1. Introduction

The Network Time Protocol (NTP) is widely used in the Internet to synchronize computer time to national standards. The current NTP population includes well over 200 primary (stratum-1) servers and 100,000 secondary (stratum-2 and above) servers and clients. It provides comprehensive mechanisms to access national time and frequency dissemination services, organize the time-synchronization subnet and adjust the local clock in each participating subnet peer. The protocol uses redundant servers, diverse network paths and crafted algorithms which cast out incorrect servers and minimize errors due to network delay jitter and local clock frequency instability. The protocol can operate in peer-peer, client-server, and multicast/anycast modes, where the identity of the servers can be cryptographically authenticated. In most places of the Internet of today, NTP provides accuracies of 1-50 ms, depending on the characteristics of the synchronization source and network paths.

The architecture, protocol and algorithms for NTP Version 3 specified in RFC-1305 [12] describes the protocol machine in terms of events, states, transition functions and actions. The protocol is designed for use by clients and servers with a wide range of capabilities and over a wide range of network delays and jitter characteristics. Most users of the Internet NTP synchronization subnet of today use the NTP reference implementation for Unix and Windows, which is a relatively complex, real-time, distributed application. The NTP architecture and protocol model is described in [10] and recent algorithm improvements in [16]. The Simple Network Time Protocol (SNTP), a derivative subset of NTP, is described in [15] and its successor Version 4 is in []. A preliminary outline of NTP Version 4, also the vehicle for this report, is in [14]. Additional information can be found at the NTP home page http://www.eecis.udel.edu/~ntp and the author's home page http://www.eecis.udel.edu/~mills.

A reliable and ubiquitous network time synchronization service such as NTP requires some provision to prevent accidental or malicious attacks on the servers or clients of the service. Reliability requires that clients can determine that received messages 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. The NTP Version 3 specification contains provisions to cryptographically authenticate individual servers; however, the protocol model does not provide a scheme for the distribution of cryptographic keys, nor does it provide for the verification of digital signatures or retrieval of cryptographic certificates that reliably bind the intended server identification data, such as the DNS name, with the associated cryptographic keys and related public values, such as the IP address.

In many ways, the NTP security model is common to many other ubiquitous, distributed applications, such as directory services, web servers and archive repositories. However, an effective authentication scheme for NTP requires it operate efficiently in all modes supported by NTP, including peer-peer, client-server and multicast-anycast modes. Current IETF key-agreement schemes like Photuris [7], SKIP [1] and ISAKMP [8] could be used with NTP as with other protocols in peer-peer modes, where each peer keeps persistent state for the other peer. However, as argued in this report, these schemes are impractical in client-server modes, where persistent state cannot be maintained by servers for client populations which may number in the thousands. A scheme which does not require persistent server state is outlined in this report.

Multicast-anycast modes present a significant challenge to the authentication scheme design. As with client-server modes, servers cannot keep persistent state for each of possibly thousands of peers. In the conventional approach, the natural model for an authentication scheme would be based on public-key cryptography. However, as argued in this report, all known public-key schemes require significant processor resources not likely to be available for the time synchronization function in either the server or its clients. A scheme which avoids most of these resource requirements is outlined in this report.

This report examines the issues incumbent on providing a robust security model and authentication scheme for NTP. It is organized in three main divisions. In the first division, the existing NTP security model and authentication scheme are described in detail. It begins with an overview of the NTP protocol operations, in order to provide a context for the remainder of this report. In the second division, extensions to the existing authentication scheme are proposed which support authentication in client-server and multicast-anycast modes. In client-server modes, the extensions provide for public-key based authentication of private values generated by the server and used by its clients, but the private values do not need to be cached at the server. In multicast-anycast modes, the extensions provide for public-key based authentication of public key values that are used only once, as in the S/KEY scheme [6].

In the third division, a comprehensive review and vulnerability analysis of the extended NTP authentication scheme is given. The analysis includes a vulnerability assessment of the protocol with and without cryptographic authentication, an assessment of the extended authentication system itself, and a summary of vulnerabilities due to possible hardware or software malfunction or design shortcomings.

2. NTP Security Model and Authentication Scheme

The existing NTP security model and authentication scheme were developed some years ago in response to what then were considered likely directions in the evolution of the Internet security model and authentication schemes. Since then, there have been significant developments in the field of cryptographic security protocols and algorithms. In order to understand how the current authentication scheme works and how it can be extended using these algorithms to provide new features, it is necessary to understand how NTP works in the various modes of operation and how previously unknown servers are discovered.

2.1 Protocol Modes of Operation

This section contains an overview of the NTP protocol operations, in particular, the operations in various modes using point-to-point, point-to-multipoint and multipoint-to-point communications paradigms. NTP Version 3 can operate in either peer-peer (point-to-point), unicast (point-to-point) or multicast (point-to-multipoint) modes. As proposed, NTP Version 4 can operate in these modes and anycast (multipoint-to-point) modes as well.

In peer-peer and client-server modes, a unicast client sends a request to a designated server at its unicast address and expects a reply from which it can determine the time and, optionally, the roundtrip delay and local clock offset relative to the server. The client and server end-system addresses are assigned following the usual IPv4, IPv6 [4] or OSI [2] conventions. Synchronization flow is strictly from server to client in client-server modes, but can flow either way in peer-peer modes.

In multicast mode, a multicast server periodically sends a unsolicited message to a designated local broadcast address or multicast group address and ordinarily expects no requests from clients. The server uses a designated local broadcast address or multicast group address. An IP local broadcast address has scope limited to a single IP subnet, since routers do not propagate IP broadcast datagrams. On the other hand, an IP multicast group address has scope extending to potentially the entire Internet.

Multicast clients listen on the designated local broadcast address or multicast group address and ordinarily send no requests. Multicast servers can also respond to client unicast requests, as well as send unsolicited multicast messages. Multicast clients may send unicast requests in order to determine the network propagation delay between the server and client and then continue operation in listen-only mode.

In NTP Version 4, anycast mode is designed for use with a set of cooperating servers whose addresses are not known beforehand by the client. An anycast client sends a request to a designated local broadcast address or multicast group address. One or more anycast servers reply with their individual unicast addresses. In Simple Network Time Protocol (SNTP), which is a subset of the full NTP specification, the client binds to the first reply received, then continues operation in ordinary client-server mode. When multiple replies are received, the client uses the standard suite of NTP algorithms to select the best subset of servers and discard those of lessor quality.

2.2 Security Model

The authentication scheme described in the NTP Version 3 specification RFC-1305 is the basis of the existing NTP security model. The goal in the design is to provide universal access to data products of the protocol, while preventing an intruder from modifying a message from a server or manufacturing a message which is acceptable to a client. It is important to point out at the outset that the NTP security model is intended only to verify that a server is in fact who the client thinks it is and is not an intruder attempting accidently or on purpose to impersonate the server and destabilize it or its clients. It is not necessary, nor would it be politically expedient, to encrypt the timestamps or otherwise hide the data bits in NTP messages, since these are public values. It is not the intent in the model to include access controls; other mechanisms based on IP address and UDP port filtering are available for that. It is not necessarily the case that the model includes protections from message loss, duplication or corruption, since these protections are an intrinsic capability of the NTP protocol itself.

The model is designed so that only the intended server can construct a legitimate message. In peer-peer and client-server modes, the message must be a bona fide reply from the intended server to a specific request previously sent by the client. In multicast-anycast modes, the message must be from a server cryptographically bound to public values which can be verified by digitally signed certificates stored in directory services. While it is not possible in the Internet architecture to completely prevent intercept and replay of legitimate messages, the model is designed to make this difficult by verifying the IP addresses used are in fact assigned to the intended server and its public values.

In the case of client-server and multicast-anycast modes, authentication is a one-way service, with assurance given only to the receiving client or clients that the sending server is authentic. In the case of peer-peer modes, authentication is a two-way service, with assurance given independently to each peer that the other peer is authentic. A consequence of this model is that a server that never sets its clock from a client need not care whether the client is authentic or not and does not have to waste cycles to authenticate client requests. (See later discussion on the hazards of clogging attacks, however.)

It is important to note that the NTP security model specifically recognizes that authentication service may not be continuously available. The NTP synchronization model assumes that individual peers can fail or operate incorrectly or even attempt to modify messages or jam the subnet in one form or another. In addition, transmission lines can fail, routes can change or become congested, and cryptographic keys and even security policies can change while the subnet is in regular, continuous operation. However, the NTP protocol is designed so that, at any one time, the NTP synchronization subnet exists as a set of spanning trees which establishes the flow of synchronization from the primary time servers to all dependent servers and clients in the subnet. In addition, the data structure represented by the certificate chain for each server can span several servers of one kind or another. In order to securely authenticate a server, it must be possible to traverse the structure to the ultimate authority for the server, while insuring the cryptographic keys used are within their declared lifetimes. The implications of these requirements will be explored in following sections.

The NTP security model requires that, if a mixture of authenticated and unauthenticated sources are available, all are maintained in the current association database. This can happen normally when a client first comes up, but has not verified the server credentials, or when a cryptographic key is changed. The unauthenticated sources will still participate in the NTP clock filter and reachability algorithms, but they will not be admitted to the clock selection or combining algorithms, and thus will be unable to affect the choice of which peers will be allowed to discipline the local clock. As argued later, the ability to operate with both authenticated and non-authenticated servers at the same time is necessary. In effect, this requires that a secure NTP client must operate in a multi-level secure environment.

The hierarchical organization of the NTP subnet requires the construction of an unbroken chain of authentication from a given client via intermediate servers to the primary server. The model does this by requiring each client at a given stratum level in the hierarchy to individually authenticate the servers at the next lower stratum level. This requires the client to know whether the server is in fact correctly authenticated from its own sources. While it is in principle possible to individually authenticate each server on the path to the primary server using the NTP control protocol (or presumably its eventual SNMP replacement), this is not considered feasible in the current populous NTP subnet.

As the synchronization subnet, evolves in response to server failures and restarts, prevailing network delay paths, etc., the authentication hierarchy evolves in the same way. It may happen that protocol operations can proceed normally; but, due to temporary lack of cryptographic key material, for example, individual servers may become cryptographically isolated from their sources, even if the synchronization data itself remains valid. If a server or client ordinarily synchronized via authenticated sources loses contact with all of these sources, leaving only unauthenticated sources available, the security model recognizes two philosophical points of view.

In one view, the unauthenticated sources are treated as unreachable and simply disappear from the set of sources available for synchronization, while the server continues as an authenticated source for its dependent clients. In this case, the server disregards the unauthenticated sources and lets the local clock coast until authenticated service is again available. In this case the local clock remains authenticated and the cryptographic database remains unaffected. As required by the specification, the synchronization distance increases with time, reflecting the frequency tolerance of the local oscillator, until a day has passed, when the isolated server or client marks itself unsynchronized and unauthenticated.

In the other view, the isolated server continues operation with unauthenticated and possibly contaminated sources, while announcing to the users and possible clients an unauthenticated condition (for example, in the return code for synchronization-dependent system calls). In this case, the local clock is marked unauthenticated, the local applications are notified, and the all cryptographic media are expunged. If all sources are lost, authenticated or not, the local clock continues to coast as described above.

The above considerations raise some interesting scenarios. Suppose a source has been running in an unauthenticated condition and suddenly becomes authenticated, perhaps as the result of receiving fresh key material. The client learns this upon arrival of a message; however, there may be several samples in the clock filter that have resulted from previous unauthenticated messages. The proper behavior in this case is to expunge all saved data and allow the authenticated data to accumulate in the usual way. This ordinarily results in a delay of up to four messages while the clock filter algorithm accumulates samples and could potentially be exploited by an intruder. These issues will be explored later in this report.

On the other hand, a source may become temporarily isolated from all authenticated sources and, if so configured, continue to operate in an unauthenticated condition. However, there may be several samples in the client clock filter that have resulted from authenticated messages. For reasons discussed later in this report, it is advisable to ignore the unauthenticated samples as long as authenticated ones remain in the clock filter and, only when all samples are shifted out of the filter, start shifting unauthenticated ones in.

In another scenario, suppose some good, low delay paths are available, but for some reason the sources available via those paths are unauthenticated. However, other sources are available which are authentic, but these via much less desirable paths. The security model allows only authenticated sources, if available, so the unauthenticated ones will not be allowed in the clock selection algorithm. Obviously, in such cases the shortest path tree which minimizes the synchronization distance metric will differ between the case where all servers are included and the case where only the authenticated servers are included. Nevertheless, the shortest path tree formed only with authenticated sources will be the one used, as required by the security model.

