before spending cycles to verify the signature.

C.6 Autokey Protocol Overview

This section presents an overview of the three dances: server, broadcast and symmetric. Each dance is designed to be non intrusive and to require no additional packets other than for regular NTP operations. The NTP and Autokey protocols operate independently and simultaneously and use the same packets. When the preliminary dance exchanges are complete, subsequent packets are validated by the autokey sequence and thus considered proventic as well. Autokey assumes clients poll servers at a relatively low rate, such as once per minute or slower. In particular, it is assumed that a request sent at one poll opportunity will normally result in a response before the next poll opportunity.

The Autokey protocol data unit is the extension field, one or more of which can be piggybacked in the NTP packet. An extension field contains either a request with optional data or a response with data. To avoid deadlocks, any number of responses can be included in a packet, but only one request. A response is generated for every request, even if the requestor is not synchronized to a proventic source, but 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 passes all extension field tests are cycles spent to verify the signature.

All dances begin with the parameter exchange where the client obtains the server host name and status word specifying the digest/signature scheme it will use and the identity schemes it supports. The dance continues with the certificate exchange where the client obtains and verifies the certificates along the trail to a trusted, self-signed certificate usually, but not necessarily, provided by a primary (stratum 1) server. Primary servers are by design proventic with trusted, self-signed certificates.

However, the certificate trail is not sufficient protection against middleman attacks unless an identity scheme such as described in Appendix H or proof-of-possession scheme in [14] is available.

While the protocol for a generic challenge/response scheme is defined in this document, the choice of one or another required or optional identification schemes is yet to be determined. If all certificate signatures along the trail are verified and the server identity is confirmed, the server is declared proventic. Once declared proventic, the client verifies packets using digital signatures and/or the autokey sequence.

Once synchronized to a proventic source, the client continues with the sign exchange where the server acting as CA signs the client certificate. The CA interprets the certificate as a X.509v3 certificate request, but verifies the signature if it is self-signed. The CA extracts the subject, issuer, extension fields and public key, then builds a new certificate with these data along with its own serial number and begin and end times, then signs it using its own public key. The client uses the signed certificate in its own role as CA for dependent clients.

In the server dance the client presents its public key and requests the server to generate and return a cookie encrypted with this key. The server constructs the cookie as described above and encrypts it using this key. The client decrypts the cookie for use in generating the key list. A similar dance is used in symmetric mode, where one peer acts as the client and the other the server. In case of overlapping messages, each peer generates a cookie and the agreed common value is computed as the EXOR of the two cookies.

The cookie is used to generate the key list and autokey values in all dances. In the server dance there is no need to provide these values to the server, so once the cookie has been obtained the client can generate the key list and validate succeeding packets directly. In other dances the client requests the autokey values from the server or, in some modes, the server provides them as each new key list is generated. Once these values have been received, the client validates succeeding packets using the autokey sequence as described previously.

A final exchange occurs when the server has the leap seconds table, as indicated in the host status word. If so, the client requests the table and compares the filestamp with its own leap seconds table filestamp, if available. If the server table is newer than the client table, the client replaces its table with the server table. The client, acting as server, can now provide the most recent table to any of its dependent clients. In symmetric mode, this results in both peers having the newest table.

C.7 Autokey State Machine

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

C.8 Status Word

Each server and client operating also as a server implements a host status word, while each client implements a server status word for each server. Both words have the format and content shown below.

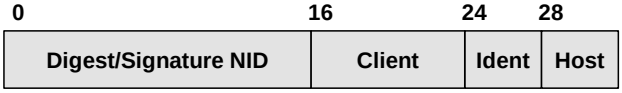

Figure 17. Status Word

The low order 16 bits of the status word define the state of the Autokey protocol, while the high order 16 bits specify the message digest/signature encryption scheme. Bits 24-31 of the status word are reserved for server use, while bits 16-23 are reserved for client use. There are four additional bits implemented separately.

The host status word is included in the ASSOC request and response messages. The client copies this word to the server status word and then lights additional status bits as the dance proceeds. Once lit, these bits never come dark unless a general reset occurs and the protocol is restarted from the beginning. The status bits are defined as follows:

`ENB` (31)

Lit if the server implements the Autokey protocol and is prepared to dance. Dim would be very strange.

`LPF` (30)

Lit if the server has loaded a valid leap seconds file. This bit can be either lit or dim.

`IDN` (24-27)

These four bits select which identity scheme is in use. While specific coding for various schemes is yet to be determined, the schemes available in the reference implementation and described in Appendix H include the following.

0x0 Trusted Certificate (TC) Scheme (default)
0x1 Private Certificate (PC) Scheme
0x2 Schnorr aka Identify-Friendly-or-Foe (IFF) Scheme
0x4 Guillard-Quisquater (GC) Scheme
0x8 Mu-Varadharajan (MV) Scheme

The PC scheme is exclusive of any other scheme. Otherwise, the IFF, GQ and MV bits can be lit in any combination.

The server status bits are defined as follows:

`VAL` 0x0100

Lit when the server certificate and public key are validated.

`IFF` 0x0200

Lit when the server identity credentials are confirmed by one of several schemes described later.

`PRV` 0x0400

Lit when the server signature is verified using the public key and identity credentials. Also called the proventic bit elsewhere in this document. When lit, signed values in subsequent messages are presumed proventic, but not necessarily time-synchronized.

`CKY` 0x0800

Lit when the cookie is received and validated. When lit, key lists can be generated.

AUT 0x1000

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

SGN 0x2000

Lit when the host certificate is signed by the server.

LPT 0x4000

Lit when the leap seconds table is received and validated.

There are four additional status bits LST, LBK, DUP and SYN not included in the status word. All except SYN are association properties, while SYN is a host property. These bits may be lit or dim as the protocol proceeds; all except LST are active whether or not the protocol is running. LST is lit when the key list is regenerated and signed and comes dim after the autokey values have been transmitted. This is necessary to avoid livelock under some conditions. SYN is lit when the client has synchronized to a proventic source and never dim after that. There are two error bits: LBK indicates the received packet does not match the last one sent and DUP indicates a duplicate packet. These bits, which are described in Appendix F, are lit if the corresponding error has occurred for the current packet and dim otherwise.

C.8.1 Host State Variables

Host Name

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

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 used to encrypt/decrypt cookies. This is also the default sign key.

Sign Key

The RSA or DSA public/private key 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 signature before encryption.

IFF Parameters

The parameters used in the IFF identity scheme described in Appendix H.

GQ Parameters

The parameters used in the GQ identity scheme described in Appendix H.

MV Parameters

The parameters used in the MV identity scheme described in Appendix H.

Server Seed

The private value hashed with the IP addresses to construct the cookie.

Certificate Information Structure (CIS)

Certificates are used to construct certificate information structures (CIS) which are stored on the certificate list. The structure includes certain information fields from an X.509v3 certificate, together with the certificate itself encoded in ASN.1 syntax. Each structure carries the public value timestamp and the filestamp of the certificate file where it was generated. Elsewhere in this document the CIS will not be distinguished from the certificate unless noted otherwise.

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

`SIGN` 0x01

The certificate signature has been verified. If the certificate is self-signed and verified using the contained public key, this bit will be lit when the CIS is constructed.

`TRST` 0x02

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 will be lit when the CIS is constructed.

`PRIV` 0x04

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 will be lit when the CIS is constructed.

`ERRR` 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 certificate file. Additional CIS entries are pushed on the list as certificates are obtained from the servers during the certificate exchange. CIS entries are discarded if overtaken by newer ones or expire due to old age.

Host Certificate

The self-signed X.509v3 certificate for the host. The subject and issuer fields consist of the host name, while the message digest/signature encryption scheme consists of the sign key and message digest defined above. Optional information used in the identity schemes is carried in X.509v3 extension fields compatible with [4].

Public Key Values

The public encryption key for the COOKIE request, which consists of the public value of the host key. It carries the public values timestamp and the filestamp of the host key file.

Leapseconds Table Values

The NIST leap seconds table from the NIST leap seconds file. It carries the public values timestamp and the filestamp of the leap seconds file.

C.8.2 Client State Variables (all modes)

Association ID

The association ID used in responses. It is assigned when the association is mobilized.

Server Association ID

The server association ID used in requests. It is initialized from the first nonzero association ID field in a response.

Server Subject Name

The server host name determined in the parameter exchange.

Server Issuer Name

The host name signing the certificate. It is extracted from the current server certificate upon arrival and used to request the next item on the certificate trail.

Server Status Word

The host status word of the server determined in the parameter exchange.

Server Public Key

The public key used to decrypt signatures. It is extracted from the first certificate received, which by design is the server host certificate.

