AUTHENTICATION SCHEME FOR DISTRIBUTED, UBIQUITOUS, REAL-TIME PROTOCOLS[1,2]

David L. Mills
University of Delaware
Newark, DE 19716

ABSTRACT

Cryptographic authentication methodology proposed for use in the Internet require substantial resources when very large client populations are involved. Resource provisioning becomes especially important when time-critical services are involved. In the cast of time- synchronization services, a special case exists, since cryptographic keys must enforce valid lifetimes, but validating key lifetimes requires cryptographic keys. This paper proposes a scheme which minimizes server resources while resolving the apparent circularity.

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 hierarchical time-synchronization subnet and adjust the clock in each participating subnet peer. The protocol uses redundant servers, diverse network paths and crafted algorithms which discard bogus servers and minimize errors due to various causes. It can operate in several modes, including peer-peer, client-server and multicast. In most places of the Internet of today, NTP provides accuracies of 1-20 ms, depending on the characteristics of the synchronization sources and subnet paths.

The NTP architecture model and supporting algorithms are described in [5], the NTP Version 3 protocol specification in RFC-1305 [6], and recent algorithm improvements in [7]. 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 service such as NTP requires some provision to prevent accidental or malicious attacks on the servers and their clients. 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 security model and authentication scheme are designed with these requirements in mind.

In many ways, the NTP requirements are shared by other ubiquitous, distributed applications, such as directory services, web servers and archive repositories. However, an effective design requires it operate efficiently in all modes supported, including peer-peer, client-server and multicast modes. Current IETF key-agreement schemes like Photuris [3], SKIP [1] and ISAKMP [4] could be used with NTP as with other protocols in peer-peer mode, but are unsuitable for client-server mode, where persistent state cannot be maintained by servers for client populations which may number in the thousands, and multicast mode, where clients do not ordinarily send messages to the servers.

While the current NTP security model and authentication scheme have been in use for well 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 architecture 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 distribute keys and manage their lifetimes infeasible. In a truly survivable network, these functions cannot rely on centralized key management; they require a distributed network design with redundant paths and diverse servers.

1. Prepared through collaborative participation in the Advanced Telecommunications/Information Distribution Research Program (ATIRP) sponsored by U.S. Army Research Laboratories Cooperative Agreement DAAL01-96-2-0002 This work is also supported by DARPA Information Technology Office Contract DABT 63-95-C-0046, NSF Grant NCR 93-01002, and U.S. Navy Contract A303 276-93.

2. This paper has been accepted for publication in Advanced Telecommunications/Information Distribution (ATIRP) Conference, College Park MD, January 1997. It should not be cited or redistributed prior to publication.

CURRENT NTP SECURITY MODEL AND AUTHENTICATION SCHEME

The authentication scheme described in the NTP Version 3 specification RFC-1305 is the basis of the current NTP security model. Its goal is to provide universal access to data products of the protocol, while preventing an intruder from modifying a message or manufacturing a fake message which is acceptable to a client. It is not necessary, nor would it be politically expedient, to encrypt the timestamps or otherwise hide the data in NTP messages, since these are public values. Furthermore, 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.

It is important to note that the NTP security model specifically recognizes that authentication service may not be continuously available. The 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. This requires that clients utilize redundant servers and diverse paths for the authentication function, as well as the synchronization function.

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 (stratum 1) servers, which are assumed authenticated by external means. Each server at a given stratum level in the hierarchy individually authenticates its assigned servers at the next lower stratum level. If at least one of them is authenticated, the server synchronizes with it and reports itself as authenticated to its dependent servers at the next higher stratum level. Whether a server is authenticated or not, the client maintains state variables for it, including its time offset relative to the client clock.

As the synchronization subnet, evolves in response to server failures and restarts, prevailing network delay paths, etc., the authentication hierarchy evolves in response. It may happen that protocol operations can proceed normally; but, due to temporary lack of cryptographic key material, for example, individual servers may become isolated from their sources, even if the timekeeping data itself remains valid. If a server ordinarily synchronized via authenticated sources loses contact with all of these sources, it coasts at its current rate for a time specified by the protocol or until all key lifetimes have expired.

A client is usually configured with a number of servers, each identified by source and destination IP addresses and assigned a secret key and key identifier, which is stored in a secure database. The key is used to construct a message digest (one-way hash function) of the contents with either keyed MD5 [10] or DES-CBC [8]. The session key identifier and message digest form a message authentication code (MAC), which is transmitted with the message. A server is usually stateless and does not retain data from one client request to the next. It uses the key identifier in the client message to retrieve the secret key from its own database and construct the MAC 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.