Now, consider the case where a some set of authenticated peers has survived the clock selection algorithm. From the point of view of the selection algorithm, any of these can be trusted to provide correct time and all can be trusted on a cryptographic basis to be authenticated to the primary server at the root of the NTP subnet. However, the clustering algorithm, which follows the intersection algorithm, will choose a subset of the survivors which provides the best statistically equivalent time. If the client is allowed to operate in an unauthenticated condition following the loss of all authenticated sources, the effect could be a sudden reconfiguration and server reselection, since the spanning trees could be quite different. This is normal behavior and does not raise any particular vulnerability concerns.

2.3 Authentication Scheme

The NTP security model is supported by the current NTP Version 3 authentication scheme, which is modified slightly from the RFC-1305 specification to include provisions for the MD5 message digest algorithm [21], in addition to the DES-CBC algorithm [17], [18]. In this scheme, each association is distinguished by source and destination IP addresses; the port numbers in the UDP header of the NTP message are ignored. Each association is assigned a secret cryptographic session key, which is stored in a secure database. This key is used to construct a message digest (one-way hash function) of the message with either keyed MD5 [9] or DES-CBC. The session key identifier and message digest are stored in the message authentication code (MAC), which is transmitted with the message. The recipient uses the key identifier included in the MAC to retrieve the secret key from its own secure database and verifies the authenticity of the message by computing the message digest and comparing it with the corresponding value included in the message.

In the implementation model assumed by the authentication scheme, the NTP daemon keeps a list of session keys used to construct the message digests for all associations. Individual keys in this list can be marked valid or invalid or replaced by management operations. Each key is assigned a key identifier, which serves a purpose similar to the security parameter identifier (SPI) described in recent internet drafts, and an algorithm identifier, either keyed MD5 or DES-CBC. If two peers are to share a key, they must use the same key identifier and algorithm identifier as well.

The authentication function itself is reasonably fast, even with thousands of clients and in servers providing other functions, such as file sharing, name resolution and security services. The busiest Internet NTP time servers have well over 750 clients producing an average (input plus output) of over ten packets per second. For a Sun IPC workstation, which is slow by today's standards, the average (input plus output) time to service a non-authenticated packet is about 1.5 milliseconds; authentication adds about 0.28 milliseconds to this figure. However, even the busiest servers spend not more than about 1.6 percent of the available processing time for all NTP operations.

Use of the existing authentication scheme is optional and selectively configured for each peer association. When configured, the server or client includes a MAC in messages sent to the peer identified in the association. If the scheme is in use, but the sender has no valid key material, the key identifier and algorithm identifier in the MAC are set to default values. By convention, the key identifier associated with the default values is zero and the associated session key contains all zero data bits (with correct parity in the case of DES). This convention is designed to allow the exchange of timestamps in order to verify correct synchronization before the exchange of certificates and keys has been completed. This convention also reduces the exposure of cryptographic media during possibly long intervals when a client may be severed from properly authenticated servers. In addition, use of the default key allows a client to independently verify the message digest, in order to reduce risk due to message modification, and to verify the authentication machinery is working correctly.

A given client can configure some of its servers to use the scheme and others not; a given server can have some of its clients using the scheme and others not. However, a server announces to its clients that it can provide cryptographically authenticated data only if it is in fact synchronized to cryptographically authenticated source(s). If this is not the case, the key identifier and key are set to the default values for all transmitted messages. In any case, a correctly authenticated condition is indicated by the use of other than the default key in the MAC.

An client or server association can operate in either a configured or stateless manner. A configured association is provided with the key and algorithm identifiers at initial startup, although these identifiers can be changed during operation, even from a remote location, using designated cryptographically authenticated management tools. This feature can be used to temporarily disable a faulty server until it can be repaired. A configured client or server uses the identifiers to construct the MAC, consisting of these identifiers and the message digest itself, in messages sent to the server. A stateless server uses the identifiers in the client message to construct the message digest in messages sent to the client. This assumes that the server has the same secret key as the client and uses the same key identifier.

In the present scheme, it is possible to share a single key among a set of servers and clients. It is also possible to engineer some interesting and useful security topologies using this scheme. For example, a closely cooperating clique of primary servers operating in peer-peer modes can share a single key, in order to provide backup for each other if a radio clock fails. This avoids having to distribute a different key for every pairwise association to every server in the clique. In another example, a set of servers can operate in multicast mode with a single key, so that a client population can synchronize to any of them without requiring separate keys for each one. These examples point up the need to authenticate an aggregate of servers as a unit, where it is not necessary to distinguish among the servers in the aggregate, at least not with respect to authentication.

In the present scheme, the key list is specified independently of the association mode. In configured associations, the key identifier is specified explicitly; in non-persistent associations, the key identifier is provided in the client request. In the current reference implementation, the NTP daemon keeps a list of trusted key identifiers which can be updated remotely using cryptographically authenticated management tools. Any key associated with an identifier in that list is accepted; others are rejected. In some configurations, such as those using symmetric-passive or multicast-client modes, there may be a number of keys, each associated with a different server, and the particular key used to set the local clock may not matter, as long as the key belongs to the trusted set.

The hierarchical organization of NTP subnets raises the issue of whether each stratum in the hierarchy is assigned a distinct key, or whether a single key should be used for the entire tree rooted at each primary server. No specific design is intended in the NTP authentication scheme for either of these approaches and either one may be appropriate for an individual instance. The single-key model allows a client greater flexibility if the particular subnet reconfigures as the result of a network outage, for example, but precludes a model as described above, where some providers, for example, may need to segregate access on administrative grounds in order to control resource usage.

2.4 Server Discovery Schemes

An important requirement for the NTP authentication scheme is that it provide for discovery of servers whose identity (e.g., IP address) is not known beforehand. There are several scenarios which provide automatic server discovery and selection for NTP clients with no pre-specified configuration, other than the client IP address and subnet mask or OSI NSAP. Clients can discover servers directly by DNS lookup or vicariously by listening to broadcast or multicast messages sent by gratuitous (and possibly compromised) servers. In the simplest, widely used scheme, the DNS is set up with a common CNAME, like "time.domain.net", and a list of address records for NTP servers in the same domain. Upon resolving time.domain.net and obtaining the list, the client selects a server at random and begins operation in unicast mode with that server.

For a IP subnet or LAN segment with broadcast capability and one or more NTP servers, clients can be configured for broadcast mode with the local broadcast address. In another scenario suitable for an extended network with significant network propagation delays, clients can be configured for multicast mode. As mentioned above, in SNTP Version 4 an anycast mode is available to expose cooperating servers who to not operate in multicast mode. Following the defined protocol, the client binds to the first reply heard and continues operation in unicast mode. In this mode the local clock can be automatically adjusted to compensate for the propagation delay.

It is assumed in this report that the protocol operations necessary to look up the server name and/or address and obtain the credentials are reliable. While the current DNS cannot guarantee this, proposals now in the IETF should result in cryptographically secure DNS services in the near future. In principle, no changes to the NTP security model or authentication scheme are required to support these services. More discussion on these points is given in following sections.

3. Design Issues

While the current NTP security model and authentication scheme have been in use for over a decade, they have several drawbacks, the most serious being the requirement that keys must be securely distributed in advance. There are no provisions in the NTP protocol itself for key distribution or management on the assumption these functions would be provided by a designated protocol other than NTP. Even if such functions were available, the large number of associations, well over 250,000 in the current NTP subnet, would make the management operations to securely distribute keys and manage their lifetimes impossible to sustain.

In addition, the recent addition of multicast and anycast modes raises new issues in the context of secure authentication, where the identity of the servers is not known in advance. The scheme must provide authentication when multicast servers and anycast clients ordinarily listen without sending messages. The existing client-server-based scheme does not work in these modes, since there is only one source/destination address pair and the session key would be by definition a public value. The scheme proposed later in this report is similar to S/KEY, in that a list of session keys is generated by a server and used in reverse order by its clients without revealing the private server value used to generate the list.

The following sections contain an enumeration of several generic issues, including the interaction between cryptographic key media lifetimes and the time synchronization function, retrieval and processing of public values, such as IP addresses, public keys and certificates, and issues in public-key cryptography. Among the conclusions reached are that existing schemes proposed by the IPSEC community, such as Photuris, are either inappropriate for stateless servers or are too slow for accurate timekeeping. Following this discussion is an overview of the proposed new scheme for NTP Version 4. The new scheme is backwards compatible with the original scheme and requires no change to the header formats described in the current specification.

3.1 On Interactions Between Synchronization and Key Lifetimes

A basic rule in all key distribution and management schemes is that cryptographic key and certificate media should have controlled lifetimes. Specific keys should be destroyed and replaced from time to time, in order to frustrate potential cryptanalysis. New keys should not work with old data and old keys should never be used again. This implies a specific vulnerability to an attack on the timekeeping system, specifically NTP. If secure timekeeping is dependent on conventional key management and distribution schemes, which themselves require secure timekeeping, an interesting circularity results. In principle, this circularity can be resolved through the use of special timekeeping hardware present in some drop-in security devices, such as the Fortezza card; however, this hardware is not ubiquitous and even it can fail.

In principle, network time synchronization is not strictly necessary for reliable key distribution and management, as long as network clocks are never reset, always run at approximately the same rate and all media lifetimes are expressed in time intervals, rather than times relative to a common timescale. In general, lifetimes expressed in this way induce a partial order on the expiration events; a total order is not necessary. A specific key material, once created, is given a designated lifetime associated with the material. Conceptually, the lifetime decrements to zero, at which time the material is expunged. It is the responsibility of the entity holding the material to decrement its lifetime at the common rate and provide the remaining lifetime when exchanging the material with its peers. As long as the time to transmit the material across the network is relatively small and the decrement operations absolutely infallible and near a common rate, the timely expiration of the key is assured.

While the above argument suggests that reliable network time synchronization is not strictly necessary for reliable key management, practical considerations require globally synchronized network clocks. For instance, many existing key exchange and certificate schemes specify times relative to Coordinated Universal Time (UTC) and expect the clocks are set by some out-of-band mechanism like eyeball-and-wristwatch. This expectation may be justified in cases where keys and clocks are managed by a single secure service facility, but not in a globally distributed network.

A central key distribution and management facility such as the Kerberos Key Distribution Center (KDC) represents a specific hazard, both as a focal point of intruder attack, and as a single point of failure for key management. In particular, if time service is to be cryptographically authenticated and the KDC cannot in principle hand out keys unless network timestamps are reliable, the NTP keys themselves cannot require KDC services. Thus, it is necessary to evolve a scheme in which the key media required by NTP servers and clients can be distributed and managed independently of a secure KDC, but yet remain dependent on a properly certified authority or web of trust.

However, this raises the issue that NTP may have to function in scenarios where reliable network timekeeping has not been established or when the keys have not yet been certificated. The most common case occurs when a client is first started and before its clock has been set. In this regime, the NTP key distribution and management functions must operate even when key lifetimes are not expressed in time intervals and even before the local clock has been reliably set. Thus, any protocols used by NTP itself to initiate cryptographic associations must not depend on prior key exchanges which are themselves dependent on synchronized clocks.

Whatever solution is devised for this problem, it should not depend on which function - server authentication or key/certificate validation is done first. In the present NTP protocol model, server state variables are developed for each peer separately, including the apparent server time relative to the local clock and related information necessary to calculate the error estimates. This process takes from one to several packet exchanges, in order to suppress outlyers and establish reliable error estimates. While this is going on, it is possible to carry out server authentication, including verifying certificates and calculating keys, using the apparent server time (not the current client local time) for reference.

Only after reliable server time and authenticated server identification has been achieved can the server be admitted to the NTP selection and combining algorithms. Only if all sanity checks implemented in these algorithms succeed is the server considered authentic. Depending upon configuration, a minimum number of peers may have to be authenticated in this way before setting the local clock. The requirement to maintain a distinct timescale specific to each server during this process complicates the specification and implementation, since presently all state variables are computed as offsets relative to the current local clock.

Note that this process has to be done for every server separately. It could happen that a number of potential servers are in various stages of DNS lookup, certificate validation, key computations and time synchronization data acquisition at the same time. In principle, these operations fit right into the existing state machine and error checking paradigm, but result in additional substates due to the intricate operations required.

There is a subtle problem when considering the design of secure DNS and secure RPC services. Ordinarily, clients of these services assume the various cryptographic keys and certificates have enforceable lifetimes; that is, the services will not use keys or certificates that have expired. When used with NTP, no assumption can be made about the lifetimes, since the clocks may not yet be synchronized. In the present approach, this doesn't matter, since synchronizing the clock and authenticating the server are performed independently. Only when the peer logical clock is synchronized, but possibly not authenticated, and the server authenticated, but possibly not synchronized, are the two functions merged as described above. Designers of secure DNS and RPC services must be prepared to deliver the data requested, even if unable to securely authenticate it at the moment.