Server Message Digest

The digest/signature scheme determined in the parameter exchange.

Identification Challenge

A 512-bit nonce used in the identification exchange.

Group Key

A 512-bit secret group key used in the identification 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.

Receive Leap second Values

The leap second table returned by a LEAP response, together with its timestamp and filestamp.

Server State Variables (broadcast and symmetric modes)

Send Cookie Values

The cookie encryption values, signature and timestamps.

Send Autokey Values

The autokey values, 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.

C.9 Autokey Messages

There are currently eight Autokey requests and eight corresponding responses. A description of these messages is given below; the detailed field formats are described in Appendix D.

C.9.1 Association Message (ASSOC)

The Association message is used in the parameter exchange to obtain the host name and related values. The request contains the host status word in the filestamp field. The response contains the status word in the filestamp field and in addition the host name as the unterminated string returned by the Unix gethostname() library function. While minimum and maximum host name lengths remain to be established, the reference implementation uses the values 4 and 256, respectively. The remaining fields are defined previously in this document.

If the server response is acceptable and both server and client share the same identity scheme, `ENB` is lit. When the PC identity scheme is in use, the ASSOC response lights `VAL`, `IFF` and `SIG`, since the IFF exchange is complete at this point.

C.9.2 Certificate Message (CERT)

The Certificate message is used in the certificate exchange to obtain a certificate and related values by subject name. The request contains the subject name. For the purposes of interoperability with older Autokey versions, if only the first two words are sent, the request is for the host certificate. The response contains the certificate encoded in X.509 format with ASN.1 syntax as described in Appendix J.

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 either the subject name matches the issuer name, indicating a self-signed certificate, or the `trst` bit is set in the CIS, indicating a trusted certificate. If a trusted certificate is found, the client stops the exchange and lights `VAL`. If a self-signed certificate is found without encountering a trusted certificate, the protocol loops until either a new certificate is signed or timeout.

C.9.3 Cookie Message (COOKIE)

The Cookie message 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 J. The response contains the cookie encrypted by the public key in the request. The signature and timestamps are determined when the cookie is encrypted. If the response is valid, the client lights `CKY`.

C.9.4 Autokey Message (AUTO)

The Autokey message is used to obtain the autokey values. The request contains no value. The response contains two 32-bit words in network order. The first word is the final key ID, while the second is the index of the final key ID. The signature and timestamps are determined when the key list is generated. If the response is valid, the client lights `AUT`.

C.9.5 Leapseconds Table Message (LEAP)

The Leapseconds Table message is used to exchange leap seconds tables. The request and response messages have the same format, except that the `R` bit is dim in the request and lit in the response. Both the request and response contains the leap seconds table as parsed from the leap seconds file from NIST. If the client already has a copy of the leap seconds data, it uses the one with the latest filestamp and discards the other. If the response is valid, the client lights `LPT`.

C.9.6 Sign Message (SIGN)

The Sign message requests the server to 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 J. The response contains the client certificate signed by the server private key. If the certificate is valid when received by the client, it is linked in the certificate list and the client lights `SGN`.

C.9.7 Identity Messages (IFF, GQ, MV)

The request contains the client challenge, usually a 160- or 512-bit nonce. The response contains the result of the mathematical operation defined in Appendix H. The Response is encoded in ASN.1 syntax as described in Appendix G. The response signature and timestamp are determined when the response is sent. If the response is valid, the client lights `IFF`.

C.10 Protocol State Transitions

The protocol state machine is very simple but robust. The state is determined by the server status bits defined above. The state transitions of the three dances are shown below. The capitalized

truth values represent the server status bits. All server bits are initialized dark and light up upon the arrival of a specific response message, as detailed above.

When the system clock is first set and about once per day after that, or when the system clock is stepped, the server seed is refreshed, signatures and timestamps updated and the protocol restarted in all associations. When the server seed is refreshed or a new certificate or leap second table is received, the public values timestamp is reset to the current time and all signatures are recomputed.

C.10.1 Server Dance

The server dance begins when the client sends an ASSOC request to the server. It ends when the first signature is verified and PRV is lit. Subsequent packets received without extension fields are validated by the autokey sequence. An optional LEAP exchange updates the leap seconds table. Note the order of the identity exchanges and that only the first one will be used if multiple schemes are available. Note also that the SIGN and LEAP requests are not issued until the client has synchronized to a proventic source.

```
   while (1) {
      wait_for_next_poll;
      make_NTP_header;
      if (response_ready)
         send_response;
      if (!ENB)                     /* parameters exchange */
         ASSOC_request;
      else if (!VAL)                /* certificate exchange */
         CERT_request(Host_Name);
      else if (IDN & GQ && !IFF)    /* GQ identity exchange */
         GQ_challenge;
      else if (IDN & IFF && !IFF)   /* IFF identity exchange */
         IFF_challenge;
      else if (!IFF)                /* TC identity exchange */
         CERT_request(Issuer_Name);
      else if (!CKY)                /* cookie exchange */
         COOKIE_request;
      else if (SYN && !SIG)         /* sign exchange */
         SIGN_request(Host_Certificate);
      else if (SYN && LPF & !LPT)   /* leapseconds exchange */
         LEAP_request;
}
```

When the PC identity scheme is in use, the ASSOC response lights VAL, IFF and SIG, the COOKIE response lights CKY and AUT and the first valid signature lights PRV.

C.10.2 Broadcast Dance

THe only difference between the broadcast and server dances is the inclusion of an autokey values exchange following the cookie exchange. The broadcast dance begins when the client receives the first broadcast packet, which includes an ASSOC response with association ID. The broadcast client uses the association ID to initiate a server dance in order to calibrate the propagation delay.

The dance ends when the first signature is verified and PRV is lit. Subsequent packets received without extension fields are validated by the autokey sequence. An optional LEAP exchange updates the leapseconds table. When the server generates a new key list, the server replaces the ASSOC response with an AUTO response in the first packet sent.

```
while (1) {
   wait_for_next_poll;
   make_NTP_header;
   if (response_ready)
      send_response;
   if (!ENB)                     /* parameters exchange */
      ASSOC_request;
   else if (!VAL)                /* certificate exchange */
      CERT_request(Host_Name);
   else if (IDN & GQ && !IFF)    /* GQ identity exchange */
      GQ_challenge;
   else if (IDN & IFF && !IFF)   /* IFF identity exchange */
      IFF_challenge;
   else if (!IFF)                /* TC identity exchange */
      CERT_request(Issuer_Name);
   else if (!CKY)                /* cookie exchange */
      COOKIE_request;
   else if (!AUT)                /* autokey values exchange */
      AUTO_request;
   else if (SYN &&! SIG)         /* sign exchange */
      SIGN_request(Host_Certificate);
   else if (SYN && LPF & !LPT)   /* leapseconds exchange */
      LEAP_request;
}
```

When the PC identity scheme is in use, the ASSOC response lights VAL, IFF and SIG, the COOKIE response lights CKY and AUT and the first valid signature lights PRV.

C.10.3 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 the same dance from the beginning. Until the active peer is synchronized to a proventic source (which could be the passive peer) and can sign messages, the passive peer loops waiting for the timestamp in the ASSOC response to light up. Until then, the active peer dances the server steps, but skips the sign, cookie and leapseconds exchanges.

```
while (1) {
   wait_for_next_poll;
   make_NTP_header;
   if (!ENB)                     /* parameters exchange */
      ASSOC_request;
   else if (!VAL)                /* certificate exchange */
      CERT_request(Host_Name);
   else if (IDN & GQ && !IFF)    /* GQ identity exchange */
      GQ_challenge;
   else if (IDN & IFF && !IFF)   /* IFF identity exchange */
      IFF_challenge;
```

```
        else if (!IFF)                  /* TC identity exchange */
           CERT_request(Issuer_Name);
        else if (SYN && !SIG)           /* sign exchange */
           SIGN_request(Host_Certificate);
        else if (SYN && !CKY)           /* cookie exchange */
           COOKIE_request;
        else if (!LST)                  /* autokey values response */
           AUTO_response;
        else if (!AUT)                  /* autokey values exchange */
           AUTO_request;
        else if (SYN && LPF & !LPT)     /* leapseconds exchange */
           LEAP_request;
}
```

When the PC identity scheme is in use, the ASSOC response lights VAL, IFF and SIG, the COOKIE response lights CKY and AUT and the first valid signature lights PRV.

Once the active peer has synchronized to a proventic source, it includes timestamped signatures with its messages. The first thing it does after lighting timestamps is dance the sign exchange so that the passive peer can survive the default identity exchange, if necessary. This is pretty weird, since the passive peer will find the active certificate signed by its own public key.