DESIGN ISSUES

In a perfect world with inexhaustible processing time and memory resources, a public-key cryptosystem such as RSA [9] would be a good foundation on which to build the NTP authentication scheme. In a public- key cryptosystem, each server computes a public/private key pair, or a clique of servers is assigned a public/private key pair using a secure secondary channel. The private key is held by the server and never divulged. A necessary property of public-key cryptosystems is that knowledge of the public key and ciphertext does not compromise the private key or plaintext. The user name, address, public key and related values are stored in a database maintained by directory servers.

In order to minimize the vulnerability to attack, public-key cryptography requires every message to be individually signed using the server private key. The same technique can be used to construct a digital signature for a unit of data or a message and later verify the signature. In order to minimize the processing required, the server constructs a digest of the message contents using a one-way hash function such as MD5, then encrypts 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 using the server public key.

Public/private key pairs are normally generated by the server. The public key, together with identification information, is signed by one or more trusted agents functioning as notaries, to construct a certificate, which is then submitted to the directory service. Certificates bind the public key and related values to identification data, such as a digitized photograph, handwritten signature or voiceprint. These data are not necessarily secure; only the server private key is considered secure, but it is never divulged. In order to verify that an information source is authentic and that the source is in fact in possession of the private key, it is necessary to verify all notary signatures on the certificate trail as well.

Constructing the MD5 message digest is a relatively fast operation; for instance, the time to compute a NTP message digest on a Hewlett Packard 9000/735 is 31 us and 263 us on a Sun Microsystems SPARC 1. However, even when the plaintext is a 128-bit MD5 hash, RSA encryption is painfully slow. For instance, the mean time to sign a NTP message ranges form 80 ms on a Digital 266-MHz Alpha to 2.1 s on a Sun SPARC 1; while the mean time to verify the signature ranges from 7.9 ms on the Alpha to 201 ms on the SPARC 1. While the MD5 running times are independent of data and key, the RSA running times are highly variable, depending on the population of one bits in the key and other factors. For example, with random bit strings as keys, the verification time on a SPARC 1 ranges from 198 ms to 273 ms. Variations as large as these would result in unacceptable loss of accuracy in many NTP applications.

Another approach uses some variant of the Station-to-Station (STS) protocol, such as Photuris to compute a shared secret used as a session key. Since the numbers involved can be very large (512 bits is typical), these operations are slow, but need to be computed only when the keys are changed. However, these protocols require persistent state at the servers, thus are not appropriate for use in NTP client-server and multicast modes with large numbers of servers and clients. 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.

A basic rule in all key distribution and management schemes is that cryptographic keys and associated values must have enforceable lifetimes. Valid keys should be replaced from time to time, in order to frustrate potential cryptanalysis. Once destroyed, a key 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 reliable authentication and, which itself requires keys sensitive to time, an interesting circularity results.

When a key with enforceable lifetime is created or used for cryptographic computations, the results of the computations cannot be validated, unless the entity performing the computations has been correctly synchronized to a source which has been authenticated by a valid certificate trail. Thus, a digital signature cannot be generated, unless the server has authentic time. On the other hand, the signature can be verified at any time, but validated only when the client has authentic time.

This raises the issue that NTP must function in scenarios where reliable network timekeeping has not yet been established, or when the certificates have not yet been verified. The most common case occurs when a client is first started and before its clock has been set. In this such cases, the synchronization and authentication functions must operate even before the 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. This design requirement is unique among all other known network services.

The client operations to synchronize the clock and authenticate the servers cannot depend on which of these functions is done first. In the present NTP protocol model, state variables are developed for each remote server separately, including its apparent time offset relative to the local clock. This process takes from one to several packet exchanges, in order to suppress outlyers and establish reliable offsets. While this is going on, the client may be in process of retrieving certificates from directory services and verifying signatures. As this process involves only public values, it can be performed while NTP is collecting data to set the time. Only after reliable server time and authenticated server identification have been achieved can the local clock be set.

There is a subtle problem when considering the design of secure directory services and related transport protocols. Ordinarily, clients of these services assume the various cryptographic keys and certificates have enforceable lifetimes; that is, the services will not themselves use keys or certificates, unless the lifetimes can be enforced. 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 does not matter, since determining the local clock offset and authenticating the server are performed independently. Designers of secure services must be prepared to deliver the data requested, even if unable to securely authenticate it at the moment.