3.2 On Public-Key and Private-Key Cryptography

In a perfect world with inexhaustible processing time and memory resources, a public-key cryptosystem such as RSA [19] would be a good foundation on which to build the NTP authentication scheme. In a typical scheme such as RSA or El Gamal digital signatures, each server or clique of servers could be assigned a public/private key pair along with other public values, such as the modulus and exponent for the algorithm. The private key is held by the server and never divulged. The server domain name, address and public values are stored in a database, such as the DNS, together with a digitally signed certificate which includes the public values of a trusted agent serving as a notary. As an optional feature (required in some contexts), the certificate can be bound to one or more certificates produced by an appropriate authority or set of introducers. Used in this manner, there is no difference in the application of public-key cryptography to NTP or to any other similar application.

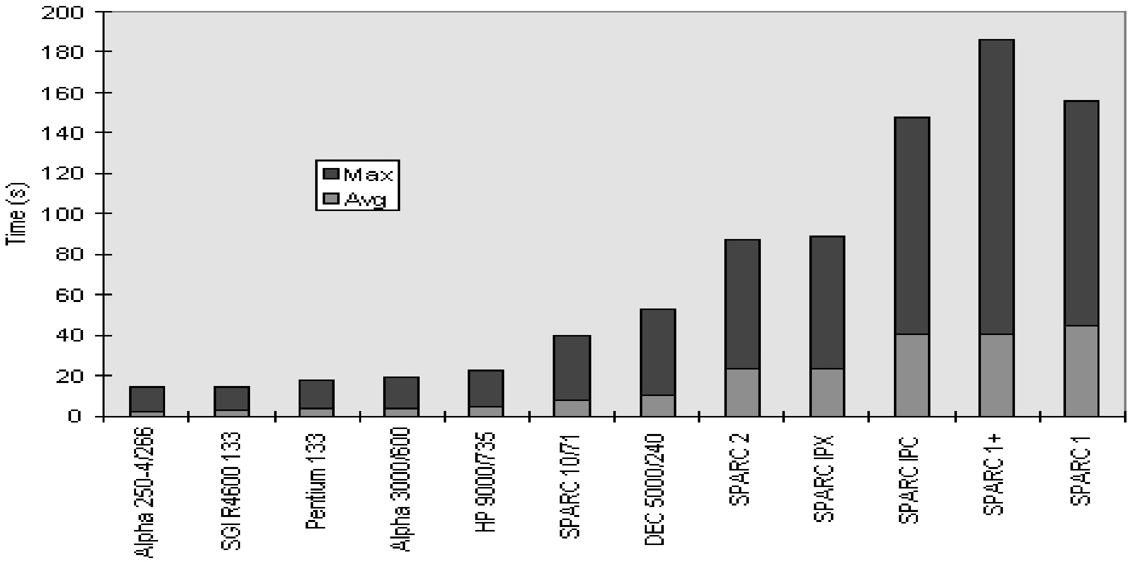

Figure 1. RSA Key Pair Generation

Public-key cryptosystems have the very valuable characteristic that the server and client need share no common state variables other than those declared public values. In the ultimate analysis, the security of such a cryptosystem depends on the integrity of the private value of the public/private key pair. Presumably, the private key is held by a legitimate server and cryptographically bound to a bona fide DNS name, IP address and list of public values. The security of the scheme depends on the fact the private value is never divulged and cannot, except with great effort, be deduced from the public values.

The preferred way to construct a public/private key pair is for the server itself to compute it using the public exponent and modulus values, which are common to all participants in a particular scheme. The processing time to construct the key pair can be significant, ranging from an average of 2.4 s for a Digital 266-MHz Alpha to over 64 s for a Sun SPARC-1 (Figure 1),. The public key is then installed in the secure DNS database along with the other public values. Clients can obtain the public keys, along with optional certificates, from the secure DNS during the name resolution exchange. The procedures to do these things are common to a number of security models other than NTP, so further discussion is beyond the scope of this discussion.

In order to minimize the vulnerability to cryptographic attack, public-key cryptography requires every message to be individually signed using the server private key. In order to reduce the processing requirements to the minimum, the usual practice is to construct the message digest using a one-way hash function such as MD5, then encrypt it using RSA and the server private key. The result is stored in the MAC and transmitted with the message. The client constructs the message digest, then compares it with the MAC decrypted with the server public key.

Constructing the MD5 message digest is a relatively fast operation; for instance, the time to construct the NTP message digest on a Hewlett Packard 9000/735 is 31 us and 263 us on a SPARC 1+ (Figure 2))... However, even when the encrypted data are only a 128-bit MD5 hash, the RSA encryption operation can burn significant quantities of processing time. For instance, the time to

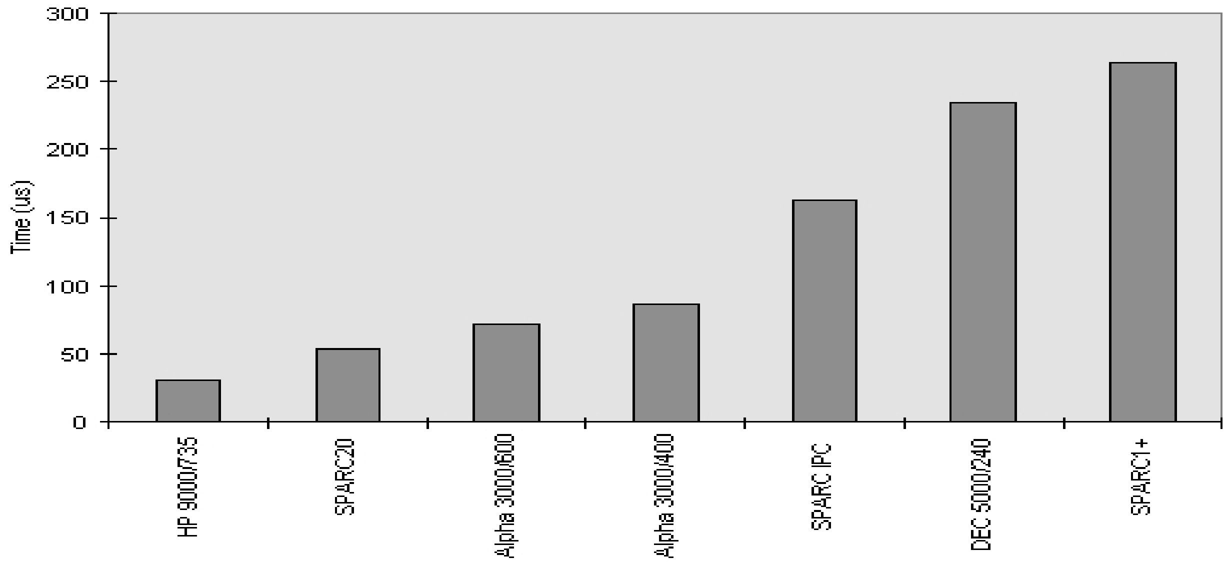

Figure 2. MD5 Hash Algorithm

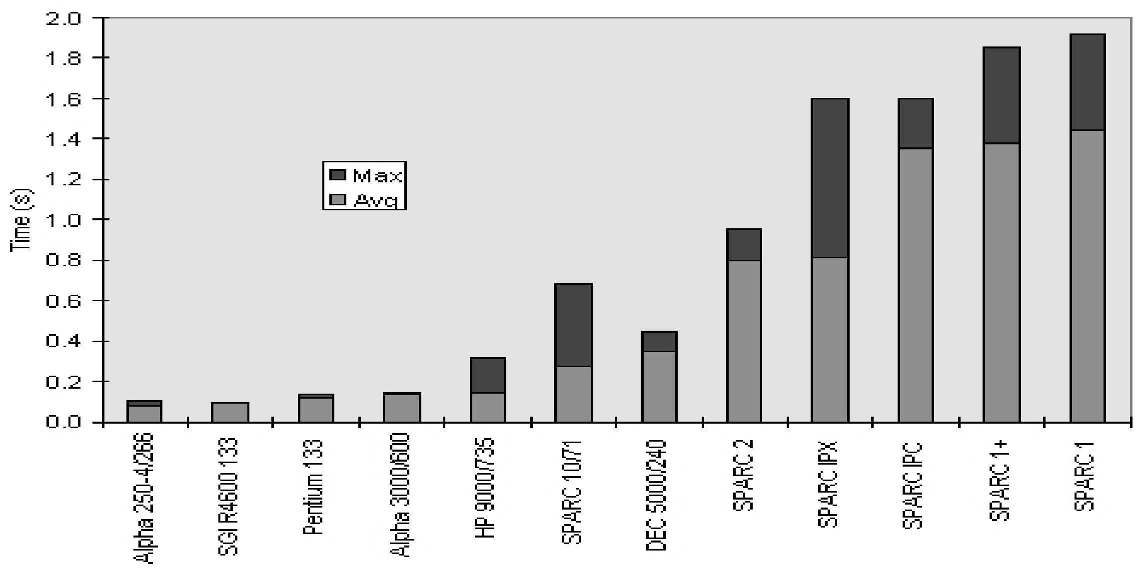

Figure 3. MD5/RSA Sign Algorithm

compute and encrypt a NTP message digest ranges form 80 ms on a Digital 266-MHz Alpha to 2.1 s on a SPARC-1. (Figure 3). The time to compute and decrypt a MD5 hash ranges from 7.9 ms on a Digital 266-MHz Alpha to 201 ms on a Sun SPARC-1 (Figure 4).. In addition, there exists a large variance in running times, depending on the population of one bits in the key and other factors. For example, with random bit strings as keys, a Sun SPARC-1 has a mean decrypting time of 201 ms, minimum 198 ms and maximum 273 ms. While there are other schemes based on public-key cryptography, all are based on computation-intense algorithms and are likely to behave in the

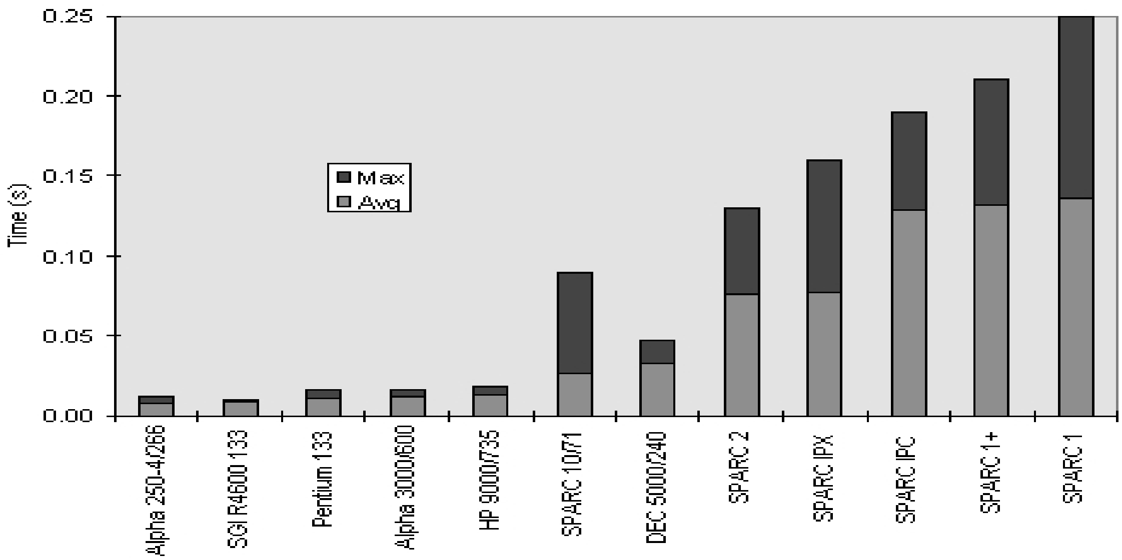

Figure 4. MD5/RSA Verify Algorithm

same way as RSA. Obviously, public-key cryptography is not practical to authenticate every packet.

However, if it is not possible to authenticate every message using this technology, a suitable alternative is to use the technology to construct private values for use as session keys in a private-key cryptosystem. This can be done in two ways: using a key distribution scheme, in which the server generates a session key and transmits it to the client using public-key cryptography, or using a key agreement scheme, in which the server and client jointly agree on a session key using some variant of the Diffie-Hellman protocol [20]. Either way requires servers to hold a distinct key for each client and for clients to hold a distinct key for each server.