The passive peer, which has been stalled waiting for the active timestamps to light up, now mates the dance. The initial value of the cookie is zero. If a COOKIE response has not been received by either peer, the next message sent is a COOKIE request. The recipient rolls a random cookie, lights CKY and returns the encrypted cookie. The recipient decrypts the cookie and lights CKY. It is not a protocol error if both peers happen to send a COOKIE request at the same time. In this case both peers will have two values, one generated by itself peer and the other received from the other peer. In such cases the working cookie is constructed as the EXOR of the two values.

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

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

C.11 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 which can occur due to such errors. Appendix F contains an extended discussion on error checking and timestamp validation.

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, manycast or symmetric active packet. A general reset re initializes 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 which maintain the liveness state of the protocol, the 8-bit reachability register defined in [11] and the watchdog timer, which is new in NTPv4. At every poll interval the reachability register is shifted left, the low order bit is dimmed and the high order bit is lost. At the same time the watchdog counter is incremented by one. If an arriving packet passes all authentication and sanity checks, the rightmost bit of the reachability register is lit and the watchdog counter is set to zero. If any bit in the reachability register is lit, the server is reachable, otherwise it is unreachable.

When the first poll is sent by an association, the reachability register and watchdog counter are zero. If the watchdog counter reaches 16 before the server becomes reachable, a general reset occurs. This resets the protocol and clears any acquired state before trying again. If the server was once reachable and then becomes unreachable, a general reset occurs. In addition, if the watchdog counter reaches 16 and the association is persistent, the poll interval is doubled. This reduces the network load for packets that are unlikely to elicit a response.

At each state in the protocol the client expects a particular response from the server. A request is included in the NTP 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 LBK and DUP status bits described above. While this defense can be easily circumvented by a middleman, 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. Following is a list of these situations.

4. When the cookie value changes for any reason.

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

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

7. 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.

C.12 References

1. Adams, C., S. Farrell. Internet X.509 public key infrastructure certificate management protocols. Network Working Group Request for Comments RFC-2510, Entrust Technologies, March 1999, 30 pp.

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

3. Guillou, L.C., and J.-J. Quisquater. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. Proc. *CRYPTO 88 Advanced in Cryptology,* Springer-Verlag, 1990, 216-231.

4. Housley, R., et al. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. Network Working Group Request for Comments RFC-3280, RSA Laboratories, April 2002, 129 pp.

5. Karn, P., and W. Simpson, "Photuris: Session-key Management Protocol", RFC-2522, March 1999.

6. Kent, S., R. Atkinson, "IP Authentication Header," RFC-2402, November 1998.

7. Kent, S., and R. Atkinson, "IP Encapsulating Security Payload (ESP)," RFC-2406, November 1998.

8. Maughan, D., M. Schertler, M. Schneider, and J. Turner, "Internet Security Association and Key Management Protocol (ISAKMP)," RFC-2408, November 1998.

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

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

11. Mills, D.L., "Network Time Protocol (Version 3) Specification, Implementation and Analysis," RFC-1305, March 1992.

12. Mu, Y., and V. Varadharajan. Robust and secure broadcasting. *Proc. INDOCRYPT 2001, LNCS 2247*, Springer Verlag, 2001, 223-231.

13. Orman, H., "The OAKLEY Key Determination Protocol," RFC-2412, November 1998.

14. Prafullchandra, H., and J. Schaad. Diffie-Hellman proof-of-possession algorithms. Network Working Group Request for Comments RFC-2875, Critical Path, Inc., July 2000, 23 pp.

15. Schnorr, C.P. Efficient signature generation for smart cards. *J. Cryptology 4*, 3 (1991), 161-174.

16. Stinson, D.R. *Cryptography - Theory and Practice*. CRC Press, Boca Raton, FA, 1995, ISBN 0-8493-8521-0.

D. Packet Formats

The NTPv4 packet consists of a number of fields made up of 32-bit (4 octet) words in network byte order. The packet consists of three components, the header, one or more optional extension fields and an optional authenticator.

D.1 Header Field Format

The header format is shown below, where the size of some fields is shown in bits if not the default 32 bits.

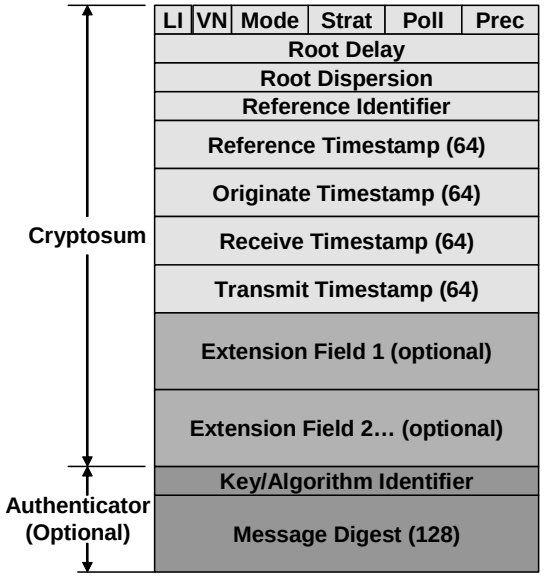

Figure 18. NTP Header Format

The NTP header extends from the beginning of the packet to the end of the Transmit Timestamp field. The format and interpretation of the header fields are backwards compatible with the NTPv3 header fields as described in [11].

A non-authenticated NTP packet includes only the NTP header, while an authenticated one contains in addition an authenticator which takes the form of a message authentication code (MAC). The MAC consisting of the Key ID and Message Digest fields.The format and interpretation of the NTPv4 MAC is described in [11] when using the Digital Encryption Standard (DES) algorithm operating in Cipher-Block Chaining (CBC) node. This algorithm and mode of operation is no longer supported in NTPv4. The preferred replacement in both NTPv3 and NTPv4 is the Message Digest 5 (MD5) algorithm, which is included in both reference implementations. For MD5 the Message Digest field is 4 words (8 octets) and the Key ID field is 1 word (4 octets).

D.2 Extension Field Format

In NTPv4 one or more extension fields can be inserted after the NTP header and before the MAC, which is always present when an extension field is present. The extension fields can occur in any order; however, in some cases there is a preferred order which improves the protocol efficiency.

While previous versions of the Autokey protocol used several different extension field formats, in version 2 of the protocol only a single extension field format is used.

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

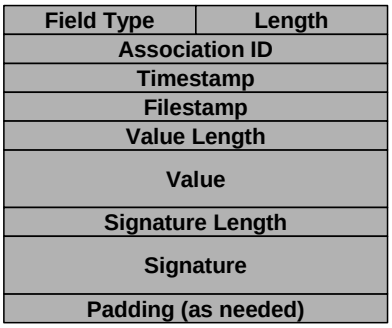

Figure 19. NTP Extension Field Format

Each extension field except the last is zero-padded to a word (4 octets) boundary, while the last is zero-padded to a doubleword (8 octets) boundary. 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.

The presence of the MAC and extension fields in the packet is determined from the length of the remaining area after the header to the end of the packet. The parser initializes a pointer just after the header. If the length is not a multiple of 4, a format error has occurred and the packet is discarded. The following cases are possible based on the remaining length in words.

0
The packet is not authenticated.

4
The packet is an error report or crypto-NAK resulting from a previous packet that failed the message digest check. The 4 octets are presently unused and should be set to 0.

2, 3, 4
The packet is discarded with a format error.

5
The remainder of the packet is the MAC.

>5
One or more extension fields are present.

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

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

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

In some cases not all fields may be present. For requests, until a client has synchronized to a proventic source, signatures are not valid. In such cases the Timestamp and Signature Length fields are 0 and the Signature field is empty. Responses are generated only when the responder has synchronized to a proventic source; otherwise, an error response message is sent. Some request and error response messages carry no value or signature fields, so in these messages only the first two words are present.

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

E. Cryptographic Key and Certificate Management

This appendix describes how cryptographic keys and certificates are generated and managed in the NTPv4 reference implementation. These means are not intended to become part of any standard that may be evolved from this document, but to serve as an example of how these functions can be implemented and managed in a typical operational environment.

The ntp-keygen utility program in the NTP software library generates public/private key files, certificate files, identity parameter files and public/private identity key files. By default the modulus of all encryption and identity keys is 512 bits. All random cryptographic data are based on a pseudo-random number generator seeded in such a way that random values are exceedingly unlikely to repeat. The files are PEM encoded in printable ASCII format suitable for mailing as MIME objects.

Every file has a filestamp, which is a string of decimal digits representing the NTP seconds the file was created. The file name is formed from the concatenation of the host name, filestamp and constant strings, so files can be copied from one environment to another while preserving the original filestamp. The file header includes the file name and date and generation time in printable ASCII. The utility assumes the host is synchronized to a proventic source at the time of generation, so that filestamps are proventic data. This raises an interesting circularity issue that will not be further explored here.

