Survivable, Real Time Network Services

Defense Advanced Research Projects Agency
Contract F30602-98-1-0225, DARPA Order G409/J175

Option: Autonomous, Distributed, Hierarchical Authentication Scheme

Addendum to Final Report

David L. Mills
Electrical and Computer Engineering Department
University of Delaware
Newark, DE 19716

20 February 2003

1. Introduction

This addendum contains material reformatted from the web pages that are the intended final product of this project. The web pages contain an extensive network of links embedding them not only in the framework of this project, but in the framework of other related projects and reference material. The links are not available in this published document.

The text in this document is generally taken verbatim from the web pages with certain minor changes to enhance readability in paper medium. The figures are taken directly from the pages, but rendered in greyscale. The equations have been redone to conform to FrameMaker sensibilities. The reference and bibliography material has been moved to the end of the base document.

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.

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.

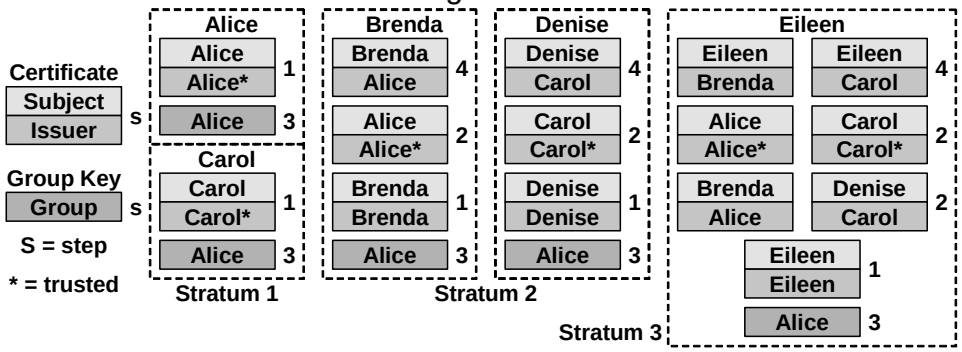
3.1 Certificate Trails

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.



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).

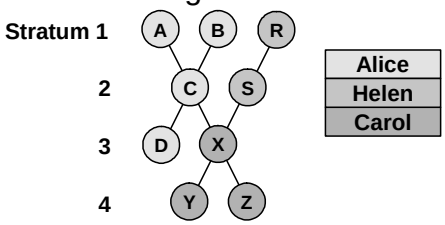
2. A host is trusted if it operates at the lowest stratum in the group and has a trusted, self-signed certificate.

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.



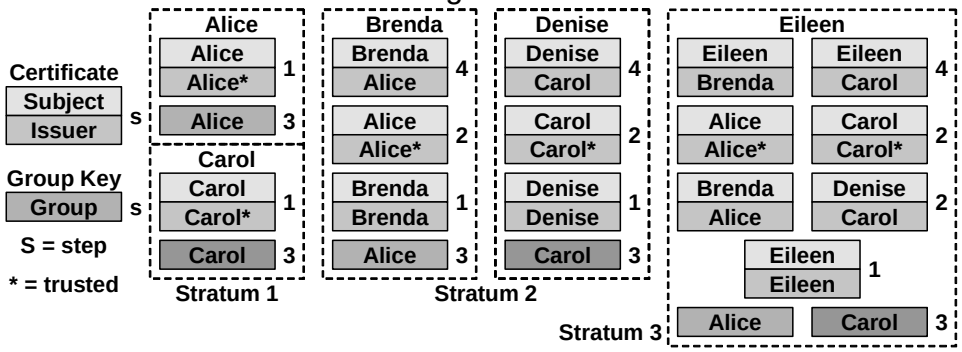
Hosts A, B, C and D belong to the Alice group, hosts R, S to the Helen group and hosts X, Y and Z to the Carol group. While not strictly necessary, hosts A, B and R are stratum 1 and presumed trusted, but the TA generating the group keys could be one of them or another not shown.

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.



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

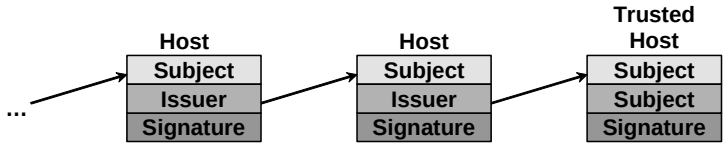
While the identity scheme described in RFC-2875 is based on a ubiquitous Diffie-Hellman infrastructure, it is expensive to generate and use when compared to others described here. There are five schemes now implemented in Autokey to prove identity: (1) private certificates (PC), (2) trusted certificates (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.

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.

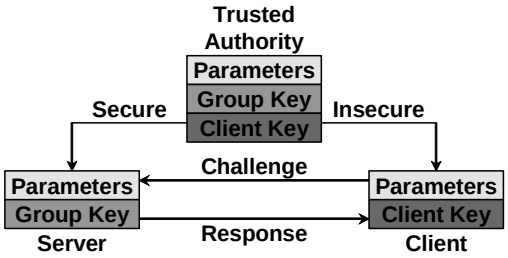
Figure 5.