Key agreement schemes based on Diffie-Hellman use modular exponentiation operations to compute a shared secret used as a session key. A natural choice would be some variant of the Station-to-Station (STS) protocol, such as Photuris, which uses certificates to avoid man-in-the-middle attacks. Since the numbers involved for the modulus and exponent can be very large (512 bits is typical), these operations are computationally intense and can seriously affect the accuracy of the synchronization function. Accordingly, the NTP authentication scheme should not require their use very often, and then only when the session key or certificate are to be updated. The protocol mechanisms and algorithms to implement a key agreement scheme in NTP peer-peer modes are common to other applications now under study by the IETF, so are considered beyond the scope of this report.

Key distribution schemes involve some kind of trusted agent, such as Kerberos. The trusted agent generates and authenticates session keys (or tickets as in Kerberos). While in principle these schemes can scale well, they provide a critical service which can be vulnerable to various kinds of physical, logical and political attacks. In addition, correct operation of the trusted agent requires secure timekeeping, since the session keys have limited lifetimes which must be respected by the users. As discussed elsewhere in this report, this invites a circularity in service dependence which is hard to resolve. Accordingly, a trusted agent scheme is not considered viable for use in the NTP authentication scheme.

On the other hand, key distribution and agreement protocols which require persistent state are not appropriate for use in NTP client-server and multicast-anycast modes with large numbers of servers and clients. In such cases, the server must operate in a stateless manner and saves no persistent state variables or cryptographic data from one client request to another. Either the server must be able to regenerate the session key as each client request is received, or some means must be provided to authenticate the current session key with respect to a previously used session key which has been cryptographically authenticated. Both of these means have been incorporated in the NTP authentication scheme extensions to be described later in this report.

3.3 On Interactions Between Time, Keys and Certificates

With any scheme, a client needing to authenticate messages from a designated server must obtain the server public values, including its IP address, public key, exponent and modulus. Presumably, these data are obtained during the initial configuration process when the client is started and configured and can be carried out in parallel with regular ongoing NTP operations with other servers. Obtaining the public values may involve additional network operations, such as traversing the DNS system, decrypting signatures, verifying certificates, etc. In principle, provisions must be made to change any of the public values; however, it is anticipated that the need to do this will be relatively infrequent and the computational burden will not affect the accuracy of ongoing NTP operations. Should any of these values change, the natural result is to fail the authentication test, timeout and terminate the association, then attempt to restart it.

A NTP client needing to establish authenticity must individually verify certificates as required on a tentative basis. While this is going on, the certificate servers may also be in process of synchronizing their own clocks, which means the NTP client may have to retry the verification process until all certificate servers on the path have authenticated their own source of network time. A consequence of this is that the authenticated NTP subnet and certificate subnet must be coordinated and configured as a unit; otherwise, black holes and races can occur.

An example may help clarify the problems that can occur in practice. Suppose server A has among its sources server B, which has a valid certificate trail ending in host C. Host C has as its only source server A. Server A and host C have just been rebooted, so have not yet synchronized their clocks or authenticated their sources. Server A cannot reliably authenticate server B, unless it can perform the signature operations involving host C. However, the keys necessary to validate the certificate in C cannot be considered valid until C is itself synchronized to server A and this source is cryptographically authenticated. Thus, a deadlock is formed. It may be hard in practice to detect and avoid such cases, which in principle are similar to routing loops that sometimes occur with routing algorithms.

In multicast-client and anycast-client modes, clients must obtain the credentials by reverse lookup starting from the apparent network address of the sender. This scheme represents a vulnerability, since in principle an intruder can simulate any IP source address. Thus, as described later, it is necessary to authenticate each discovered anycast server using public values cryptographically bound to that server with signed credentials. Note that the credentials themselves cannot be considered reliable until their lifetimes are verified. In principle, lifetimes cannot be trusted, unless the client clock is synchronized; therefore, a bulletproof model requires that the credentials be verified each time the client clock is changed by more than a “safe” interval. These issues will be expanded in later sections of this report.

As mentioned earlier, it is important that every key pair generated carry a reliable lifetime qualification. In order that this requirement be preserved, it is necessary to observe two rules: (1) upon the first use or transmission of a new key, the generator or transmitted must be properly synchronizes and authenticated to its sources, and (2) upon every use of a key, the user must be properly synchronizes and authenticated to its sources. Since certificates also require use of one or more keys, the certificate processor must also be properly synchronizes and authenticated to its sources. The implication of these rules may involve substantial network engineering issues in their own right and thus remain an important topic for further research.

4. Extensions to the NTP Security Model and Authentication Scheme

Certain extensions to the existing NTP security model and authentication scheme are described in following sections. The general principles are that every server must be cryptographically authenticated using a public-key cryptosystem with respect to its name, address, public key, private key and optional set of certificates. The session keys must be developed from both private information, such as the RSA private key and other private random values, as well as public information, such as IP addresses, without revealing the private values themselves. Finally, except in multicast mode, session keys must be private relative to each server and client.

Whatever the mode of operation, peer-peer, client-server or multicast/anycast, the client must obtain the public values from the secure DNS. In the case of configured modes, such as symmetric-active or client modes, the client does this using the standard name resolution procedures. In the case of multicast and anycast client modes, the client does this by reverse lookup from the server address. The procedures to do the name or address lookups is common to other similar applications, so is not discussed further in this report.

The new scheme uses cryptographic message digests in the same way as the original scheme. The contents of the NTP header shown in Figure 5 are hashed with keyed MD5 and a 16-octet session key, yielding a 16-octet message digest. The message authentication code (MAC) transmitted following the header consists of a four-octet key identifier followed by a 16-octet message digest. In the original scheme, the key identifier field is used to select which of possibly many secret keys. The keys themselves are transmitted by a separate method which is not specified. Note that the keyed MD5 algorithm used in both the original and new NTP schemes is not the same as in [9]. In the NTP scheme the message digest is computed on the concatenation of the 16-octet key and the NTP header up to, but not including, the MAC.

In the new scheme, each server maintains a private random value which is used together with a private key and corresponding public values to generate session keys. The private random value is replaced at relatively short intervals, such as a day, depending on the needs of the security model, but never divulged. The private key used in the public-key cryptosystem is replaced at longer intervals, such as a week, since this requires all clients to independently verify its authenticity by possibly tedious operations. A public-key cryptosystem such as RSA is used in a secure RPC to encrypt and decrypt data in some messages exchanged between the server and its clients. In addition, a secure DNS is assumed available from which public keys and certificates can be obtained.

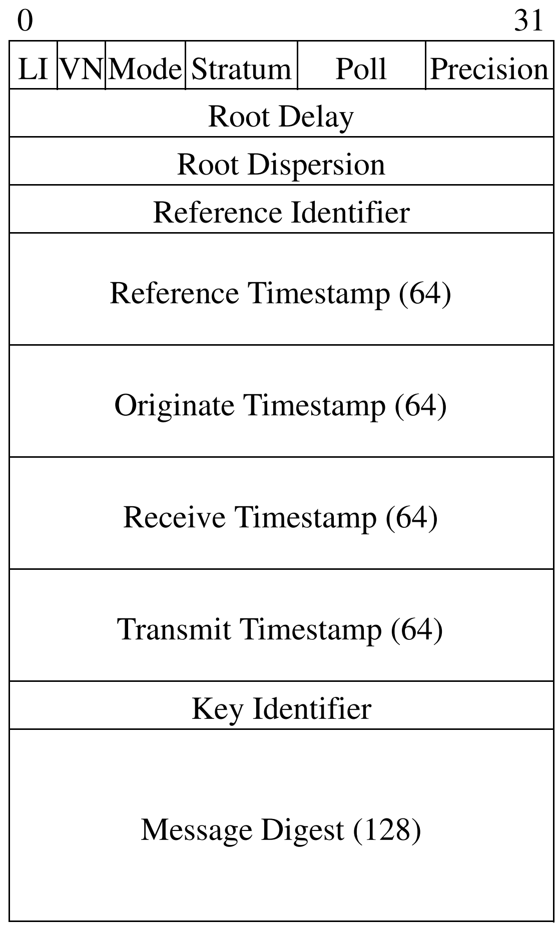

Figure 5. NTP Message Format

The mechanisms used to obtain the public keys and verify the certificates are the subject of current proposals, but are not discussed further in this report.

The scheme works differently for peer-peer, client-server and unicast/anycast modes; however, the message digest is calculated in the same way in all modes. The MD5 algorithm is used to hash the concatenated server private random value, private key, IP source address, IP destination address and key identifier fields. The resulting 16-octet value is the session key used to construct the message digest, which is computed as in the original scheme. Note that the new scheme in effect includes all significant fields of the message, not just the NTP header as in the original scheme, and thus provides additional security.

In the new scheme, as in the old scheme, a key identifier field of all zeros has special significance. In this case, a session key of all zero bits is assumed, which indicates the server is not currently authenticated to its own sources. By convention, values of key identifier values less than 65,536 designate the old scheme and are used as defined in RFC-1305. Values greater than or equal to 65,536 designate the new scheme are used as described in this report.

It is an important characteristic of any scheme that it need use only those cryptographic algorithms likely to be available in the US as well as other countries, since they may be used for purposes other than NTP authentication. Since export controls still exist for DES, the decision has been made to decommission DES for use in any NTP operations. In the extended scheme, the only algorithms required are keyed-MD5, used to construct the message digest, and RSA, used to encrypt, decrypt, sign and verify the session keys. These algorithms, as well as those used to generate the public/private key pairs, are ordinarily found in a public library such as RSAREF [available from the RSA Data Security, Inc., FTP site ftp.rsa.com].

4.1 Secure Network Services - DNS and RPC

The proposed scheme requires a reliable and secure DNS [5] to obtain the server public values, including the DNS name, IP address, public key, RSA exponent and modulus, and optional certificates. Presumably, the availability and authenticity of these data depend on databases accessible via inband or outband mechanisms outside the scope of the scheme described here. Since these data are cryptographically bound together in a web of trust, presumably via diversity paths to redundant servers, their compromise is unlikely.

The proposed scheme depends on the use of secure RPC services, which provide a secure channel to transmit keys, sign and seal messages and related data as required. It is expected that the security of these services will make use of public-key cryptography in order to encrypt, decrypt, sign and verify as necessary. In principle, the secure RPC service could be provided by a generic mechanism used for applications other than NTP; however, in this case the keys used must satisfy the same lifetime constraints outlined above. For the purposes of this report, the secure RPC paradigm presumes only an insecure message-passing function, where messages are encrypted using the RSA algorithm.

As mentioned earlier, there is a subtle problem with secure DNS and secure RPC services. They must be able to operate correctly even if the keys involved in providing the service cannot be trusted as within the specified lifetime of validity. The DNS security model is similar to the NTP model in that it does not provide privacy, since all values returned to clients are public. On the other hand, it is expected to provide secure authentication of the public values, which requires a secure chain of certificates back to the ultimate authority. Since a reliable decryption cannot be done unless the clock is synchronized and the lifetimes verified, this requires the client to fetch all required certificates first, while performing the decryptions on a tentative basis. When sufficient data have accumulated to reliably synchronize the client, the lifetimes must be checked against the tentative time. If all checks agree, the client can declare itself synchronized to its users and other dependent clients.

The secure RPC is used on occasion to provide privacy. As in the secure DNS case, it must be operable even if the client and server clocks are not authenticated; but, in this case, the data transferred must be marked tentative and the client must be prepared to retry the operation until it succeeds or a hard failure is declared. In the schemes described below, the secure RPC is not necessary if the client or server is not synchronized, since in this condition its keys cannot be considered reliable.

While for full functionality, the new scheme requires access to security services, including a secure DNS and a secure RPC based on a public-key cryptosystem such as RSA, a significant improvement over the original scheme is possible using the new scheme without these services. Thus, deployment of the new scheme may proceed without these security services, with the expectation that they will be integrated into the specification and implementations at a later date.

4.2 Authentication in Symmetric (Peer-peer) Modes

The three modes of NTP operation: peer-peer, client-server and multicast-anycast present quite different security models. In peer-peer modes (symmetric-active and symmetric-passive), both peer associations are persistent, so predistributed session keys cause little additional burden other than as now with the current authentication scheme. In the current reference implementation, the keys are stored in a protected file. Presumably, the contents of this file can be accessed and updated by some external means without impinging on the current NTP protocol specification or reference implementation. This can be done using the secure RPC services described above or a key server such as Kerberos. The latter brings with it the hazards mentioned earlier in this report; but, this might be the method of choice during a transition period before completion of full NTP Version 4 and generic Internet services such as secure DNS and secure RPC.

