NTP Security Protocol

David L. Mills
University of Delaware
http://www.eecis.udel.edu/~mills
mailto:mills@udel.edu

Sir John Tenniel; *Alice's Adventures in Wonderland,*Lewis Carroll

Security protocol requirements

- It must interoperate with the existing NTP architecture model and protocol design. In particular, it must support the symmetric key scheme described in RFC-1305.
- It must provide for the independent collection of cryptographic values and time values. A NTP packet is accepted for processing only when the required cryptographic values have been obtained and verified and the NTP header has passed all sanity checks.
- It must not significantly degrade the potential accuracy of the NTP synchronization algorithms. In particular, it must not make unreasonable demands on the network or host processor and memory resources.
- It must be resistant to cryptographic attacks, specifically those identified in the security model above. In particular, it must be tolerant of operational or implementation variances, such as packet loss or misorder, or suboptimal configurations.

Security protocol requirements (continued)

- It must build on a widely available suite of cryptographic algorithms, yet be independent of the particular choice. In particular, it must not require data encryption other than incidental to signature and cookie encryption operations.
- It must function in all the modes supported by NTP, including client/server, symmetric and broadcast modes.
- It must not require intricate per-client or per-server configuration other than the availability of the required cryptographic keys and certificates.
- The reference implementation must contain provisions to generate cryptographic key files specific to each client and server.

Autokey security protocol

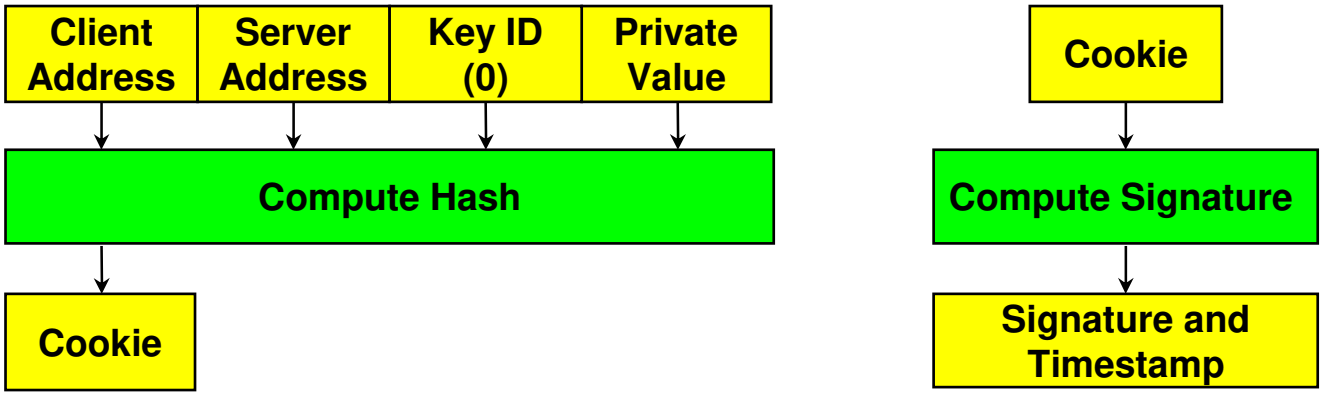
- NTP and Autokey protocols work independently for each client, with tentative outcomes confirmed only after both succeed.
- Public keys and certificates are obtained and verified relatively infrequently using X.509 certificates and certificate trails.
- Session keys are derived from public keys using fast algorithms.
- Each NTP message is individually authenticated using session key and message digest (keyed MD5).
- A proventic trail is a sequence of NTP servers each synchronized and cryptographically verified to the next lower stratum server and ending on one or more trusted servers.
- Proventic trails are constructed from each server to the trusted hosts at decreasing stratum levels.
- When server time and at least one proventic trail are verified, the host is admitted to the population used to synchronize the system clock.