The generated files are typically stored in NFS mounted file systems, with files containing private keys obscured to all but root. Symbolic links are installed from default file names assumed by the NTP daemon to the selected files. Since the files of successive generations and different hosts have unique names, there is no possibility of name collisions.

Public/private keys must be generated by the host to which they belong. OpenSSL public/private RSA and DSA keys are generated as an OpenSSL structure, which is then PEM encoded in ASN.1 syntax and written to the host key file. The host key must be RSA, since it is used to encrypt the cookie, as well as encrypt signatures by default. In principle, these files could be generated directly by OpenSSL utility programs, as long as the symbolic links are consistent. The optional sign key can be RSA or DSA, since it is used only to encrypt signatures.

Identity parameters must be generated by the ntp-keygen utility, since they have proprietary formats. Since these are private to the group, they are generated by one machine acting as a trusted authority and then distributed to all other members of the group by secure means. Public/private identity keys are generated by the host to which they belong along with certificates with the public identity key.

Certificates are usually, but not necessarily, generated by the host to which they belong. The ntp-keygen utility generates self-signed X.509v3 host certificate files with optional extension fields. Certificate requests are not used, since the certificate itself is used as a request to be signed. OpenSSL X.509v3 certificates are generated as an OpenSSL structure, which is then PEM encoded in ASN.1 syntax and written to the host certificate file. The string returned by the Unix gethostname() routine is used for both the subject and issuer fields. The serial number and begin time fields are derived from the filestamp; the end time is one year hence. The host certificate is signed by the sign key or host key by default.

An important design goal is to make cryptographic data refreshment as simple and intuitive as possible, so it can be driven by scripts on a periodic basis. When the ntp-keygen utility is run for the first time, it creates by default a RSA host key file and RSA-MD5 host certificate file and necessary symbolic links. After that, it creates a new certificate file and symbolic link using the existing host key. The program run with given options creates identity parameter files for one or both the IFF or GQ identity schemes. The parameter files must then be securely copied to all other group members and symbolic links installed from the default names to the installed files. In the GQ scheme the next and each subsequent time the ntp-keygen utility runs, it automatically creates or updates the private/public identity key file and certificate file using the existing identity parameters.

F. Autokey Error Checking

Exhaustive examination of possible vulnerabilities at the various processing steps of the NTPv3 protocol as specified in [11] have resulted in a revised list of packet sanity tests. There are 12 tests in the NTPv4 reference implementation, called TEST1 through TEST12, which are performed in a specific order designed to gain maximum diagnostic information while protecting against an accidental or malicious clogging attacks. These tests are described in detail in the NTP software documentation. Those relevant to the Autokey protocol are described in this appendix.

The sanity tests are classified in four tiers. The first tier deflects access control and message digest violations. The second, represented by the autokey sequence, deflects spoofed or replayed packets. The third, represented by timestamped digital signatures, binds cryptographic values to verifiable credentials. The fourth deflects packets with invalid NTP header fields or out of bounds time values. However, the tests in this last group do not directly affect cryptographic protocol vulnerability, so are beyond the scope of discussion here.

F.1 Packet Processing Rules

Every arriving NTP packet is checked enthusiastically for format, content and protocol errors. Some packet header fields are checked by the main NTP code path both before and after the Autokey protocol engine cranks. These include the NTP version number, overall packet length and extension field lengths. Extension fields may be no longer than 1024 octets in the reference implementation. Packets failing any of these checks are discarded immediately. Packets denied by the access control mechanism will be discarded later, but processing continues temporarily in order to gather further information useful for error recovery and reporting.

Next, the cookie and session key are determined and the MAC computed as described above. If the MAC fails to match the value included in the packet, the action depends on the mode and the type of packet. Packets failing the MAC check will be discarded later, but processing continues temporarily in order to gather further information useful for error recovery and reporting.

The NTP transmit and receive timestamps are in effect nonces, since an intruder cannot effectively guess either value in advance. To minimize the possibility that an intruder can guess the nonces, the low order unused bits in all timestamps are obscured with random values. If the transmit timestamp matches the transmit timestamp in the last packet received, the packet is a duplicate, so the DUP bit is lit. If the packet mode is not broadcast and the last transmit timestamp does not match the originate timestamp in the packet, either it was delivered out of order or an intruder has injected a rogue packet, so the LBK bit is lit. Packets with either the DUP or LBK bit lie be discarded later, but processing continues temporarily in order to gather further information useful for error recovery and reporting.

Further indicators of the server and client state are apparent from the transmit and receive timestamps of both the packet and the association. The quite intricate rules take into account these and the above error indicators They are designed to discriminate between legitimate cases where the server or client are in inconsistent states and recoverable, and when an intruder is trying to destabilize the protocol or force consumption of needless resources. The exact behavior is beyond the scope of discussion, but is clearly described in the source code documentation.

Next, the Autokey protocol engine is cranked and the dances evolve as described above. Some requests and all responses have value fields which carry timestamps and filestamps. When the server or client is synchronized to a proventic source, most requests and responses with value fields carry signatures with valid timestamps. When not synchronized to a proventic source, value fields carry an invalid (zero) timestamp and the signature field and signature length word are omitted.

The extension field parser checks that the Autokey version number, operation code and field length are valid. If the error bit is lit in a request, the request is discarded without response; if an error is discovered in processing the request, or if the responder is not synchronized to a proventic source, the response contains only the first two words of the request with the response and error bits lit. For messages with signatures, the parser requires that timestamps and filestamps are valid and not a replay, that the signature length matches the certificate public key length and only then verifies the signature. Errors are reported via the security logging facility.

All certificates must have correct ASN.1 encoding, supported digest/signature scheme and valid subject, issuer, public key and, for self-signed certificates, valid signature. While the begin and end times can be checked relative to the filestamp and each other, whether the certificate is valid relative to the actual time cannot be determined until the client is synchronized to a proventic source and the certificate is signed and verified by the server.

When the protocol starts the only response accepted is ASSOC with valid timestamp, after which the server status word must be nonzero. ASSOC responses are discarded if this word is nonzero. The only responses accepted after that and until the PRV bit is lit are CERT, IFF and GQ. Once identity is confirmed and IFF is lit, these responses are no longer accepted, but all other responses are accepted only if in response to a previously sent request and only in the order prescribed in the protocol dances. Additional checks are implemented for each request type and dance step.

F.2 Timestamps, Filestamps and Partial Ordering

When the host starts, it reads the host key and certificate files, which are required for continued operation. It also reads the sign key and leap seconds files, 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. In general, at the time the files are read, the host is not synchronized, so it cannot determine whether the filestamps are bogus other than these simple checks.

In the following the relation $A \rightarrow B$ 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 $A \leftrightarrow B$; that is, false if the events are simultaneous. For timestamps compared to filestamps and filestamps compared to filestamps, the relation is true if $A \leftrightarrow B$. Note that the current time plays no part in these assertions except in (6) below; however, the NTP protocol itself insures 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 $T0 \rightarrow T1$ and $F0 \rightarrow F1$. The requirement that $T0 \rightarrow T1$ is the primary defense against replays of old messages.

2. For timestamp T and filestamp F, $T \rightarrow T$; that is, the timestamp must not be earlier than the filestamp. 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, $S \rightarrow C \rightarrow D \rightarrow A$; 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, $S \rightarrow C \rightarrow B \rightarrow E$; 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 $C0 \rightarrow C1$.

6. A certificate with begin time B and end time E is invalid and can not be used to sign certificates if $t \rightarrow B$ or $E \rightarrow t$, where t is the current time. Note that the public key previously extracted from the certificate continues to be valid for an indefinite time. This raises the interesting possibilities where a truechimer server with expired certificate or a falseticker with valid certificate are not detected until the client has synchronized to a clique of proventic truechimers.

G. Security Analysis

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

Throughout the following discussion the cryptographic algorithms and private values themselves are assumed secure; that is, a brute force crypt analytic 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 or predict the next autokey. On the other hand, the intruder can remember the totality of all past values for all packets ever sent.

G.1 Protocol Vulnerability

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

The LBK and DUP bits described in the main body and Appendix F are effective whether or not cryptographic means are in use. The DUP bit deflects duplicate packets in any mode, while the LBK bit deflects bogus packets in all except broadcast mode. All packets must have the correct MAC, as verified with correct key ID and cookie. In all modes the next key ID cannot be predicted by a wiretapper, so are of no use for cryptanalysis.