It is also possible in peer-peer modes for each peer to independently authenticate itself to the other peer using the same exchange as for client-server modes, as described below. However, since peer-peer modes are inherently stateful, the session key can be cached without requiring regeneration as each NTP message is received. This requires secure DNS and secure RPC services, but not necessarily a key server or key agreement protocol.

For the strongest intruder protection, a full suite of generic Internet security services is necessary, including a key agreement protocol. These services are presumed available from a system library and run independently of NTP. If these services are available and callable at run time, they can be used to generate keys independently of NTP and without requiring a key server. Since implementation of the required services in peer-peer modes is straightforward and follows the current developmental approach, additional discussion in this report is not necessary.

4.3 Authentication in Client-server Modes

In client-server modes, the server operation is stateless. The requirement for stateless operation extends also to the cryptographic means for authentication. The server cannot remember session keys from one client request to another; therefore, the session key must be regenerated for each received client request. In order to prevent forgery, it must not be possible for an intruder to eavesdrop on an exchange between a client and a legitimate server to mimic the key generation process for that client and server. However, even if session keys are compromised, they are different for each client-server association, so are of no help, except in the case where the intruder forges the source address of a legitimate server and intercepts its client requests. The scheme described below, which was originally suggested by Steven Kent of BBN, requires that the server regenerate a secret key upon each message arrival from the client; however, the computations to regenerate the key are relatively minor.

First, the client sends a request to the server to compute the session key, which is done as above using the private and public values, as described previously. The computed hash is returned to the client encrypted by the server private key and then the client public key, which is included in the client request. This method prevents a man-in-the-middle attack, but does consume significant server and client resources. However, the exchange needs to be done only infrequently when the client is first started up and when the server private values are changed. Alternatively, if the danger of a man-in-the-middle attack can be avoided through the use of secure address filtering, for

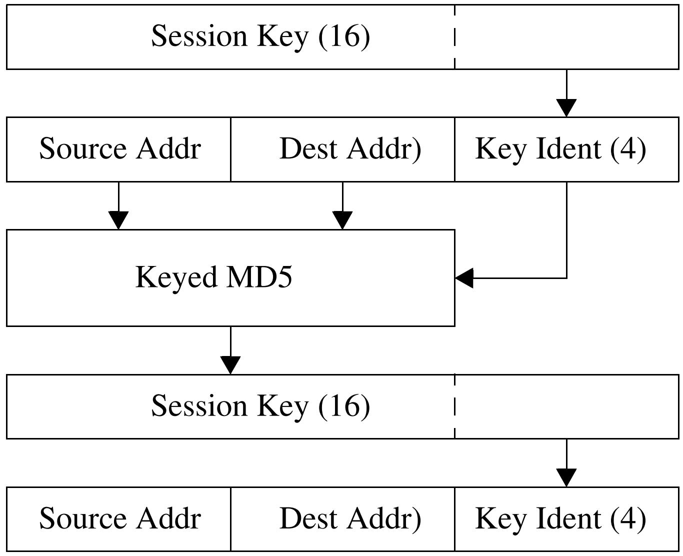

Figure 6. Multicast Authentication

example, the session key can be considered a public value with controlled scope. In this case, the session key can be transmitted as plaintext, since it is not useful for any other source address.

The session key is cached by the client and used by the MD5 algorithm to compute the message digest. The server recomputes the session key as each request is received. This is done exactly as above when generating the hash returned to the client as the session key. The session key and message digest are then computed as above. The session key is unique to the particular server and client involved and need not be retained by the server between requests. Note that an intruder cannot modify and replay a message as valid, even if it forges the source address, since only the server and client can construct the correct session key

The key identifier in the above schemes is considered a public value. Where backwards compatibility is required, this field can be used as in the original scheme to determine the session key, instead of the above operations. Where backwards compatibility is not required, the key identifier field can be filled with a random bit string in order to further confound intruder attack.

4.4 Authentication in Multicast Modes

In multicast mode, a server first calculates a list of 16-octet session keys for later use, as in the S/KEY system [6]. It determines the session key as in client-server mode and uses this as the first entry in the list. The second entry is calculated using keyed MD5 on the concatenated server source address and destination local broadcast or multicast group address. The MD5 key for this operation is constructed from the low order four octets of the first entry replicated four times in the 16-octet key field. Continuing in this way, the server fills the list, which may have from a few to several hundred entries. Figure6 illustrates the procedure.

The server uses the list in inverse order; that is, the last entry is used first, then the next before that, and so on until all entries except the first have been used. At this point, the server generates a new private random value and recomputes the list. Each time the server uses an entry, it stores the low order four octets of the previous (unused) entry in the key identifier field. Formulated in this way, the server does not need to recalculate session keys as they are used.

A client authenticates each message relative to the message that immediately precedes it. It computes the session key and message digest as described above. It then extracts the low order four octets of the session key and compares with the key identifier field in the last message received. If the values agree, the current message is considered valid. If not, a message might have been discarded in transit, so the client hashes again. This procedure may continue for a fixed number of hashes, following which the client abandons the attempt and uses the secure RPC to obtain the current session key.

In order for a new client to cryptographically authenticate the current session key, the server makes it available via secure RPC. This may take some time, due to the expensive computations required, but need be done only infrequently when the server makes a new list or when the client has lost connectivity for a substantial interval. In response to a client RPC request, the server encrypts the current session key with its private key. A client receiving this message decrypts the key with the server public key, perhaps using certificates to establish authenticity, and compares the low order four octets of the result with the next most recent key identifier field. If identical, the client is assured the current key-generating key, and thus all the previous ones, are valid.

The session key applies only to the current message and is not useful for any subsequent message. However, an intruder (man-in-the-middle) could intercept a query response message and learn the current session key, from which all session keys used prior to this message can be determined. While these session keys will not be used again, it is conceivable, although unlikely, that the intruder could trick a client who has not yet heard a prior message into accepting a bogus message. In order to succeed, the intruder would have to impersonate at least the IP source address of any messages it sends to the unsuspecting client, whic is considered possible, but unlikely.

It is important to understand that the session key obtained in this way is not a secret in the ordinary sense, since any client can obtain it or forge it without cryptographic authentication or encryption of any kind, other than knowledge of the server public key. Its purpose is to provide a shared value dependent upon a secret value held only by the server and used in subsequent steps to generate the message digest of each transmitted message. Thus, while the shared value can be obtained by any intruder and used subsequently as a key to generate a message digest, the actual secret used to generate the shared value is not divulged and, presumably, cannot be obtained by an intruder. Neither can future secrets be predicted by an intruder. In this sense, the scheme has perfect forward secrecy.

5. Protocol Security Analysis

There are a number of vulnerabilities with the proposed scheme, some of which are shared with other similar authentication schemes and others unique to the proposed one. The following sections contain a detailed analysis of the security vulnerabilities of the extended NTP protocol, security model and authentication scheme. All are based on the ability of an intruder to outwit the cryptographic means, manufacture fake messages or to intercept, delay and replay valid messages with or without modification. These abilities can be used to impersonate clients or servers in order to destabilize timekeeping functions in a network of NTP servers and clients.

The analysis is focussed primarily on the NTP protocol and authentication scheme. Codebreaking attacks, where the intruder attempts to guess the cryptographic key(s) or private values in use, are beyond the scope of this report, as are attacks on the secure DNS or secure RPC assumed in preceding sections. The analysis is confined to the NTP protocol itself and does not consider the NTP control protocol, which is itself cryptographically protected, but likely to be overtaken by secure SNMP.

5.1 Goals of the Intruder

Before considering in detail the specific vulnerabilities of NTP, it is useful to consider what it is the intruder wants to accomplish. A useful criterion for success in such an attack is the following: Consider the partial ordering of events in all useful scenarios in a particular distributed processing application, such as building an application program from distributed sources, reserving an airline seat, updating a shared file, etc. Therefore, in the ultimate scheme of things, an attack succeeds if it can cause the intended ordering of events to change, resulting in some outcome other than the intended.

It is known from the literature that, if a reliable upper bound exists on the time error between any two clocks in the network and the interval between successive events is greater than this bound, the partial ordering can be preserved. It is possible to guarantee an upper bound on the time error between a pair of clients in the subnet simply by discarding all measurements where the roundtrip delay is above a specified limit [3]. As the limit is reduced in order to improve accuracy, a greater fraction of the measurements is discarded. Thus, there is a tradeoff between the number of samples used to improve the accuracy on a statistical basis and the accuracy achievable. The NTP clock filter algorithm is based on these principles, as described in the specification. In all cases, however, the problem of separating normal jitter during legitimate operations with correctly operating clients or servers from intentional glitches introduced by an intruder attempting to disrupt the timekeeping function remains.

The crafted NTP algorithms filtering, selection, clustering and combining algorithms are designed to discard poor timekeeping data and to distinguish between what are considered truechimers who are correctly synchronized to UTC, and falsetickers who are not. For this purpose, a clock is considered truechimer only if (a) it contains at least one point in its correctness interval in common with a majority of other clocks (intersection algorithm), (b) is correctly authenticated (authentication scheme), (c) is a member of the best subset of truechimers selected on a statistical basis (clustering algorithm), and (d) passes several sanity checks on the various protocol fields. The bottom line then is that an attack succeeds if the intruder can fool all of these algorithms into accepting a real falseticker as a truechimer or rejecting a real truechimer as a falseticker.

It is difficult to determine precisely what kinds of attacks could result in changing the partial ordering of events. A more practical vulnerability assessment would consider what specific attacks could defeat the intended purpose of these algorithms. Some of the attacks may not be immediately apparent; for example, the following are examples of some common successful attacks.

1. An attack succeeds if it results in denial of service for a legitimate server or client by disabling the network, protocol, secure RPC, DNS or any other supporting asset, or as the result of induced congestion or misrouting in the network.

2. An attack succeeds if it results in a legitimate server or client being discarded as unreachable, when in fact it is reachable.

3. An attack succeeds if any of the protocol sanity checks can be defeated, resulting in an otherwise unacceptable server being used or an acceptable server being discarded.

4. An attack succeeds if it results in improper operation of a radio clock or modem; e.g., jamming the radio, cutting the telephone wires.

5. An attack succeeds if it results in a client believing an unauthoritative server to be authorized or an authoritative server to be unauthorized.

6. An attack succeeds if it results in forgery of credentials, addresses or names in the secure DNS, divulging of private data held in secure databases, or forgery of data exchanged between the clients or servers and the secure DNS or secure RPC.

7. An attack succeeds if intruders can conspire to present a sufficient number of falsetickers to dilute the truechimer population to the point that no surviving clique of truechimers remains with at least half the total number of servers.

There are two scenarios where an intruder could affect the quality of synchronization via NTP. In one, the intruder plays the role of man-in-the-middle and can intercept and replay messages between the client and server or even manufacture fake messages that may or may not be believed by the client or server. In the other, the intruder attempts to disrupt server or client operations without necessarily affecting the actual synchronization functions. This could result in legitimate servers being disregarded as unreachable, having degraded accuracy or clogging system time and memory resources.

In following sections a number of specific vulnerabilities are described in which one or more of the above attacks can succeed. In all of these cases, an intruder is assumed capable of otherwise correct operation, but which has incorrect or missing credentials somewhere in the credential verification system.

5.2 Attacks on the Cryptosystem

An obvious attack is to try to break the various cryptographic keys used by the authentication scheme. The authentication schemes discussed in this report are based on well known algorithms such as MD5 and RSA, which have been subject to much scrutiny. No claim is made in this report about the strength or possible weaknesses in these algorithms, other than to estimate the work function to break the code by exhaustive search. A probably more effective strategy is to attack the ancillary resources needed to support the algorithms, in particular the time and memory resources required to generate keys, perform encryption and decryption operations (especially public-key cryptosystems) and support the secure RPC and secure DNS necessary to make the authentication scheme secure.

In general, the work function to break a good cryptographic algorithm is an exponential function of the minimum key length in bits. In the case of the peer-peer and client-server modes described previously, which are essentially private key cryptosystems, the minimum key length is 128 bits. However, the signature scheme used in these modes is vulnerable to a birthday attack, so the equivalent work function is 127 bits. For the purposes this scheme is intended, this value can be considered cryptographically strong.