NTP VERSION 4 SECURITY MODEL AND AUTHENTICATION SCHEME

The Version 4 security model and authentication scheme is designed to be backwards compatible with previous versions, except in a few unavoidable cases. The model adds new features that provide for a self-keyed style of operation in conjunction with new directory and certificate retrieval services now in the planning process in the IETF. The new scheme uses cryptographic message digests in the same way as the original scheme. The contents of the NTP header are hashed with keyed MD5 and a 16-octet session key, yielding a 16-octet message digest. The MAC transmitted following the NTP header consists of a four-octet key identifier followed by a 16-octet message digest.

A client authenticates the server by first obtaining the server name, IP address, public key and related certificate media. Obtaining the public values may involve additional network operations, such as traversing the directory tree, 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.

In the new scheme, each server maintains a private random value which is used together with its private key and other 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 is replaced at longer intervals, such as a week, since this requires all clients to independently verify its authenticity using relatively tedious operations. The RSA public-key cryptosystem is used to encrypt and decrypt data in some messages exchanged between the server and its clients. In addition, secure directory services are assumed available from which public keys and certificates can be obtained. 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 paper.

A server generates a public-private key pair using algorithms well-known in the art. It then generates a certificate binding the public key to identification values and sends it to one or more trusted agents for signatures, then sends it to the directory service for public access. A client authenticates a server by sending either its name or address, depending on how it first learned of its existence, and retrieves the public key and related certificate media. It then verifies the public key using the certificates as necessary. This need be done only once, after which the public key can be cached at the client. These operations use standard procedures, so are not discussed further here.

The scheme works differently for peer-peer, client-server and multicast 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 MAC 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.

The scheme adds new key-request and key-response messages to the suite of control messages already defined. The key-request message sent by a client includes a copy of the client public key. The key-response message sent by the server includes the current session key encrypted first by the server private key and then by the public key in the key-request message. Since the only use of the client public-private key pair is to verify and obscure the response, the public key need not be certificated.

The three modes of NTP operation: peer-peer, client-server and multicast present quite different security models. In peer-peer modes, 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 means external to the protocol without impinging on the current NTP protocol specification or reference implementation. This can be done using schemes proposed by the IETF and are not discussed further here, other than to point out the scheme described below for client- server modes can be used as well.

In client-server mode, the server maintains no per-client state between client requests, either for timekeeping data or cryptographic media. 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 or server. The scheme described below, which was originally suggested by Steven Kent of BBN[3], 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 key-request message to the server, which then generates and encrypts the session key as above and returns it in a key- response message to the client. The client decrypts the session key and caches it in its secure database. 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 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 cached session key is used by the client to compute the message digest in the usual way. 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.

In multicast mode, servers send messages at a controlled rate and respond only to key-request messages. A server first calculates a list of 16-octet session keys for later use, as in the S/KEY system [2]. It first computes the session key as in client-server mode and uses this as the first entry in the list. For this purpose, the IP source and destination addresses are the server address and assigned multicast group address, respectively, and the key identifier is a random roll. The low order four octets of this session key are used to generate the next session key and become the key identifier associated with that key. The server uses the same IP addresses and this session key to generate the next session key. Continuing in this way, the server fills the list, which may have from a few to several hundred entries.

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 session key in the key identifier field.

3. Personal communication.

A client authenticates each message relative to the message that immediately preceded 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 sends a key- request message to obtain the current session key.

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 which, although possible, is 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. 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.

SUMMARY AND CONCLUSIONS

With relevance to Army battlefield systems, The level of intricate dependencies in this paper confirms that good authentication scheme design is a tricky business and invites vulnerabilities in surprising places. With particular relevance to network timekeeping, the most significant requirement is that time synchronization and source authentication must be decoupled and allowed to proceed independently until a sufficient set of timely servers are found and their authenticity confirmed.

The authentication schemes described for NTP client-server and multicast modes have direct application to Army tactical networks and command/control networks, where survivability and independence from centralized control is essential.

REFERENCES

1. Aziz, A., T. Markson, H. Prafullchandra. SKIP extensions for IP multicast. Internet Draft, Sun Microsystems, December 1995, 11 pp. Eastlake, D., 3rd., and C. Kaufman. Domain Name System security extensions. Internet Draft, CyberCash, December 1995, 45 pp.

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

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

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

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

6. 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.