Session keys

NTPv4 Session Key

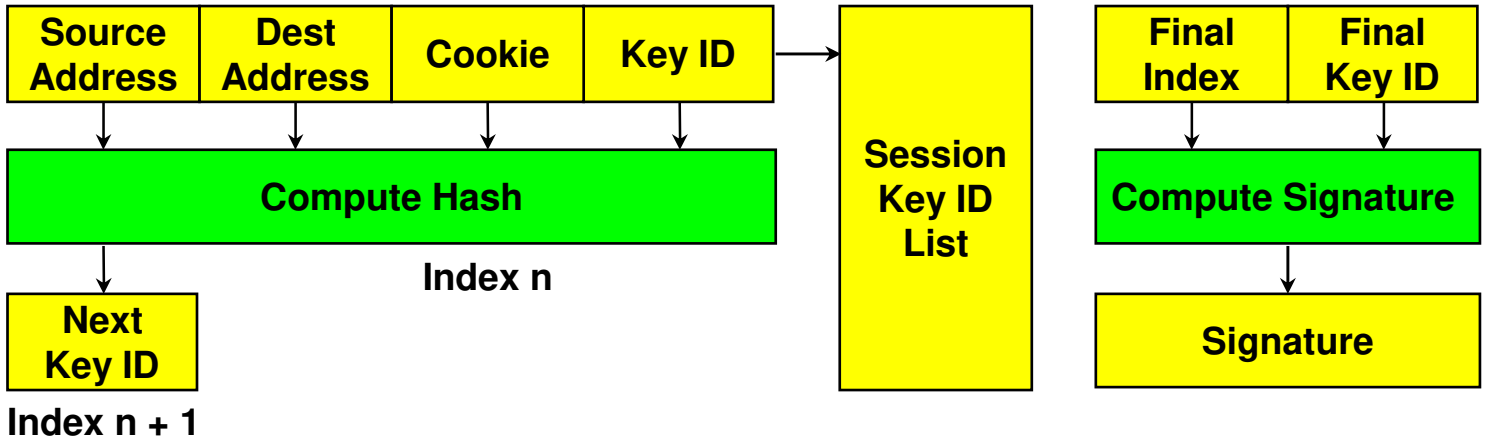
- NTPv4 session keys have four 32-bit words (16 octets total).
- The session key value is the 16-octet MD5 message digest of the session key.
- Key IDs have pseudo-random values and are used only once. A special key ID value of zero is used as a crypto-NAK reply.
- In broadcast modes and in any message including an extension field, the cookie has a public value (zero). These messages are always signed.
- In client/server modes the cookie is a hash of the addresses and a private value.
- In symmetric modes the cookie is a random roll; in case both peers generate cookies, the agreed cookie is the EXOR of the two values.

Computing the cookie



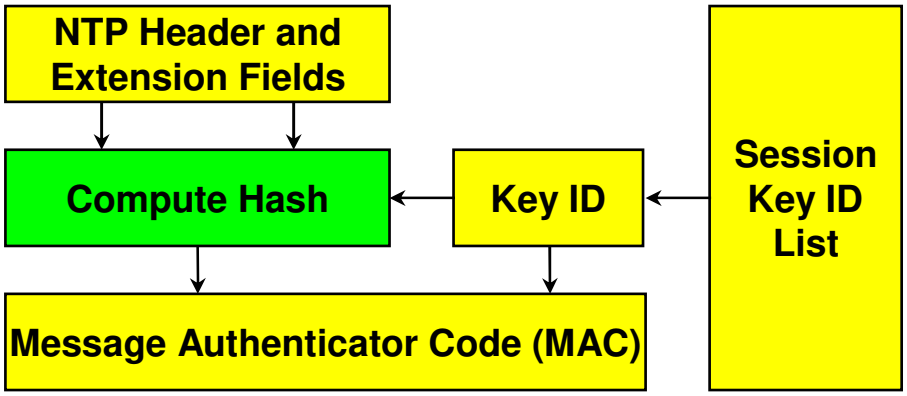
- The server generates a cookie unique to the client and server addresses and its own private value. It returns the cookie, signature and timestamp to the client in an extension field.
- The cookie is transmitted from server to client encrypted by the clien public key.
- The server uses the cookie to validate requests and construct replies.
- The client uses the cookie to validate the reply and checks that the request key ID matches the reply key ID.

Generating the session key list



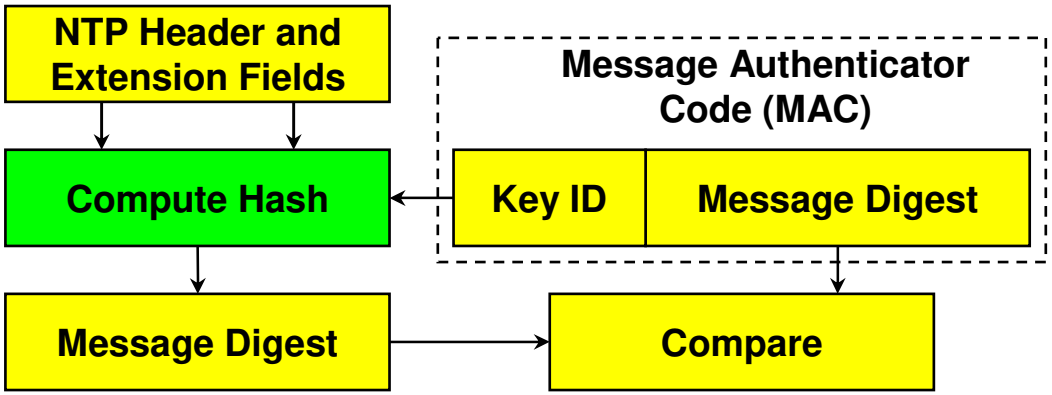
- The server rolls a random 32-bit seed as the initial key ID and selects the cookie. Messages with a zero cookie contain only public values.
- The initial session key is constructed using the given addresses, cookie and initial key ID. The session key value is stored in the key cache.
- The next session key is constructed using the first four octets of the session key value as the new key ID. The server continues to generate the full list.
- The final index number and last key ID are provided in an extension field with signature and timestamp.