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.

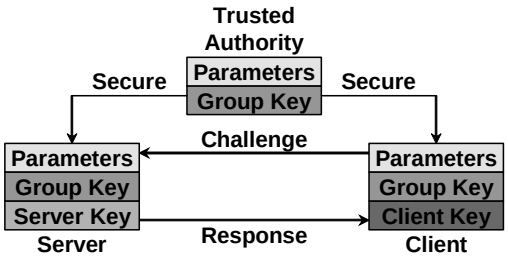
The three remaining schemes IFF, GQ and MV involve a cryptographically strong challenge-response exchange where an intruder cannot learn the group key, even after repeated observations of multiple exchanges. In addition, the IFF and GQ are properly described as zero-knowledge proofs, because the client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

Figure 6.



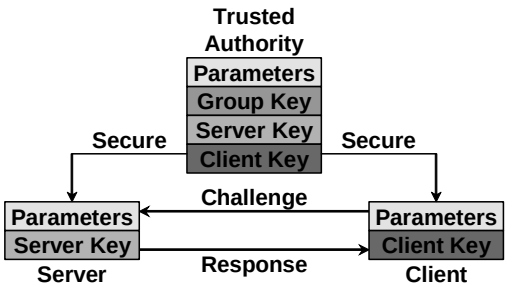
These schemes start when the client sends a nonce to the server, which then rolls its own nonce, performs a mathematical operation and sends the results along with a message digest to the client. The client performs another mathematical operation and verifies the results match the message digest. The 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.



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.



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

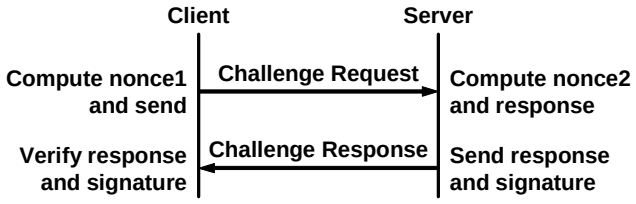
The cryptographic data used by Autokey are generated by the ntp-keygen utility program included in the NTP software distribution. This program generates several files, containing MD5 symmetric keys, RSA and DSA public keys, identity group keys and self signed X.509 Version 3 certificates. The certificate 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.



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.

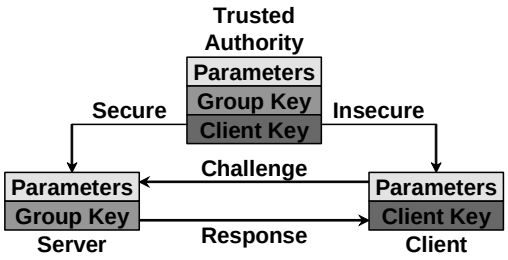
The MV scheme is intended for the most challenging scenarios where it is necessary to protect against both TA and server masquerade. The private values used by the TA to generate the cryptosystem are not available to the servers and the private values used by the servers to encrypt data are not available 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.



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.

Figure 11.

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

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

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

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

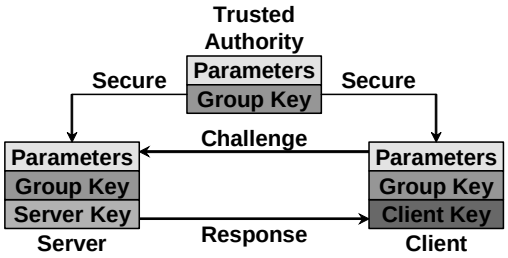
3. Alice computes $z = g^y v^r \bmod p$ and verifies hash(z) equals hash(x).

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 12.



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.

Figure 13.

I

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

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).

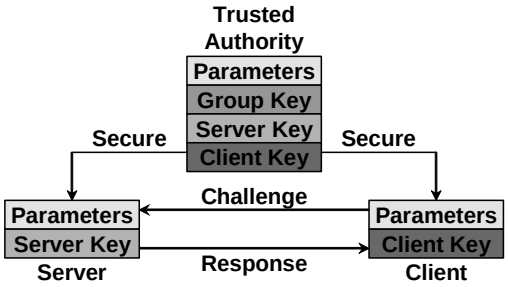
4.3 Mu-Varadharajan (MV) Cryptosystem

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 14.



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

Figure 15.

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

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.

Figure 16.

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

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.

6. Appendix 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.

6.1 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.

6.2 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.

6.3 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.

6.4 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.

6.5 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.

6.6 Error Codes</h4>

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.

6.7 Files

See the ntp-keygen page.

6.8 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.

7. Appendix B. Program Manual Page: ntp-keygen Program

7.1 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]

7.2 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.

7.3 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.

7.4 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.

7.5 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.

7.6 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.

7.7 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.

7.8 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.

7.9 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.