As long as the client has validated the server certificate trail, a wiretapper cannot produce a convincing signature and cannot produce cryptographic values acceptable to the client. As long as the identity values are not compromised, a middleman cannot masquerade as a legitimate group member and produce convincing certificates or signatures. In server and symmetric modes after the preliminary exchanges have concluded, extension fields are no longer used, a client validates the packet using the autokey sequence. A wiretapper cannot predict the next Key IDs, so cannot produce a valid MAC. A middleman cannot successfully modify and replay a message, since he does not know the cookie and without the cookie cannot produce a valid MAC.

In broadcast mode a wiretapper cannot produce a key list with signed autokey values that a client will accept. The most it can do is replay an old packet causing clients to repeat the autokey hash operations until exceeding the maximum key number. However, a middleman could intercept an otherwise valid broadcast packet and produce a bogus packet with acceptable MAC, since in this case it knows the key ID before the clients do. Of course, the middleman key list would eventu-

ally be used up and clients would discover the ruse when verifying the signature of the autokey values. There does not seem to be a suitable defense against this attack.

During the exchanges where extension fields are in use, the cookie is a public value rather than a shared secret and an intruder can easily construct a packet with a valid MAC, but not a valid signature. In the certificate and identity exchanges an intruder can generate fake request messages which may evade server detection; however, the LBK and DUP bits minimize the client exposure to the resulting rogue responses. A wiretapper might be able to intercept a request, manufacture a fake response and loft it swiftly to the client before the real server response. A middleman could do this without even being swift. However, once the identity exchange has completed and the PRV bit lit, these attacks are readily deflected.

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.

G.2 Clogging Vulnerability

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

In order to reduce exposure to decryption attacks the LBK and DUP bits deflect bogus and replayed packets before invoking any cryptographic operations. In order to reduce exposure to encryption attacks, signatures are computed only when the data have changed. For instance, the autokey values are signed only when the key list is regenerated, which happens about once an hour, while the public values are signed only when one of them changes or the server seed is refreshed, which happens about once per day.

In some Autokey dances the protocol precludes server state variables on behalf of an individual client, so a request message must be processed and the response message sent without delay. The identity, cookie and sign exchanges are particularly vulnerable to a clogging attack, since these exchanges can involve expensive cryptographic algorithms as well as digital signatures. A determined intruder could replay identity, cookie or sign requests at high rate, which may very well be a useful munition for an encryption attack. Ordinarily, these requests are seldom used, except when the protocol is restarted or the server seed or public values are refreshed.

Once synchronized to a proventic source, a legitimate extension field with timestamp the same as or earlier than the most recent received of that type is immediately discarded. This foils a middleman cut-and-paste attack using an earlier AUTO response, for example. A legitimate extension field with timestamp in the future is unlikely, as that would require predicting the autokey

sequence. In either case the extension field is discarded before expensive signature computations. This defense is most useful in symmetric mode, but a useful redundancy in other modes.

The client is vulnerable to a certificate clogging attack until declared proventic, after which CERT responses are discarded. Before that, a determined intruder could flood the client with bogus certificate responses and force spurious signature verifications, which of course would fail. The intruder could flood the server with bogus certificate requests and cause similar mischief. Once declared proventic, further certificate responses are discard, so these attacks would fail. The intruder could flood the server with replayed sign requests and cause the server to verify the request and sign the response, although the client would drop the response due invalid timestamp.

An interesting adventure is when an intruder replays a recent packet with an intentional bit error. A stateless server will return a crypto-NAK message which the client will notice and discard, since the LBK bit is lit. However, a legitimate crypto-NAK is sent if the server has just refreshed the server seed. In this case the LBK bit is dim and the client performs a general reset and restarts the protocol as expected. Another adventure is to replay broadcast mode packets at high rate. These will be rejected by the clients by the timestamp check and before consuming signature cycles.

n broadcast and symmetric modes the client must include the association ID in the AUTO request. Since association ID values for different invocations of the NTP daemon are randomized over the 16-bit space, it is unlikely that a bogus request would match a valid association with different IP addresses, for example, and cause confusion.

H. Identity Schemes

The Internet infrastructure model described in [1] is based on certificate trails where a subject proves identity to a certificate authority (CA) who then signs the subject certificate using the CA issuer key. The CA in turn proves identity to the next CA and obtains its own signed certificate. The trail continues to a CA with a self-signed trusted root certificate independently validated by other means. If it is possible to prove identity at each step, each certificate along the trail can be considered trusted relative to the identity scheme and trusted root certificate.

The important issue with respect to NTP and ad hoc sensor networks is the cryptographic strength of the identity scheme, since if a middleman could compromise it, the trail would have a security breach. In electric mail and commerce the identity scheme can be based on handwritten signatures, photographs, fingerprints and other things very hard to counterfeit. As applied to NTP subnets and identity proofs, the scheme must allow a client to securely verify that a server knows the same secret that it does, presuming the secret was previously instantiated by secure means, but without revealing the secret to members outside the group.

While the identity scheme described in RFC-2875 [14] is based on a ubiquitous Diffie-Hellman infrastructure, it is expensive to generate and use when compared to others described here. There are five schemes now implemented in Autokey to prove identity: (1) private certificates (PC), (2) trusted certificates (TC), (3) a modified Schnorr algorithm (IFF aka Identify Friendly or Foe), (4) a modified Guillou-Quisquater algorithm (GQ), and (5) a modified Mu-Varadharajan algorithm (MV). The available schemes are selected during the key generation phase, with the particular scheme selected during the parameter exchange. Following is a summary description of each of these schemes.

The IFF, GQ and MV schemes involve 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 along with a message digest to the client. The client performs a second mathematical operation to produce a digest that must match the one included in the message. To the extent that a server can prove identity to a client without either knowing the group key, these schemes are properly described as zero-knowledge proofs.

H.1 Certificates

Certificate extension fields are used to convey information used by the identity schemes, such as whether the certificate is private, trusted or contains a public identity key. While the semantics of these fields generally conforms with conventional usage, there are subtle variations. The fields used by Autokey Version 2 include:

H.1.1 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.

H.1.2 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.

H.1.3 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.

H.1.4 Subject Key Identifier:

This field contains the public identity key used in the GQ identity scheme. It is present only if the GQ scheme is configured.

H.2 Private Certificate (PC) Scheme

The PC scheme shown below involves the use of a private certificate as group key. A certificate is

Figure 20. Private Certificate (PC)Identity Scheme

designated private by a X509 Version 3 extension field when generated by utility routines in the NTP software distribution. The certificate is distributed to all other group members by secure means and is never revealed inside or outside the group. A client is marked trusted in the Parameter Exchange and authentic when the first signature is verified. This scheme is cryptographically strong as long as the private certificate is protected; however, it can be very awkward to refresh the keys or certificate, since new values must be securely distributed to a possibly large population and activated simultaneously.

The PC scheme uses a private certificate as group key. A certificate is designated private for the purpose of the this scheme if the CIS Private bit is lit. The certificate is distributed to all other group members by secret means and never revealed outside the group. There is no identity exchange, since the certificate itself is the group key. Therefore, when the parameter exchange completes the VAL, IFF and SGN bits are lit in the server status word. When the following cookie exchange is complete, the PRV bit is lit and operation continues as described in the main body of this document.

H.3 Trusted Certificate (TC) Scheme

All other schemes involve a conventional certificate trail as shown below. As described in RFC-

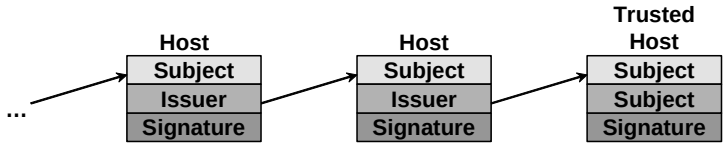

Figure 21. Trusted Certificate (TC) Identity Scheme

2510, 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 by a X509 Version 3 extension field when generated by utility routines in the NTP software distribution. A host obtains the certificates of all other hosts along the trail leading to a trusted host by the Autokey protocol, then requests the immediately ascendant host to sign its certificate. Subsequently, these certificates are provided to descendent hosts by the Autokey protocol. In this scheme keys and certificates can be refreshed at any time, but a masquerade vulnerability remains unless a request to sign a client certificate is validated by some means such as reverse-DNS. If no specific identity scheme is specified in the Identification Exchange, this is the default TC scheme.

The TC identification exchange follows the parameter exchange in which the `VAL` bit is lit. It involves a conventional certificate trail and a sequence of certificates, each signed by an issuer one stratum level lower than the client, and terminating at a trusted certificate, as described in [1]. A certificate is designated trusted for the purpose of the TC scheme if the CIS Trust bit is lit and the certificate is self-signed. Such would normally be the case when the trail ends at a primary (stratum 1) server, but the trail can end at a secondary server if the security model permits this.