In multicast mode, the intruder has to guess only 32 bits in order to forge a session key acceptable to the client population. As in the above case, this scheme is vulnerable to a birthday attack, so the equivalent work function is 31 bits. For the purposes this scheme is intended, this value must be considered cryptographically weak. On the other hand, the vulnerability of this scheme to man-in-the-middle attacks probably outweigh the vulnerability due to the weak keys. While it would be simple to extend the key length by changing the packet header format, this would create an incompatibility with older versions of the protocol. The difficulties this would create are believed to outweigh the advantages of a longer key.

5.3 Replay Attacks and Spoofing

A client can conclude that, unless the authentication scheme is compromised, the only messages accepted will be those that have originated at the designated server or are unmodified copies, perhaps delayed, of such messages. A server has no such assurances about messages received from clients, but the server does not ordinarily set its clock from the client, unless it and the client operate in peer-peer modes; but, in this case both the server and client are authenticated to each other. Thus, the only effective action on the part of an intruder is to delay or replay a reply message or delay/spoof/replay a client request message. While the intruder cannot modify the server reply message or try to make the client believe it came from a different server, the intruder can delay and replay the message.

An intruder can intercept, delay and replay messages from the client to the server or from the server to the client. Each of these messages will be properly authenticated, so the delayed replies will be accepted by the client. The result may well be errors in time synchronization which may be hard to detect, since much the same thing could occur as the result of normal network behavior. There is no intrinsic protection against this attack, other than the inherent resistance of the NTP clock filter, selection and clustering algorithms, which rely on the redundancy and diversity of the synchronization subnet topology.

In the Version 3 authentication scheme, an intruder could modify the IP or UDP headers of the NTP client request or server reply, which is not covered by the message digest (but is covered by the UDP pseudo-header checksum). In the extended NTP Version 4 scheme, the IP addresses are included in the message digest, so an intruder can modify them only if in possession of the session key. However, since servers by design do not authenticate clients, an intruder could fake a request message to a server by using a victim client IP address in the IP source address field. This could result in an unexpected, spurious reply to appear at the client. However, as described below, the client can detect and avoid most spoofing and replay attacks, even if the authentication scheme has been compromised.

What remains is to justify the claim that, even in the case where the authentication scheme is not in use or has been compromised, the NTP protocol in and of itself is largely resistant to most spoofing, replay and clogging attacks. Every NTP message sent by a server and client includes a transmit timestamp, which designates the local time the message was sent. A legitimate server returns the transmit timestamp of the client request in the originate timestamp field along with its own transmit timestamp in the reply to a client request. The protocol specification RFC-1305 requires the client to save the original transmit timestamp and compare it to the originate timestamp in the server reply. The message is discarded if the two values disagree. In general, this makes it difficult for an intruder to concoct an otherwise acceptable reply, unless it can intercept the client request and fake the IP addresses in its reply.

In point of fact, the transmit timestamp represents a reasonably good one-time pad, since it is unlikely an intruder can predict it in advance. If it were possible to predict the transmit timestamp and circumvent the authentication scheme, an intruder could create a bogus offset/delay calculation. This is most unlikely using the specified sanity checks, since the precision of the timestamps (232 picoseconds) is far beyond the precision of the local clock in any machine up to and including the fastest likely to exist in the next few decades. A well behaved NTP daemon will in any case fill unused low order bits with random values. Therefore, while not impossible, it is unlikely that an intruder could spoof a request by guessing the next transmit timestamp field. While not in the original NTP specification, an additional check should be done to insure only the first message with the expected value is accepted. This further resists disruptions if an intruder replays client requests (the server replies will all carry the same originate timestamp, but have different transmit timestamps).

The NTP protocol has explicit provisions to resist replay attacks, which could be due to a copy of an old server reply. For those reply messages that pass the spoofing check, the client saves the transmit timestamp. The transmit timestamp in the message is compared with that of the most recent prior reply. The message is discarded if the two values agree, which could be due to a replay of the most recent reply. In this way the client is protected against replay attacks. These and other sanity checks should be done before the computationally expensive authentication check, so that processor cycles lost to clogging attacks can be minimized.

5.4 Man-In-the-Middle Attacks

The above attacks do not in themselves directly affect the quality of synchronization, other than possibly increasing apparent network delays. The attacks summarized in this section assume the ability of the intruder to intercept, modify and replay legitimate messages between the client and the server or between peer servers.

There are two ways in which such an attack could be mounted. In one, the intruder intercepts a valid message from the server to the client, modifies some fields and then sends it, possibly repeating it a number of times, to the client. The objective in this case is to cause the client either to accept the message and compute an incorrect time, or to cause the client to disbelieve time provided by other servers used by the client. In the other way, the intruder intercepts a valid request message from the client to the server, modifies some fields and then sends it, possibly repeated a number of times, to the server. The objectives in this case are to cause the server to emit spurious duplicates or provide incorrect time to the client.

In the case of peer-peer modes, another objective of this attack is to cause the victim to demobilize other legitimate associations or to form spurious associations in the same or another mode. In order to defend against such attacks, the NTP Version 4 specification has been changed so that distinct associations are mobilized for every combination of legitimate modes; thus, it is not possible, for example, to piggyback a spurious client-server association on an ongoing peer-peer one.

Even if it is not possible for the intruder to modify and replay or to spoof valid NTP messages, there is still the possibility of a statistical attack. In this kind of attack, the intruder can intercept a message from either the client or server or both and pass on only those message having a certain quality, while discarding all others. For instance, the intruder can independently calculate the time offsets apparent between the client and server and discard all those messages which show a positive offset or all those with synchronization distance less than some value. The effect would be to introduce a statistical bias in the timekeeping data.

The intruder can also create an artificial bias in the timekeeping data by selectively discarding messages according to a profile, such as synthetic offset spectra, in order to make the client think the server is flaky, erratic or improperly synchronized. A detailed analysis of these attacks is summarized below in the section on attacking specific protocol fields.

The intruder can disable a properly authenticated source by a contrived jamming attack. If it knows the source address, destination address and mode of a properly authenticated message to a client, it can purposely insert an unauthenticated message in the data stream. By the security model described previously, this could cause the client to expunge all unauthenticated data in the clock filter and cause a delay of up to four authenticated messages until the clock filter algorithm again has sufficient data for the clock selection algorithm. The intruder can then completely disable the association simply by allowing another authenticated message and then following it by another unauthenticated one.

In order to protect against this kind of attack, the protocol model should be to disregard unauthenticated messages as long as authenticated ones are in the clock filter. In addition, it is probably a good idea to immediately expunge the clock filter of unauthenticated data when an authenticated message is received.

An intruder attempting to simulate an authenticated server must be able to generate the correct message digest, which means the intruder must learn the session key specific to the victim in client-server modes or the next sequential session key in multicast mode. In a man-in-the-middle attack in client/server mode, the intruder cannot modify and replay a message, unless the server private key and private value are both compromised or the session key itself is compromised. In multicast mode, the intruder cannot generate a correct message digest for the next following message, unless it tricks the client into accepting its own session key list instead of the one produced by the legitimate server. The certificate procedures insure that the key list can come only from the real server.

However, there is a serious vulnerability in multicast mode where the intruder can capture a message from a legitimate server, while preventing its transmission to the client population. The intruder computes the session key and manufacture a bogus message with a message digest acceptable to the client population, then transmits the bogus message to that population. This is an intrinsic weakness shared with any scheme where server messages are not individually signed using public-key cryptography.

5.5 Clogging Attacks (Denial of Service)

In a clogging attack, the intruder attempts to deny service by overwhelming the resources of the client, server or network by generating large volumes of traffic. Even if the attack does not succeed, the result is likely to produce an erosion in timekeeping accuracy if only because of network overload. In fact, it might not be possible, other than by statistical analysis, to determine that such an attack is in progress or the result of normal network loading patterns. In general, it has been the practice at some high profile primary servers to maintain peer associations with some subset of the remaining primary servers in order to record network delay variations on a continuous basis. An attack, usually in the form of a broken client which sends NTP messages continuously, can cause the statistical patterns to change in ways that become clear only upon statistical analysis.

Clogging attacks to not require the intruder to pry open legitimate NTP messages, just the capability to clone an apparently valid NTP message. By its very nature, NTP operating in client-server or multicast modes is designed as a ubiquitous service; that is, a server will ordinarily respond to any client request without necessarily authenticating the request or checking to see if it contains valid data. As the processor cycles necessary to reply to a client request are only modest and no additional state persists after the reply has been generated, the vulnerability of a server to a clogging attack is minimal.

On the other hand, an intruder could attempt to synchronize using peer-peer modes and a bogus source address, thus forcing the server to create an additional association to hold the persistent state variables and then attempt to authenticate the intruder using the expensive public-key schemes as described above. Following this, the server can flood messages to the bogus address as per the NTP specification, which results in additional overhead. Should the intruder continue the attack using new bogus source addresses, the server could eventually run out of storage in which to hold the association variables, or could become mired in the resulting cryptographic overload.

In peer-peer modes, the authentication scheme described previously requires the session key to be determined and distributed in advance using conventional schemes such as Photuris, which itself contains some defensive mechanisms to foil clogging attacks. In these modes, the cryptographic operations in NTP involve only MD5, which does not involve expensive public-key operations. In addition, persistent associations in peer-peer modes require that arriving messages be validated by MD5 before a mobilized association becomes permanent; therefore, a mobilized association can exist only for the period necessary to discover the intruder has incorrect or missing credentials and cache the bogus source address on the access control list.

In another clogging attack, the intruder sends apparently valid (or replies of valid) messages to either a client or a server at a rate greater than the client or server or network ability to process them. The result can be congestion of the network, message buffer overflow at the recipient or any number of related resources starvations. To some extent, these attacks can be mitigated through the use of access control, either at a network node (firewall) or the receiving entity, using the access control features built in the NTP reference implementation. These features provide a mask-and-match function similar to that used in Internet firewalls.

Operational experience with some popular primary (stratum-1) servers with very large client populations has demonstrated that poorly configured client networks can produce a large volume of requests from clients on that network, which can then overwhelm the network paths to a server on another network. This occurs in cases where the client network administrators clone a working copy of a configuration file in hundreds of clients on that network. In order to discourage such practices, the NTP reference implementation caches the source addresses of recent requests and limits the number of clients from any single network to a specified maximum. This is implemented as part of the access control mechanism.

A clogging attack is always possible at some point or other in the secure DNS or secure RPC mechanisms. One point of vulnerability is the original DNS query, if used, since this requires public-key cryptography. This must be dealt with by mechanisms beyond the scope of NTP itself. Another point of vulnerability is the secure RPC exchange which provides the session key to the client. An intruder could flood a server with requests for session keys, which could bog the server down with encryption operations. In client-server mode, an intruder could flood the server with session key requests, which require public-key cryptography to encrypt the session keys. In multicast mode, the session key and its encryption need be computed only when the key list is constructed, not on every request. If the encryption is computed and cached at that time, subsequent requests will incur no additional overhead. In any case, the only effective defense is to identify instances of probable abuse and cache the intruder on the access control list.

An intruder could intercept a server RPC reply and flood the client with repeated copies. Since these messages may require the client to decrypt the reply before determining it is in fact a copy, this could consume significant resources. To protect against this kind of abuse, clients should take precautions to discard RPC messages unless expecting them.

5.6 Attacks on Specific NTP Protocol Fields

Assume an intruder has compromised the authentication scheme and attacks the protocol by intercepting a legitimate server message, modifying one or more protocol header fields and replaying the message. In the following, each protocol header field is considered in turn and the vulnerability of the protocol to modifications in these fields is considered. For this purpose, only those modes involving a client request and server response (client-server and peer-peer modes) are considered. The vulnerability to attacks in multicast mode and anycast modes are considered separately.

As evident from a detailed examination of the protocol, the transmit timestamp field is the only one which can affect the server reply, since the server copies this field intact from the request. The remaining data in the header and timestamps are developed only from server variables unaffected by client requests. An intruder attempting a replay or clogging attack by spoofing client requests to legitimate servers can only result in spurious replies to the client, which will reject the reply on the basis of the previously described sanity checks. Therefore, the only result from replayed or modified requests is to consume processing cycles at the server and client.

The discussion in the following assumes an attack on server replies, in which the intruder modifies certain fields in the expectation this will disrupt the client operations in one way or another. The attacks are of three general classes. In the first class, the modification has the potential to directly affect the client time; in the second, the only affect is to widen the interval considered correct, possibly including more peers that would otherwise be considered falseticker; in the third, the only affect would be for the client to drop the packet; thus, the attack would have no effect other than possible denial of service.