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

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

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

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

AUTHENTICATION SCHEME FOR DISTRIBUTED, UBIQUITOUS, REAL-TIME PROTOCOLS[1,2]

David L. Mills
University of Delaware
Newark, DE 19716

ABSTRACT

Cryptographic authentication methodology proposed for use in the Internet require substantial resources when very large client populations are involved. Resource provisioning becomes especially important when time-critical services are involved. In the cast of time- synchronization services, a special case exists, since cryptographic keys must enforce valid lifetimes, but validating key lifetimes requires cryptographic keys. This paper proposes a scheme which minimizes server resources while resolving the apparent circularity.

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 hierarchical time-synchronization subnet and adjust the clock in each participating subnet peer. The protocol uses redundant servers, diverse network paths and crafted algorithms which discard bogus servers and minimize errors due to various causes. It can operate in several modes, including peer-peer, client-server and multicast. In most places of the Internet of today, NTP provides accuracies of 1-20 ms, depending on the characteristics of the synchronization sources and subnet paths.

The NTP architecture model and supporting algorithms are described in [5], the NTP Version 3 protocol specification in RFC-1305 [6], and recent algorithm improvements in [7]. 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 service such as NTP requires some provision to prevent accidental or malicious attacks on the servers and their clients. 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 security model and authentication scheme are designed with these requirements in mind.

In many ways, the NTP requirements are shared by other ubiquitous, distributed applications, such as directory services, web servers and archive repositories. However, an effective design requires it operate efficiently in all modes supported, including peer-peer, client-server and multicast modes. Current IETF key-agreement schemes like Photuris [3], SKIP [1] and ISAKMP [4] could be used with NTP as with other protocols in peer-peer mode, but are unsuitable for client-server mode, where persistent state cannot be maintained by servers for client populations which may number in the thousands, and multicast mode, where clients do not ordinarily send messages to the servers.

While the current NTP security model and authentication scheme have been in use for well 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 architecture 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 distribute keys and manage their lifetimes infeasible. In a truly survivable network, these functions cannot rely on centralized key management; they require a distributed network design with redundant paths and diverse servers.

1. Prepared through collaborative participation in the Advanced Telecommunications/Information Distribution Research Program (ATIRP) sponsored by U.S. Army Research Laboratories Cooperative Agreement DAAL01-96-2-0002 This work is also supported by DARPA Information Technology Office Contract DABT 63-95-C-0046, NSF Grant NCR 93-01002, and U.S. Navy Contract A303 276-93.

2. This paper has been accepted for publication in Advanced Telecommunications/Information Distribution (ATIRP) Conference, College Park MD, January 1997. It should not be cited or redistributed prior to publication.

CURRENT NTP SECURITY MODEL AND AUTHENTICATION SCHEME

The authentication scheme described in the NTP Version 3 specification RFC-1305 is the basis of the current NTP security model. Its goal is to provide universal access to data products of the protocol, while preventing an intruder from modifying a message or manufacturing a fake message which is acceptable to a client. It is not necessary, nor would it be politically expedient, to encrypt the timestamps or otherwise hide the data in NTP messages, since these are public values. Furthermore, 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.

It is important to note that the NTP security model specifically recognizes that authentication service may not be continuously available. The 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. This requires that clients utilize redundant servers and diverse paths for the authentication function, as well as the synchronization function.

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 (stratum 1) servers, which are assumed authenticated by external means. Each server at a given stratum level in the hierarchy individually authenticates its assigned servers at the next lower stratum level. If at least one of them is authenticated, the server synchronizes with it and reports itself as authenticated to its dependent servers at the next higher stratum level. Whether a server is authenticated or not, the client maintains state variables for it, including its time offset relative to the client clock.

As the synchronization subnet, evolves in response to server failures and restarts, prevailing network delay paths, etc., the authentication hierarchy evolves in response. It may happen that protocol operations can proceed normally; but, due to temporary lack of cryptographic key material, for example, individual servers may become isolated from their sources, even if the timekeeping data itself remains valid. If a server ordinarily synchronized via authenticated sources loses contact with all of these sources, it coasts at its current rate for a time specified by the protocol or until all key lifetimes have expired.

A client is usually configured with a number of servers, each identified by source and destination IP addresses and assigned a secret key and key identifier, which is stored in a secure database. The key is used to construct a message digest (one-way hash function) of the contents with either keyed MD5 [10] or DES-CBC [8]. The session key identifier and message digest form a message authentication code (MAC), which is transmitted with the message. A server is usually stateless and does not retain data from one client request to the next. It uses the key identifier in the client message to retrieve the secret key from its own database and construct the MAC 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.