When a certificate is obtained from a server, or when a certificate is signed by a server, A CIS for the new certificate is pushed on the certificate list, but only if the certificate filestamp is greater than any with the same subject name and issuer name already on the list. The list is then scanned looking for signature opportunities. If a CIS issuer name matches the subject name of another CIS and the issuer certificate is verified using the public key of the subject certificate, the Sign bit is lit in the issuer CIS. Furthermore, if the Trust bit is lit in the subject CIS, the Trust bit is lit in the issuer CIS.

The client continues to follow the certificate trail to a self-signed certificate, lighting the Sign and Trust bits as it proceeds. If it finds a self-signed certificate with Trust bit lit, the client lights the IFF and PRV bits and the exchange completes. It can, however, happen that the client finds a self-signed certificate with Trust bit dark. This can happen when a server is just coming up, has synchronized to a proventic source, but has not yet completed the sign exchange. This is considered a temporary condition, so the client simply retries at poll opportunities until the server certificate is signed.

H.4 Schnorr (IFF) Scheme

The Schnorr (IFF) identity scheme is useful when certificates are generated by means other than the NTP software library, such as a trusted public authority. In this case a X.509v3 extension field might not be available to convey the identity public key. The scheme involves a set of parameters which persist for the life of the scheme. New generations of these parameters must be securely

transmitted to all members of the group before use. The scheme is self contained and independent of new generations of host keys, sign keys and certificates.

Certificates can be generated by the OpenSSL library or an external public certificate authority, but conveying an arbitrary public value in a certificate extension field might not be possible. The TA generates IFF parameters and keys and distributes them by secure means to all servers, then removes the group key and redistributes these data to dependent clients. Without the group key a client cannot masquerade as a legitimate server.

The IFF parameters are generated by OpenSSL routines normally used to generate DSA keys. By happy coincidence, the mathematical principles on which IFF is based are similar to DSA, but only the moduli *p, q* and generator *g* are used in identity calculations. The parameters hide in a RSA cuckoo structure and use the same members. The values are used by an identity scheme based on DSA cryptography and described in [15] and [16] p. 285. 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 *q*th 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 IFF identity scheme is shown below. The TA generates a DSA parameter structure for use as

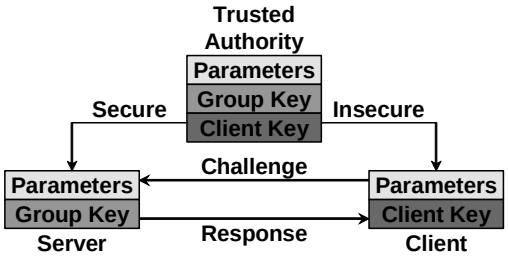

Figure 22. Schnorr (IFF) Identity Scheme

IFF parameters. The IFF parameters are identical to the DSA parameters, so the OpenSSL library DSA parameter generation routine can be used directly. The DSA parameter structure shown in Table is written to a file as a DSA private key encoded in PEM. Unused structure members are set

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

Table 5. IFF Identity Scheme Parameters

to one.

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 $y = k + br \bmod q$ and $x = g^k \bmod p$, then sends (y, hash(x)) to Alice.

3. Alice computes $z = g^y v^r \bmod p$ 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. On success the IFF and PRV bits are lit in the server status word.

H.5 Guillard-Quisquater (GQ) Scheme

The Guillou-Quisquater (GQ) identity scheme is useful when certificates are generated using the NTP software library. These routines convey the GQ public key in a X.509v3 extension field. The scheme involves a set of parameters which persist for the life of the scheme and a private/public identity key, which is refreshed each time a new certificate is generated. The utility inserts the client key in an X.509 extension field when the certificate is generated. The client key is used when computing the response to a challenge. The TA generates the GQ parameters and keys and distributes them by secure means to all group members. The scheme is self contained and independent of new generations of host keys and sign keys and certificates.

The GQ parameters are generated by OpenSSL routines normally used to generate RSA keys. By happy coincidence, the mathematical principles on which GQ is based are similar to RSA, but only the modulus n is used in identity calculations. The parameters hide in a RSA cuckoo structure and use the same members. The values are used in an identity scheme based on RSA cryptography and described in [3] and [16] p. 300 (with errors). The 512-bit public modulus $n = pq$, where p and q are secret large primes. The TA rolls random group key b ($0 < b < n$) and distributes (n, b) to all group members using secure means. The private server key and public client key are constructed later.

The GQ identity scheme is shown below. When generating new certificates, the server rolls new

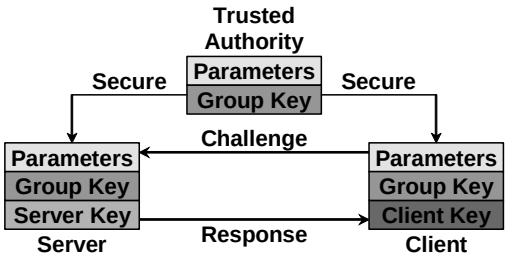

Figure 23. Guillard-Quisquater (GQ) Identity Scheme

random private server key u ($0 < u < n$) and public client key its inverse obscured by the group key $v = (u^{-1})^b \bmod n$. These values replace the private and public keys normally generated by the RSA scheme. In addition, the public client key is conveyed in a X.509 certificate extension. The updated GQ structure shown in Table is written as a 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	server
u	p	server key	server
v	q	client key	client

Table 6. GQ Identity Scheme Parameters

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 $y = ku^r \bmod n$ and $x = k^b \bmod n$, then sends (*y*, hash(*x*)) to Alice.

3. Alice computes $z = v^r y^b \bmod n$ 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. Further evidence is the certificate containing the public identity key, since this is also signed with Bob's private key. On success the IFF and PRV bits are lit in the server status word.

H.6 Mu-Varadharajan (MV) Identity Scheme

The Mu-Varadharajan (MV) scheme was originally intended to encrypt broadcast transmissions to receivers which do not transmit. There is one encryption key for the broadcaster and a separate decryption key for each receiver. It operates something like a pay-per-view satellite broadcasting system where the session key is encrypted by the broadcaster and the decryption keys are held in a tamper proof set-top box. We don't use it this way, but read on.

The MV scheme is perhaps the most interesting and flexible of the three challenge/response schemes. It can be used 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 TA generates an intricate cryptosystem involving public and private encryption keys, together with a number of activation keys and associated private client decryption keys. The activation keys are used by the TA to activate and revoke individual client decryption keys without changing the decryption keys themselves.

The TA provides the server with a private encryption key and public decryption key. The server adjusts the keys by a nonce for each plaintext encryption, so they appear different on each use. The encrypted ciphertext and adjusted public decryption key are provided in the client message. The client computes the decryption key from its private decryption key and the public decryption key in the message.

In the MV scheme the activation keys are known only to the TA. The TA decides which keys to activate and provides to the servers a private encryption key *E* and public decryption keys $\bar{g}$ and

$\hat{g}$ 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 private decryption keys $\bar{x}_j$ and $\hat{x}_j$, which do not need to be changed if the TA activates or deactivates this key. The clients have no further information and, in particular, cannot masquerade as a server or TA.

The MV values hide in a DSA cuckoo structure which uses the same parameters, but generated in a different way. The values are used in an encryption scheme similar to El Gamal cryptography and a polynomial formed from the expansion of product terms $\prod_{0<j\le n}(x-x_j)$, as described in [12]. The paper has significant errors and serious omissions.

The MV identity scheme is shown below. The TA writes the server parameters, private encryption

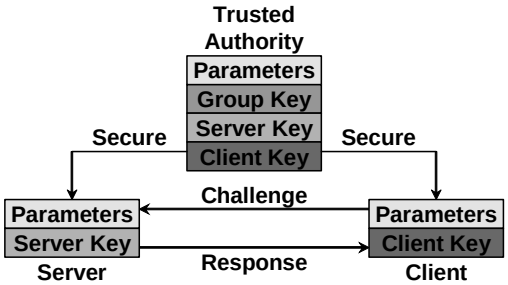

Figure 24. Mu-Varadharajan (MV) Identity Scheme

key and public decryption keys for all servers as a DSA private key encoded in PEM, as shown in the table below.

MV	DSA	Item	Include
p	p	modulus	all
q	q	modulus	server
E	g	private encrypt	server
$\bar{g}$	priv_key	public decrypt	server
$\hat{g}$	pub_key	public decrypt	server

Table 7. MV Scheme Server Parameters

The TA writes the client parameters and private decryption keys for each client as a DSA private key encoded in PEM. It is used only by the designated recipient(s) who pay a suitably outrageous fee for its use. Unused structure members are set to one, as shown in Table.