Source Address, Destination Address, Mode

These values are used to identify the NTP association. Each distinct value of the bit string consisting of the concatenation of all of these fields uniquely identifies the association. In principle, it is possible for an intruder to copy these values in a made-up packet, which will cause the packet to be delivered to the protocol machinery of that association. The behavior in this case is described below under algorithmic attacks. Note that, in the present protocol design and reference implementation, the message digest includes only the Mode field, so tampering with the Host Address/Port and/or Peer Address/Port would cause the message to be delivered to some server other than NTP and thus serve no useful purpose to the intruder.

Leap Indicator

These bits define whether the sender expects a leap second to be inserted (01 or deleted (10) at the end of the current day, or whether the sender is unsynchronized (11), or whether normally synchronized (00). The treatment of these bits in the specification and reference implementation represents a plausible vulnerability in the following way. The intruder intercepts a legitimate message, alters the leap bits and sends it on. If the client believes it (i.e., the cryptographic authentication feature is not in use or has been compromised), the client could fail to implement the leap or implement a spurious leap, but only if this occurred shortly before the leap is to take place. As mentioned elsewhere, a careful client will not believe the leap bits, unless the truechimer servers with consistent leap bits are in the majority.

A simple denial of service attack is to intercept the server messages and set both leap bits in the replay. The only reason this would be done, rather than simply to drop the message, is to fool the client into thinking the server source of synchronization is down, but the network itself is operational. Neither of these attacks works on the server, since in client-server modes the server ignores client leap bits.

Version Number

The specification calls for a packet with incorrect version number to be dropped without notice. The reference implementation can accept packets in some older versions in the interest of compatibility. If the intruder knows of a protocol vulnerability with an older version, it could claim to operate with that version and exploit the vulnerability.

Stratum

The specification calls for the client to operated at a stratum level one greater than its server. An intruder could claim to operate at a lower level, which could cause it to be selected for synchronization, in effect, forcing the client to synchronize to its clock, rather than the legitimate server. This is a plausible vulnerability; the only known defense is reliance on the NTP algorithms to detect and suppress falsetickers. Note that the client discards a message if the stratum is greater than its stratum or in any case greater than 15.

Poll Interval

This field is used to control the poll interval in client-server and peer-peer modes. In peer-peer modes, the actual poll interval used by each peer is the minimum of the poll interval field of either peer. In server mode, the association is ephemeral, so that the server responds only when polled. The only attack possible on this field is with peer-peer modes, where an intruder could force either of both peers to poll more rapidly than necessary, thus contribution to the traffic and possible congestion.

Precision, Root Delay, Root Dispersion

These fields are used to calculate the maximum error (synchronization distance) and estimated error (dispersion) used by various filtering and selection algorithms. Adroit manipulation of these fields may accomplish secondary goals, such as admitting other compromised sources from the final set of survivors or denying access for truechimers. An intruder can attempt to gain preferential consideration by artificially lowering the values in these fields, there is nothing to be gained by raising them, except to cause the client to discard the message. In any case, the client discards the message if the values in the root delay or root dispersion exceed one second.

Reference Clock Identifier

This field is used to detect timing loops, where each member of a pair of servers may synchronize to the other. In case of reference clocks, this field identifies the clock type; in other cases, it contains the IP address of the server currently selected by the sender. This allows the receiver to walk the subnet in principle to the root and individually verify the authenticity of that server. The goal of an intruder may be to break that chain or redirect it to a compromised server. The intruder can even create a synchronization loop, causing the flow of synchronization between the clients and servers in the loop to become unstable and to isolate them from legitimate timing sources.

Reference Timestamp

This field establishes the time of the last clock update computed by the sender. If the value is later than the transmit timestamp, or if it is earlier be over 24 hours, the NTP specification requires the message to be ignored. The only damage possible by tampering with this field is to cause the client to disregard the message.

Originate Timestamp (T1)

This field establishes the time the last message from the receiver was sent. It is used to foil man-in-the-middle attacks as follows. The client keeps a copy of its latest transmit timestamp, which appears as a nonce, since with proper blinding it is not practical to predict it. The server copies the transmit timestamp in the client request to the originate timestamp in the server reply. The client checks that the originate timestamp agrees with its saved value and discards the reply if the two values disagree. If this field or the receive timestamp field below is zero, presumably the server has never seen a client message, which could happen only if the sender is in symmetric-active mode. In this case, the client records the transmit timestamp field for use in the next message sent to the server and discards the message.

Receive Timestamp (T2)

This field establishes the time the last client request message was received at the server. See the above originate timestamp field for the client behavior if the receive timestamp field is zero. The client is vulnerable to a man-in-the-middle attack on this field, as described elsewhere.

Transmit Timestamp (T3)

This field establishes the time the last message was sent by the sender. The client remembers the value of the transmit timestamp of the most recent prior message. If a later message carries the same transmit timestamp, the message must be a duplicate and is discarded. The client is vulnerable to a man-in-the-middle attack on this field, as described elsewhere.

Key Identifier

This field contains the key identifier and algorithm as defined by the cryptographic authentication scheme. The vulnerability of this field is discussed under a separate heading.

Cryptographic Checksum

This field contains the cryptographic message digest computed on the NTP header of the current message. The vulnerability of this field is discussed under a separate heading.

6. Hardware and Software Fault Vulnerabilities

NTP in and of itself is a protocol specification which describes the behavior of a set of cooperating state machines, together with a set of algorithms that refine the data and discipline the local clock. The algorithms themselves are vulnerable to faults in one form or another, due for example to out of tolerance components (e.g., clock oscillators), software implementation errors, and miscellaneous problems due to such things as idiosyncratic Unix kernels. The following sections discuss several of these vulnerabilities.

6.1 Vulnerabilities to Broken Hardware

It will come as no surprise to learn that improperly operating hardware can result in faulty clock synchronization. To the extent that some kinds of improper operation can be discovered in the NTP protocol operations themselves, NTP can become a useful system diagnostic aid in its own right. This section considers the consequences of various kinds of hardware failure in the synchronization functions.

There are two main causes of broken hardware: the clock oscillator operating outside of expected frequency tolerance and violation of the strictly monotonic requirement (clock "running backwards"). It is a well known fact that some systems (Sun) commonly exhibit very large frequency offsets, while others (Digital) are much better. Generally, the wide variations are due to the clock hardware specification in each case, but some are due to kernel bugs. For instance, due to a coding error, Sun 4.1.1 systems incorrectly calculated the number of counter cycles between timer interrupts, leading to a systematic frequency error of 100 parts-per-million (PPM) in addition to the error due to normal frequency tolerance. The problem was fixed in the 4.1.3 release; but, apparently has returned in later releases.

Among the causes of frequency error are normal ambient temperature variations. The coefficient of frequency change for ordinary room-temperature crystal oscillators is in the order of 1 PPM/degree C. For all but truly exceptional cases, possibly including the case when a laptop is taken from an overheated room and dropped in a snowdrift, the NTP algorithms can handle these variations. The frequency tolerance engineered in the NTP reference implementation is currently hard-limited at 500 PPM, which is enough to handle the vast majority of cases in the field. The clock discipline algorithm used in NTP will not lock on a clock operating outside of this limit and exhibit a typical sawtooth time error readily apparent to management functions.

A clock counter can sometimes fail in such a way as to appear to run backward or deliver inconsistent readings. It can also happen in multiple processor systems where each processor has an integral clock and the selection of which processor is used to derive system timing is changed. To protect against errors due to these causes, the precision time kernel code described in [13] includes a carefully designed routine that saves each reading and uses that to insure a strict monotonic progression of clock readings, as described below.

6.2 Vulnerabilities to Interrupt Latencies

A much more common problem is latencies of one kind or another, which can cause large jitter spikes and sometimes large timing errors. Such errors can occur due to one or more of the following causes:

There are several places in the NTP reference implementation which requires reading the system clock when a hardware interrupt occurs. It is always desirable to capture a timestamp as close in time to the interrupt as possible; however, in principle, constant latencies can be calibrated out. The real problem is avoiding excessive latency jitter, which can have several causes. First, there is jitter as the various hardware devices contend for the interrupt service routines. The Unix kernel often queues hardware interrupts immediately upon clearing the hardware interrupt flag and saving the raw data and status, if any. It then queues the remainder of interrupt-level processing according to process priority. If as the result of interrupt processing a process is scheduled, additional latency and latency jitter can accrue as the result of priority scheduling.

Most Unix kernels are configured to capture a timestamp immediately upon device interrupt from a network adapter, which can be very effective in hiding the latency and latency jitter of subsequent processing. When a reference clock is configured, latencies can occur in the serial device driver and associated buffer management and protocol routines. For example, as seen by the NTP daemon itself, the latency of a serial port interrupt on a Sun IPC is in the order of several milliseconds. However, in most cases, the latency variations are minor and can be neglected relative to other factors in the latency budget. There are various means in the reference NTP implementation to reduce the effects of these latency variations.

On output to the network, additional latencies due to queueing delay, output blocking and network contention can occur. While it is possible to suppress the effects of these latencies by capturing a timestamp at the output interrupt and passing it back to the daemon for inclusion in a following message, this would result in considerable complexity and intrusion in stock Unix kernels; therefore, this has not been attempted in the reference implementation.

When encrypted authentication is in use, there is an additional contribution to the latency jitter budget due to the execution time of the encryption routine. For DES, this delay is machine dependent, varying from almost a millisecond for a VAX/780 to ten microseconds for a HP 9000/735 (The MD5 delay is approximately three times these figures). However, in the case of both DES and MD5, the running time of the algorithm is independent of the plaintext or key values, so can be considered constant. The NTP reference implementation measures these delays for each machine as part of the initialization process and advances the transmit timestamp to compensate for the delay. On rare occasion, possibly do to interruption by a higher priority process during execution of the encryption routine, the measurement may be in error, resulting in an incorrect measurement. In order to reduce the effect of such an event, the implementation averages each new measurement and the existing measurement with a time constant of one-half.

While the timer interrupt routine runs at a relatively high priority and is unlikely to be itself interrupted by a routine at higher priority, there is one significant exception, the serial device driver. When the PPS signal is connected by a modem control lead interrupt, as is supported in the precision clock kernel modifications, it can happen that the PPS signal interrupt can occur during the execution of the timer interrupt routine. If this occurs before the kernel clock variable has been updated, the result is an incorrect timestamp for the PPS transition in error by one tick. The kernel PPS signal processing includes a three-stage median filter designed to reject such samples.

6.3 Vulnerabilities in the Clock Discipline Process

Most Unix workstations include a time-of-day clock chip with an internal oscillator which runs continuously, even during periods when the machine power is off. Once initialized by reliable means, presumably with eyeball-and-wristwatch, this serves as a sanity check on all timekeeping operations. In addition, Unix file systems include a timestamp on every file, which serves as an additional sanity check. Since it is highly unlikely that a whole year could go by without creating or modifying a file in the filesystem, the timestamp of the most recent file establishes the current year and the clock chip the current time of year to within a few minutes.

In the NTP model, the protocol can adjust the system clock only if the difference between the clock chip time and the time provided by NTP is less than 1000 seconds. It is unlikely that a properly functioning clock chip will accumulate that amount of error during the operating lifetime of the machine; if it does, the NTP protocol will declare the machine unsynchronized and require an operator to take remedial action. It is of course possible for the operating system to periodically reset the clock chip to the current time of day, as provided by NTP, on the assumption that the NTP sources are correct and properly authenticated. This is not recommended, as the chip in and of itself represents the sanity check of last resort.

The unmodified Unix kernel provides two means to set the system clock, the adjtime() system call, which introduces a programmed time offset by running the system logical clock at a fixed frequency either above or below the normal frequency for a period long enough to implement the offset, and the settimeofday() system call, which simply resets the system clock to a value given as the argument. In the case of the modified Unix kernel, the adjtime() function is implemented by the ntp_adjtime() system call, which functions as a hybrid phase/frequency-lock loop as described elsewhere. The settimeofday() function remains unchanged.

In the NTP model, calls to advance or retard the system clock are guaranteed not to exceed 128 ms. Offsets greater than this are implemented using the Unix settimeofday() system call. However, offsets greater than 128 ms are implemented only if no sample in the last fifteen minutes is within the 128-ms window. For example, consider the case with a primary server and connected radio clock. The server has been advised a leap second will occur and steps the system time by one second at the predicted epoch. However, the radio clock itself is unaware of this and sails through the leap without implementing the step. Some time later, the radio resynchronizes to the correct time. During the resynchronization interval, it delivers presumably incorrect time; however, the above algorithm will discard these readings, unless they persist for longer than 15 minutes, which is considered unlikely.