DESIGN ISSUES

In a perfect world with inexhaustible processing time and memory resources, a public-key cryptosystem such as RSA [9] would be a good foundation on which to build the NTP authentication scheme. In a public- key cryptosystem, each server computes a public/private key pair, or a clique of servers is assigned a public/private key pair using a secure secondary channel. The private key is held by the server and never divulged. A necessary property of public-key cryptosystems is that knowledge of the public key and ciphertext does not compromise the private key or plaintext. The user name, address, public key and related values are stored in a database maintained by directory servers.

In order to minimize the vulnerability to attack, public-key cryptography requires every message to be individually signed using the server private key. The same technique can be used to construct a digital signature for a unit of data or a message and later verify the signature. In order to minimize the processing required, the server constructs a digest of the message contents using a one-way hash function such as MD5, then encrypts 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 using the server public key.

Public/private key pairs are normally generated by the server. The public key, together with identification information, is signed by one or more trusted agents functioning as notaries, to construct a certificate, which is then submitted to the directory service. Certificates bind the public key and related values to identification data, such as a digitized photograph, handwritten signature or voiceprint. These data are not necessarily secure; only the server private key is considered secure, but it is never divulged. In order to verify that an information source is authentic and that the source is in fact in possession of the private key, it is necessary to verify all notary signatures on the certificate trail as well.

Constructing the MD5 message digest is a relatively fast operation; for instance, the time to compute a NTP message digest on a Hewlett Packard 9000/735 is 31 us and 263 us on a Sun Microsystems SPARC 1. However, even when the plaintext is a 128-bit MD5 hash, RSA encryption is painfully slow. For instance, the mean time to sign a NTP message ranges form 80 ms on a Digital 266-MHz Alpha to 2.1 s on a Sun SPARC 1; while the mean time to verify the signature ranges from 7.9 ms on the Alpha to 201 ms on the SPARC 1. While the MD5 running times are independent of data and key, the RSA running times are highly variable, depending on the population of one bits in the key and other factors. For example, with random bit strings as keys, the verification time on a SPARC 1 ranges from 198 ms to 273 ms. Variations as large as these would result in unacceptable loss of accuracy in many NTP applications.

Another approach uses some variant of the Station-to-Station (STS) protocol, such as Photuris to compute a shared secret used as a session key. Since the numbers involved can be very large (512 bits is typical), these operations are slow, but need to be computed only when the keys are changed. However, these protocols require persistent state at the servers, thus are not appropriate for use in NTP client-server and multicast modes with large numbers of servers and clients. 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.

A basic rule in all key distribution and management schemes is that cryptographic keys and associated values must have enforceable lifetimes. Valid keys should be replaced from time to time, in order to frustrate potential cryptanalysis. Once destroyed, a key 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 reliable authentication and, which itself requires keys sensitive to time, an interesting circularity results.

When a key with enforceable lifetime is created or used for cryptographic computations, the results of the computations cannot be validated, unless the entity performing the computations has been correctly synchronized to a source which has been authenticated by a valid certificate trail. Thus, a digital signature cannot be generated, unless the server has authentic time. On the other hand, the signature can be verified at any time, but validated only when the client has authentic time.

This raises the issue that NTP must function in scenarios where reliable network timekeeping has not yet been established, or when the certificates have not yet been verified. The most common case occurs when a client is first started and before its clock has been set. In this such cases, the synchronization and authentication functions must operate even before the 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. This design requirement is unique among all other known network services.

The client operations to synchronize the clock and authenticate the servers cannot depend on which of these functions is done first. In the present NTP protocol model, state variables are developed for each remote server separately, including its apparent time offset relative to the local clock. This process takes from one to several packet exchanges, in order to suppress outlyers and establish reliable offsets. While this is going on, the client may be in process of retrieving certificates from directory services and verifying signatures. As this process involves only public values, it can be performed while NTP is collecting data to set the time. Only after reliable server time and authenticated server identification have been achieved can the local clock be set.

There is a subtle problem when considering the design of secure directory services and related transport protocols. Ordinarily, clients of these services assume the various cryptographic keys and certificates have enforceable lifetimes; that is, the services will not themselves use keys or certificates, unless the lifetimes can be enforced. 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 does not matter, since determining the local clock offset and authenticating the server are performed independently. Designers of secure services must be prepared to deliver the data requested, even if unable to securely authenticate it at the moment.