MV	DSA	Item	Include
p	p	modulus	all
$\bar{x}_j$	priv_key	private decrypt	client
$\hat{x}_j$	pub_key	private decrypt	client

Table 8. MV Scheme Client Parameters

The devil is in the details. Let q be the product of n distinct primes s'_j ($j = 1...n$), where each s'_j, also called an activation key, has m significant bits. Let prime $p = 2q + 1$, so that q and each s'_j divide $p - 1$ and p has $M = nm + 1$ significant bits. Let g be a generator of the multiplicative group Z_p*; that is, gcd(g, $p - 1$) = 1 and $g^q = 1 \bmod p$. We do modular arithmetic over Z_q and then project into Z_p* as powers of g. Sometimes we have to compute an inverse b^{-1} of random b in Z_q, but for that purpose we require gcd(b, q) = 1. We expect M to be in the 500-bit range and n relatively small, like 30. The TA uses a nasty probabilistic algorithm to generate the cryptosystem.

1. Generate the m-bit primes s'_j $(0 < j \le n)$, which may have to be replaced later. As a practical matter, it is tough to find more than 30 distinct primes for $M \approx 512$ or 60 primes for $M \approx 1024$. The latter can take several hundred iterations and several minutes on a Sun Blade 1000.

2. Compute modulus $q = \prod_{0 < j \le n} s'_j$, then modulus $p = 2q + 1$. If p is composite, the TA replaces one of the primes with a new distinct prime and tries again. Note that q will hardly be a secret since p is revealed to servers and clients. However, factoring q to find the primes should be adequately hard, as this is the same problem considered hard in RSA. Question: is it as hard to find n small prime factors totalling M bits as it is to find two large prime factors totalling M bits? Remember, the bad guy doesn't know n.

3. Associate with each s'_j an element s_j such that $s_j s'_j = s'_j \bmod q$. One way to find an s_j is the quotient $s_j = \frac{q + s'_j}{s'_j}$. The student should prove the remainder is always zero.

4. Compute the generator g of Z_p using a random roll such that $\gcd(g, p - 1) = 1$ and $g^q = 1 \bmod p$. If not, roll again.

Once the cryptosystem parameters have been determined, the TA sets up a specific instance of the scheme as follows.

1. Roll n random roots x_j $(0 < x_j < q)$ for a polynomial of order n. While it may not be strictly necessary, Make sure each root has no factors in common with q.

2. Expand the n product terms $\prod_{0 < j \le n} (x - x_j)$ to form $n + 1$ coefficients $a_i \bmod q$ $(0 \le i \le n)$ in powers of x using a fast method contributed by C. Boncelet.

3. Generate $g_i = g^{a_i} \bmod p$ for all i and the generator g. Verify $\prod_{0 \le i \le n,\, 0 < j \le n} g_i^{a_i x_j^{\,i}} = 1 \bmod p$ for all i, j. Note the $a_i x_j^{\,i}$ exponent is computed mod q, but the g_i is computed mod p. Also note the expression given in the paper cited is incorrect.

4. Make master encryption key $A = \prod_{0 < i \le n,\, 0 < j < n} g_i^{x_j} \bmod p$. Keep it around for awhile, since it is expensive to compute.

5. Roll private random group key b ($0 < b < q$), where gcd(b, q) = 1 to guarantee the inverse exists, then compute $b^{-1} \bmod q$. If b is changed, all keys must be recomputed.

6. Make private client keys $\bar{x}_j = b^{-1} \sum_{0 < i \le n,\, i \ne j} x_i^{\,n} \bmod q$ and $\hat{x}_j = s_j x_j^{\,n} \bmod q$ for all j. Note that the keys for the jth client involve only s_j, but not s'_j or s. The TA sends (p, $\bar{x}_j$, $\hat{x}_j$) to the jth client(s) using secure means.

7. The activation key is initially q by construction. The TA revokes client j by dividing q by s'_j. The quotient becomes the activation key s. Note we always have to revoke one key; otherwise, the plaintext and cryptotext would be identical. The TA computes $E = A^s$, $\bar{g} = \bar{x}^s \bmod p$, $\hat{g} = \hat{x}^{sb} \bmod p$ and sends (p, E, $\bar{g}$, $\hat{g}$) to the servers using secure means.

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 $E' = E^k \bmod p$ and public decryption key $\bar{g}' = \bar{g}^k \bmod p$ and $\hat{g}' = \hat{g}^k \bmod p$. He encrypts $x = E'r$ and sends (hash(x), $\bar{g}'$, $\hat{g}'$) to Alice.

3. Alice computes the session decryption key $E'^{-1} = \bar{g}'^{\hat{x}_j} \hat{g}'^{\bar{x}_j} \bmod p$, recovers the encryption key $E' = (E'^{-1})^{-1} \bmod p$, encrypts $z = E'r \bmod p$, then verifies that hash(z) = hash(x).

H.7 Interoperability Issues

A specific combination of authentication scheme (none, symmetric key, Autokey), digest/signature scheme and identity scheme (PC, TC, IFF, GQ, MV) is called a cryptotype, although not all combinations are possible. There may be management configurations where the servers and clients may not all support the same cryptotypes. A secure NTPv4 subnet can be configured in several ways while keeping in mind the principles explained in this section. Note however that some cryptotype combinations may successfully interoperate with each other, but may not represent good security practice.

The cryptotype of an association is determined at the time of mobilization, either at configuration time or some time later when a packet of appropriate cryptotype arrives. When a client, broadcast or symmetric active association is mobilized at configuration time, it can be designated non-authentic, authenticated with symmetric key or authenticated with some Autokey scheme, and subsequently it will send packets with that cryptotype. When a responding server, broadcast client or symmetric passive association is mobilized, it is designated with the same cryptotype as the received packet.

When multiple identity schemes are supported, the parameter exchange determines which one is used. The request message contains bits corresponding to the schemes it supports, while the response message contains bits corresponding to the schemes it supports. The client matches the server bits with its own and selects a compatible identity scheme. The server is driven entirely by the client selection and remains stateless. When multiple selections are possible, the order from most secure to least is GC, IFF, TC. Note that PC does not interoperate with any of the others, since they require the host certificate which a PC server will not reveal.

Following the principle that time is a public value, a server responds to any client packet that matches its cryptotype capabilities. Thus, a server receiving a non-authenticated packet will respond with a non-authenticated packet, while the same server receiving a packet of a cryptotype it supports will respond with packets of that cryptotype. However, new broadcast or manycast client associations or symmetric passive associations will not be mobilized unless the server supports a cryptotype compatible with the first packet received. By default, non-authenticated associations will not be mobilized unless overridden in a decidedly dangerous way.

Some examples may help to reduce confusion. Client Alice has no specific cryptotype selected. Server Bob supports both symmetric key and Autokey cryptography. Alice's non-authenticated packets arrive at Bob, who replies with non-authenticated packets. Cathy has a copy of Bob's symmetric key file and has selected key ID 4 in packets to Bob. If Bob verifies the packet with key ID 4, he sends Cathy a reply with that key. If authentication fails, Bob sends Cathy a thing called a crypto-NAK, which tells her something broke. She can see the evidence using the utility programs of the NTP software library.

Symmetric peers Bob and Denise have rolled their own host keys, certificates and identity parameters and lit the host status bits for the identity schemes they can support. Upon completion of the parameter exchange, both parties know the digest/signature scheme and available identity schemes of the other party. They do not have to use the same schemes, but each party must use the digest/signature scheme and one of the identity schemes supported by the other party.

It should be clear from the above that Bob can support all the girls at the same time, as long as he has compatible authentication and identification credentials. Now, Bob can act just like the girls in his own choice of servers; he can run multiple configured associations with multiple different servers (or the same server, although that might not be useful). But, wise security policy might preclude some cryptotype combinations; for instance, running an identity scheme with one server and no authentication with another might not be wise.

I. File Examples

This appendix shows the file formats used by the OpenSSL library and the reference implementation. These are not included in the specification and are given here only as examples. In each case the actual file contents are shown followed by a dump produced by the OpenSSL asn1 program.