In order to avoid errors due to malfunctioning software, the kernel implementation requires a strictly monotonic adjustment process. This is insured if the adjustment incorporated at each hardware clock interrupt is less than the tick increment. In the case of the adjtime() system call, the amount added at each interrupt, tickadj, must be less than the interrupt increment, called tick. To allow compensation for system clock intrinsic frequency errors as much as 500 PPM, as found on some machines, the value of the tick increment, tickadj, must be at least 5 for 100-Hz clocks and not more than 1 for 1024-Hz clocks. (Values greater than these can cause roundoff errors in some kernel implementations and should be avoided.)

In the case of modified kernels, the kernel implementation includes explicit limit checks on both time and frequency offsets. The time offset argument is clamped at 128 ms and the frequency offset clamped at 500 PPM. These values are assumed architecture constants and not considered vulnerable in the NTP security model. However, the maximum frequency offset produced by a clock update may exceed this limit if the discipline loop time constant is reduced too far. This causes no problem other than increasing the time for the frequency to stabilize. During the development process of the modified kernels, the operation of the discipline loop was tested using a purpose-built simulator which exercised the actual code used in a simulated environment, where the external driving signals could be varied throughout the operating range and beyond. This was used to confirm that the design operated as expected and without unexpected overflow or roundoff error.

6.4 On Dealing with the Monotonic Requirement

Most Unix systems with high resolution clocks have a kernel routine called gettimeofday() (or equivalent) which returns the system clock in seconds and microseconds (or nanoseconds). This routine resolves concurrency issues, in case of multiprocessor systems, and insures monotonicity. In this context, monotonicity means that the clock always moves forward, so that every clock reading is strictly greater than the previous reading, even if the clock has not actually advanced since the previous reading. The execution time for the gettimeofday() system call is about 40 microseconds on a Sun IPC and three microseconds on a Sun SPARC-20. Thus, with current generation hardware and kernel software, it is highly unlikely that the clock would fail to increment between readings. However, for this and other reasons, recent kernel implementations have improved the clock resolution from microseconds to nanoseconds.

If the interval between calls to gettimeofday() is less than the interval between increments, it may be necessary to artificially advance the reading, so the monotonicity requirement can be satisfied. This of course means that the readings can diverge from real time, depending on the reading rate. In the extreme case when the average reading rate exceeds the increment rate, the readings could diverge indefinitely.

There are three ways to address this problem: (1) the system specifications are modified to remove the requirement that the clock always advance, so that it may happen (rarely) that two succeeding clock readings may return the same value; (2) the specifications are modified to require that the average reading rate must be less than some fraction of the increment rate, so that successive readings will almost always result in errors less than a defined upper bound; (3) an explicit system wait function is implemented so that the actual time to read the clock is always greater than the resolution. The design most often adopted in current practice is (2). This admits a specific hazard as noted above; however, the vulnerability of an attack by these means is considered negligible.

If the clock adjustment procedures describe above operate correctly, it is not possible for the system clock to stand still or run backwards. However, there is another problem that can occur with high resolution clock hardware. In some designs, such as Sun systems, the oscillator that drives the timer interrupt also drives the high resolution clock counter. In principle and in practice with careful coding and accounting for the interrupt latency, successive clock readings are accurate, continuous and glitch free. However, in some designs, such as DEC multiprocessor Alpha systems, each processor has its own high resolution clock counter, while all processors share a single timer interrupt oscillator. Since the various oscillators are not themselves synchronized to a common timescale, small variations between them may occur. Specifically, near the time of a timer interrupt, the various processor counters may not exactly line up with the system clock increment - some counters may have stepped beyond the tick by a microsecond or two and some may not have reached it yet. The result is an expected jitter of a microsecond or two and the possibility that the clock may appear to step backward.

The solution for this problem is to provide a kernel variable to hold the last reported system clock time. Each time the system clock is read, which includes the high resolution counter, the time is compared with the last reported time. If the current time is less than or equal to the last reported time, an artificial increment is added so that the reported time is at least one microsecond later than the last reported time. The last reported time variable is set to zero before the system enters service.

With this design, unless modified, a specific vulnerability would exist should the system clock ever purposely be set backward, such as after repair or modification. There are two approaches to this problem. In one, the clock is never allowed to run backward, but must be adjusted to run slow relative to actual time, until actual time catches up to system time. For most applications, this would be unacceptable, since, say for a lag adjustment of one minute, and a slew rate of 500 PPM, it would take well over a day during which the clock would be in error over one second relative to actual time.

An alternative approach to this problem is to set the clock backward and fix whatever breaks as a consequence. In general, the need to set the clock backwards occurs only after a relatively long period during which the machine is inoperable, such as during reboot after crash, repair or modification. In such cases the last reported time kernel variable is first initialized at zero and retains that value until updated by the hardware clock chip or external manual or automatic means. The NTP gettimeofday() implementation model includes a state variable that records whether the clock has ever been set and returns this in the ntp_gettimeofday() system call, so applications can be warned accordingly.

The question remains, in the very rare case where the system clock is set backwards after once having been synchronized, how the last reported time kernel variable is adjusted. In principle, the only way the system clock can be set backwards is using the settimeofday() system call, so the adjustment could be done in that code; however, Unix kernels being what they are, it is not safe to rule out other mechanisms that might set the clock. Accordingly, the NTP implementation model requires the gettimeofday() routine to test whether the current clock reading lags the last reported clock reading by more than one second. If so, the last reported time variable is set to the current time and operation continues from there.

6.5 Irregularities in Leap Second Processing

Since 1972 the world's timekeepers have kept the time in International Atomic Time (TAI), which runs at a constant rate, regardless of irregularities in the Earth's rotation. At intervals of about 18 months, it is necessary to insert a leap second in UTC, in order to maintain agreement with TAI. The NTP implementation model expects advance warning of a leap second, so that the system clock can be adjusted independently in every member of the synchronization subnet at the proper instant. However, UTC time is properly reckoned as the number of TAI seconds from 1 January 1972, which changes at every insertion of a leap second. Conceptually, UTC time counts up to the insertion point at the end of the last day of June or December, then counts off an additional second, numbered second 23:59:60 of that day, then begins the next UTC day. However, the NTP timescale is based on the number of TAI seconds since 1 January 1972, which does not include leap seconds.

The result is that the NTP timescale must be morphed to the UTC timescale during or shortly after the leap insertion [11]. To the NTP timekeeper, it appears that the last second of the UTC day is repeated, giving the system clock time to catch up to the actual time. It appears in this case that the clock has been set backward by one second at the instant of leap. However, due to the actions of the gettimeofday() routine described above, the reported time will advance only at the rate implied in the calling pattern; that is, for each clock reading the reported time will increase by one microsecond or nanosecond. Thus, during the morphing interval, which normally does not exceed one second, various implementations conforming to the NTP model may be in error up to one second. The only case where the morphing interval can exceed one second is if the calling rate exceeds the increment rate, which would cause errors even when leap seconds are not a consideration.

The NTP implementation model uses a state machine to implement the leap second function. The machine makes a transition at every seconds overflow of the system clock. At each transition, the leap bits, one for insert, the other for delete, are inspected and the transition conditioned accordingly. By convention, if both the insert and delete bits are set, the system clock is considered unsynchronized and no leap event occurs. Transitions are conditioned by a counter designed to wrap around at the end of every day. Thus, the machine can initiate an add/delete cycle only at the end of the day. This requires the leap bits to be set on the day of the leap and reset on the following day. A violation of this assumption will lead to an either a spurious leap event or a missing one.

7. Summary and Conclusions

The level of intricate analysis in this report confirms that good security design is a tricky business and invites vulnerabilities in surprising places. With particular relevance in NTP, the most tricky business is in the requirement that time synchronization and source authentication must be decoupled and allowed to proceed independently until a sufficient set of authenticated servers are found and their authenticity confirmed. This requirement is likely to cause some disruption in the design and deployment of other protocols that depend on secure, accurate time, such as key agreement and distribution.

A particularly difficult issue is avoidance of deadlocks involving synchronization and certification. The defining example is where one host depends on another for synchronization, while the latter depends on the former for certification. The only way to reliably avoid such deadlocks is for each host to proceed through all steps leading to authenticated synchronization, including fetching of all certificates, then as a final step verifying the key lifetimes relative to the tentative time. Additional details will have to be worked out in the final protocol specification.

The design of the NTP security model is similar to others involving authenticated, ubiquitous services, such as DNS and web servers. Public-key cryptography is ideally suited for these services. However, these schemes require considerable machine resources inappropriate where thousands of clients are involved in simultaneous message exchanges which repeat at intervals as short as one minute. When large numbers of clients are involved in client-server modes, servers cannot maintain specific state for each client separately, so in NTP the cryptographic means must be regenerated at each client request.

In multicast modes, servers ordinarily receive no information from clients, so that the cryptographic means must be periodically regenerated by the server and distributed to the client population. The scheme described in this report provides a way for the clients to verify that the current message is correctly authenticated to the previous in a chain of messages leading to a value cryptographically bound to the authenticated server.

This report includes an extensive review of vulnerabilities in the current and proposed NTP authentication scheme, including attacks on the cryptosystem, various NTP algorithms, and misbehaving hardware and software infrastructure. While the analysis exposes some plausible vulnerabilities, most can be avoided using appropriate access control and the many sanity checks required in the NTP specification and included in the reference implementation. Notwithstanding these measures, the bottom line in the NTP defensive measures, continues to be the crafted algorithms which groom the date from multiple servers, select the truechimers from the falsetickers and maintain harmony with the system hardware (clock chip) and file system.

8. Bibliography and References

1. Aziz, A., T. Markson, H. Prafullchandra. SKIP extensions for IP multicast. Internet Draft, Sun Microsystems, December 1995, 11 pp.

2. Colella, R., R. Callon, E. Gardner, Y. Rekhter. Guidelines for OSI NSAP allocation in the Internet. Network Working Group Report RFC-1629, NIST, May 1994, 52 pp.

3. Cristian, F. A probabilistic approach to distributed clock synchronization. Proc. Ninth IEEE International Conference on Distributed Computing Systems<D> (June 1989), 288-296.

4. Deering, S., R. Hinden. Internet Protocol, Version 6 (IPv6) specification. Network Working Group Report RFC-1883, Xerox and Ipsilon, January 1996, 37 pp.

5. Eastlake, D., 3rd., and C. Kaufman. Domain Name System security extensions. Internet Draft, CyberCash, December 1995, 45 pp.

6. Haller, N. The S/KEY one-time password system. Network Working Group Report RFC-1760. Bellcore, February 1995, 12 pp.

7. Karn, P., and W.A. Simpson. The Photuris session key management protocol. Network Working Group Internet Draft (ipsec-photuris), Qualcomm, November 1995, 66 pp.

8. Maughan, D., M. Schertler. Internet security association and key management protocol (ISAKMP). Internet Draft, National Security Agency, November 1995, 59 pp.

9. Metzger, P., and W. Simpson. IP authentication using keyed MD5. Network Working Group Report RFC-1828, Piermont and Daydreamer, August 1995, 5 pp.

10. Mills, D.L. Internet time synchronization: the Network Time Protocol. IEEE Trans. Communications COM-39, 10<D> (October 1991), 1482-1493.

11. Mills, D.L. On the chronology and metrology of computer network timescales and their application to the Network Time Protocol. ACM Computer Communications Review 21, 5 (October 1991), 8-17.

12. Mills, D.L. Network Time Protocol (Version 3) specification, implementation and analysis. Network Working Group Report RFC-1305, University of Delaware, March 1992, 113 pp.

13. Mills, D.L. Unix kernel modifications for precision time synchronization. Electrical Engineering Department Report 94-10-1, University of Delaware, October 1994, 24 pp.

14. 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.

15. Mills, D.L. Simple Network Time Protocol (SNTP) version 4 for IPv4, IPv6 and OSI. Electrical Engineering Report 96-10-2, University of Delaware, October 1996, 13 ppp.

16. Mills, D.L. Improved algorithms for synchronizing computer network clocks. IEEE/ACM Trans. Networks<D> (June 1995), 245-254.

17. DES modes of operation. FIPS Publication 81, National Bureau of Standards, December 1980.

18. Data encryption standard. FIPS Publication 46-1, National Bureau of Standards, January 1988.

19. PKCS #1: RSA encryption standard, Version 1.5. RSA Laboratories, November 1993.

20. PKCS #3: Diffie-Hellman key-agreement standard, version 1.4. RSA Laboratories, November 1993.

21. Rivest, R. The MD5 message-digest algorithm. Network Working Group Report RFC-1321, MIT and RSA, April 1992, 21 pp.