NTP VERSION 4 SECURITY MODEL AND AUTHENTICATION SCHEME

The Version 4 security model and authentication scheme is designed to be backwards compatible with previous versions, except in a few unavoidable cases. The model adds new features that provide for a self-keyed style of operation in conjunction with new directory and certificate retrieval services now in the planning process in the IETF. The new scheme uses cryptographic message digests in the same way as the original scheme. The contents of the NTP header are hashed with keyed MD5 and a 16-octet session key, yielding a 16-octet message digest. The MAC transmitted following the NTP header consists of a four-octet key identifier followed by a 16-octet message digest.

A client authenticates the server by first obtaining the server name, IP address, public key and related certificate media. Obtaining the public values may involve additional network operations, such as traversing the directory tree, 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.

In the new scheme, each server maintains a private random value which is used together with its private key and other 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 is replaced at longer intervals, such as a week, since this requires all clients to independently verify its authenticity using relatively tedious operations. The RSA public-key cryptosystem is used to encrypt and decrypt data in some messages exchanged between the server and its clients. In addition, secure directory services are assumed available from which public keys and certificates can be obtained. 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 paper.

A server generates a public-private key pair using algorithms well-known in the art. It then generates a certificate binding the public key to identification values and sends it to one or more trusted agents for signatures, then sends it to the directory service for public access. A client authenticates a server by sending either its name or address, depending on how it first learned of its existence, and retrieves the public key and related certificate media. It then verifies the public key using the certificates as necessary. This need be done only once, after which the public key can be cached at the client. These operations use standard procedures, so are not discussed further here.

The scheme works differently for peer-peer, client-server and multicast 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 MAC 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.

The scheme adds new key-request and key-response messages to the suite of control messages already defined. The key-request message sent by a client includes a copy of the client public key. The key-response message sent by the server includes the current session key encrypted first by the server private key and then by the public key in the key-request message. Since the only use of the client public-private key pair is to verify and obscure the response, the public key need not be certificated.

The three modes of NTP operation: peer-peer, client-server and multicast present quite different security models. In peer-peer modes, 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 means external to the protocol without impinging on the current NTP protocol specification or reference implementation. This can be done using schemes proposed by the IETF and are not discussed further here, other than to point out the scheme described below for client- server modes can be used as well.

In client-server mode, the server maintains no per-client state between client requests, either for timekeeping data or cryptographic media. 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 or server. The scheme described below, which was originally suggested by Steven Kent of BBN[3], 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 key-request message to the server, which then generates and encrypts the session key as above and returns it in a key- response message to the client. The client decrypts the session key and caches it in its secure database. 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 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 cached session key is used by the client to compute the message digest in the usual way. 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.

In multicast mode, servers send messages at a controlled rate and respond only to key-request messages. A server first calculates a list of 16-octet session keys for later use, as in the S/KEY system [2]. It first computes the session key as in client-server mode and uses this as the first entry in the list. For this purpose, the IP source and destination addresses are the server address and assigned multicast group address, respectively, and the key identifier is a random roll. The low order four octets of this session key are used to generate the next session key and become the key identifier associated with that key. The server uses the same IP addresses and this session key to generate the next session key. Continuing in this way, the server fills the list, which may have from a few to several hundred entries.

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 session key in the key identifier field.

3. Personal communication.

A client authenticates each message relative to the message that immediately preceded 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 sends a key- request message to obtain the current session key.

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 which, although possible, is 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. 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.

SUMMARY AND CONCLUSIONS

With relevance to Army battlefield systems, The level of intricate dependencies in this paper confirms that good authentication scheme design is a tricky business and invites vulnerabilities in surprising places. With particular relevance to network timekeeping, the most significant requirement is that time synchronization and source authentication must be decoupled and allowed to proceed independently until a sufficient set of timely servers are found and their authenticity confirmed.

The authentication schemes described for NTP client-server and multicast modes have direct application to Army tactical networks and command/control networks, where survivability and independence from centralized control is essential.

REFERENCES

1. Aziz, A., T. Markson, H. Prafullchandra. SKIP extensions for IP multicast. Internet Draft, Sun Microsystems, December 1995, 11 pp. Eastlake, D., 3rd., and C. Kaufman. Domain Name System security extensions. Internet Draft, CyberCash, December 1995, 45 pp.

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

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

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

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

6. 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.

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

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

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

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