I.1 RSA-MD5cert File and ASN.1 Encoding

```
# ntpkey_RSA-MD5cert_whimsy.udel.edu.3236983143
# Tue Jul 30 01:59:03 2002
-----BEGIN CERTIFICATE-----
MIIBkTCCATugAwIBAgIEwPBxZzANBgkqhkiG9w0BAQQFADAaMRgwFgYDVQQDEw93
aGltc3kudWRlbC5lZHUwHhcNMDIwNzMwMDE1OTA3WhcNMDMwNzMwMDE1OTA3WjAa
MRgwFgYDVQQDEw93aGltc3kudWRlbC5lZHUwWjANBgkqhkiG9w0BAQEFAANJADBG
AkEA2PpOz6toSQ3BtdGrBt+F6cSSde6zhayOwRj5nAkOvtQ505hdxWhudfKe7ZOY
HRLLqACvVJEfBaSvE5OFWldUqQIBA6NrMGkwDwYDVR0TAQH/BAUwAwEB/zALBgNV
HQ8EBAMCAoQwSQYDVR0OBEIEQEVFGZar3afoZcHDmhbgiOmaBrtWTlLHRwIJswge
LuqB1fbsNEgUqFebBR1Y9qLwYQUm7ylBD+3z9PlhcUOwtnIwDQYJKoZIhvcNAQEE
BQADQQAVZMiNbYV2BjvFH9x+t0PB9//giOV3fQoLK8hXXpyiAF4KLleEqP13pK0H
TceF3e3bxSRTndkIhklEAcbYXm66
-----END CERTIFICATE-----
```

```
  0:d=0  hl=4 l= 401 cons: SEQUENCE
  4:d=1  hl=4 l= 315 cons: SEQUENCE
  8:d=2  hl=2 l=   3 cons: cont [ 0 ]
 10:d=3  hl=2 l=   1 prim: INTEGER:02
 13:d=2  hl=2 l=   4 prim: INTEGER :-3F0F8E99
 19:d=2  hl=2 l=  13 cons: SEQUENCE
 21:d=3  hl=2 l=   9 prim: OBJECT:md5WithRSAEncryption
 32:d=3  hl=2 l=   0 prim: NULL
 34:d=2  hl=2 l=  26 cons: SEQUENCE
 36:d=3  hl=2 l=  24 cons: SET
 38:d=4  hl=2 l=  22 cons: SEQUENCE
 40:d=5  hl=2 l=   3 prim: OBJECT:commonName
 45:d=5  hl=2 l=  15 prim: PRINTABLESTRING :whimsy.udel.edu
 62:d=2  hl=2 l=  30 cons: SEQUENCE
 64:d=3  hl=2 l=  13 prim: UTCTIME:020730015907Z
 79:d=3  hl=2 l=  13 prim: UTCTIME:030730015907Z
 94:d=2  hl=2 l=  26 cons: SEQUENCE
 96:d=3  hl=2 l=  24 cons: SET
 98:d=4  hl=2 l=  22 cons: SEQUENCE
100:d=5  hl=2 l=   3 prim: OBJECT:commonName
105:d=5  hl=2 l=  15 prim: PRINTABLESTRING :whimsy.udel.edu
122:d=2  hl=2 l=  90 cons: SEQUENCE
124:d=3  hl=2 l=  13 cons: SEQUENCE
126:d=4  hl=2 l=   9 prim: OBJECT:rsaEncryption
137:d=4  hl=2 l=   0 prim: NULL
139:d=3  hl=2 l=  73 prim: BIT STRING
214:d=2  hl=2 l= 107 cons: cont [ 3 ]
216:d=3  hl=2 l= 105 cons: SEQUENCE
218:d=4  hl=2 l=  15 cons: SEQUENCE
220:d=5  hl=2 l=   3 prim: OBJECT:X509v3 Basic Constraints
225:d=5  hl=2 l=   1 prim: BOOLEAN:255
228:d=5  hl=2 l=   5 prim: OCTET STRING
```

```
235:d=4  hl=2 l=  11 cons: SEQUENCE
237:d=5  hl=2 l=   3 prim: OBJECT:X509v3 Key Usage
242:d=5  hl=2 l=   4 prim: OCTET STRING
248:d=4  hl=2 l=  73 cons: SEQUENCE
250:d=5  hl=2 l=   3 prim: OBJECT:X509v3 Subject Key Identifier
255:d=5  hl=2 l=  66 prim: OCTET STRING
323:d=1  hl=2 l=  13 cons: SEQUENCE
325:d=2  hl=2 l=   9 prim: OBJECT:md5WithRSAEncryption
336:d=2  hl=2 l=   0 prim: NULL
338:d=1  hl=2 l=  65 prim: BIT STRING
```

I.2 RSAkey File and ASN.1 Encoding

```
# ntpkey_RSAkey_whimsy.udel.edu.3236983143
# Tue Jul 30 01:59:03 2002
-----BEGIN RSA PRIVATE KEY-----
MIIBOgIBAAJBANj6Ts+raEkNwbXRqwbfhenEknXus4WsjsEY+ZwJDr7UOdOYXcVo
bnXynu2TmB0Sy6gAr1SRHwWkrxOThVpXVKkCAQMCQQCQpt81HPAws9Z5NnIElQPx
Lbb5Sc0DyF8rZfu9W18p4Zb5UH3KYqZfAO4K0GTmxuriFphgS9bELSw5L6ow4t6D
AiEA7ACLlKZtCp91CaDohViPhs7KBdRVq7DG9n88z9MM/gMCIQDrXRQMb2dqR/ww
PHJ7aljkhhTE78mxLpn2Po82PfYI4wIhAJ1VsmMZngcU+LEV8FjltQSJ3APi48fL
L07/fd/iCKlXAiEAnOi4CEpE8YVSytL2/PGQmFljLfUxIMm7+X8KJClOsJcCICgU
1w07kRO2ycicL2QRVh8J8vQL68VfH53H+oobKDCd
-----END RSA PRIVATE KEY-----
```

```
  0:d=0  hl=4 l= 314 cons: SEQUENCE
  4:d=1  hl=2 l=   1 prim: INTEGER:00
  7:d=1  hl=2 l=  65 prim: INTEGER:<hex string omitted>
 74:d=1  hl=2 l=   1 prim: INTEGER:03
 77:d=1  hl=2 l=  65 prim: INTEGER:<hex string omitted>
144:d=1  hl=2 l=  33 prim: INTEGER:<hex string omitted>
179:d=1  hl=2 l=  33 prim: INTEGER:<hex string omitted>
214:d=1  hl=2 l=  33 prim: INTEGER:<hex string omitted>
249:d=1  hl=2 l=  33 prim: INTEGER:<hex string omitted>
284:d=1  hl=2 l=  32 prim: INTEGER:<hex string omitted>
```

I.3 IFFpar File and ASN.1 Encoding

```
# ntpkey_IFFpar_whimsy.udel.edu.3236983143
# Tue Jul 30 01:59:03 2002
-----BEGIN DSA PRIVATE KEY-----
MIH4AgEAAkEA7fBvqq9+3DH5BnBScMkruqH4QEB76oec1zjWQ23gyoP2U+L8tHfv
z2LmogOqE1c0McgQynyfQMSDUEmxMyiDwQIVAJ18qdV84wmiCGmWgsHKbpAwepDX
AkA4y42QqZ8aUzQRwkMuYTKbyRRnCG1TJi5eVJcCq65twl5c1bnn24xkbl+FXqck
G6w9NcDtSzuYg1gFLxEuWsYaAkEAjc+nPJR7VY4BjDleVTna07edDfcySl9vy8Pa
B4qArk51LdJlJ49yxEPUxFy/KBIFEHCwRZMc1J7z7dQ/Af26zQIUMXkbVz0D+2Yo
YlG0C/F33Q+N5No=
-----END DSA PRIVATE KEY-----
```

```
  0:d=0  hl=3 l= 248 cons: SEQUENCE
  3:d=1  hl=2 l=   1 prim: INTEGER:00
  6:d=1  hl=2 l=  65 prim: INTEGER:<hex string omitted>
 73:d=1  hl=2 l=  21 prim: INTEGER:<hex string omitted>
 96:d=1  hl=2 l=  64 prim: INTEGER:<hex string omitted>
```

```
162:d=1  hl=2 l=  65 prim: INTEGER:<hex string omitted>
229:d=1  hl=2 l=  20 prim: INTEGER:<hex string omitted>
```

J. ASN.1 Encoding Rules

Certain value fields in request and response messages contain data encoded in ASN.1 distinguished encoding rules (DER). The BNF grammer 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 [2]. The particular algorithms required for conformance are not specified in this document.

J.1 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
}
```

J.2 CERT response, SIGN request and response

The value field contains a X509v3 certificate encoded by the `i2d_X509()` routine in the OpenSSL distribution. The encoding follows the rules stated in [4], including the use of X509v3 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 ::= INTEGER
Validity ::= SEQUENCE {
   notBefore                        UTCTime,
   notAfter                         UTCTime
}
```

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

```
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
}

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

For all certificates, the subject HostName is the unique DNS name of the host to which the public key belongs. The reference implementation uses 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.

17.