Sending messages



- The message authenticator code (MAC) consists of the MD5 message digest of the NTP header and extension fields using the session key ID and value stored in the key cache.
- The server uses the session key ID list in reverse order and discards each key value after use.
- An extension field containing the last index number and key ID is included in the first packet transmitted (last on the list).
- This extension field can be provided upon request at any time.
- When all entries in the key list are used, a new one is generated.

Receiving messages



- The intent is not to hide the message contents, just verify where it came from and that it has not been modified in transit.
- The MAC message digest is compared with the computed digest of the NTP header and extension fields using the session key ID in the MAC and the key value computed from the addresses, key ID and cookie.
- If the cookie is zero, the message contains public values. Anybody can validate the message or make a valid message containing any values.
- If the cookie has been determined by secret means, nobody except the parties to the secret can validate a message or make a valid message.

NTP protocol header and timestamp formats

NTP Protocol Header Format (32 bits)

LI	VN	Mode	Strat	Poll	Prec
Root Delay					
Root Dispersion					
Reference Identifier					
Reference Timestamp (64)					
Originate Timestamp (64)					
Receive Timestamp (64)					
Transmit Timestamp (64)					
Extension Field 1 (optional)					
Extension Field 2… (optional)					
Key/Algorithm Identifier					
Message Digest (128)					

Cryptosum

Authenticator (Optional)

LI	leap warning indicator
VN	version number (4)
Strat	stratum (0-15)
Poll	poll interval (log2)
Prec	precision (log2)

NTP Timestamp Format (64 bits)

Seconds (32)	Fraction (32)

Value is in seconds and fraction since 0^h 1 January 1900

NTP v4 Extension Field

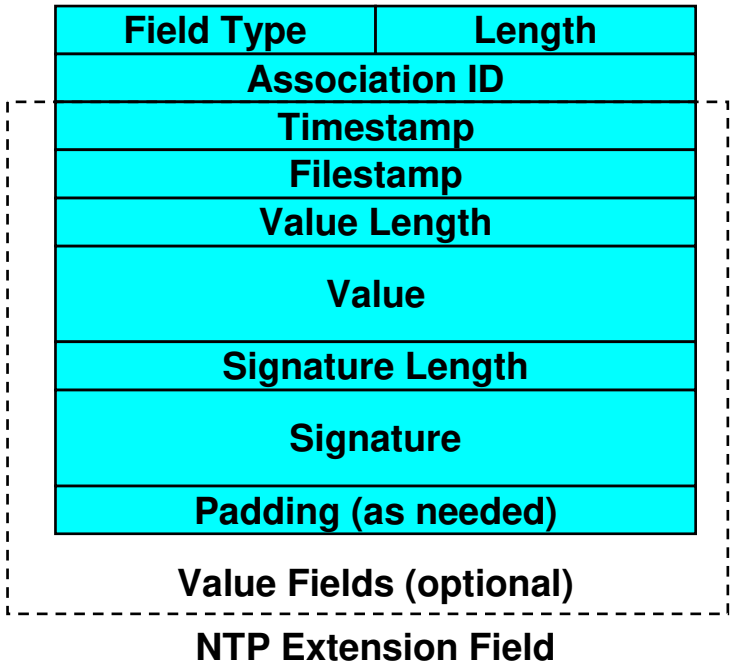
Field Type	Length
Extension Field (padded to 32-bit boundary)	

Last field padded to 64-bit boundary

NTP v3 and v4
NTP v4 only
authentication only

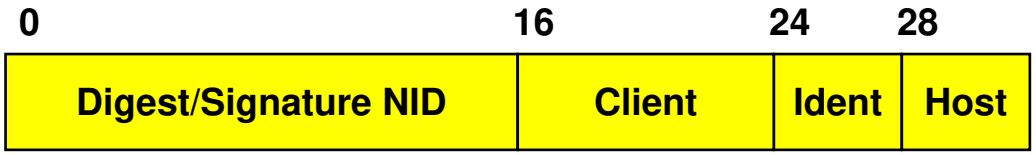
Authenticator uses MD5 cryptosum of NTP header plus extension fields (NTPv4)

NTPv4 extension fields



- New extension fields format defined for NTP Version 4
  - Fields are processed in order.
  - Requests may be transmitted with or without value fields.
  - Last field padded to 64-bit boundary; all others padded to 32-bit boundary.
  - Field length covers all payload and padding.

Host status word

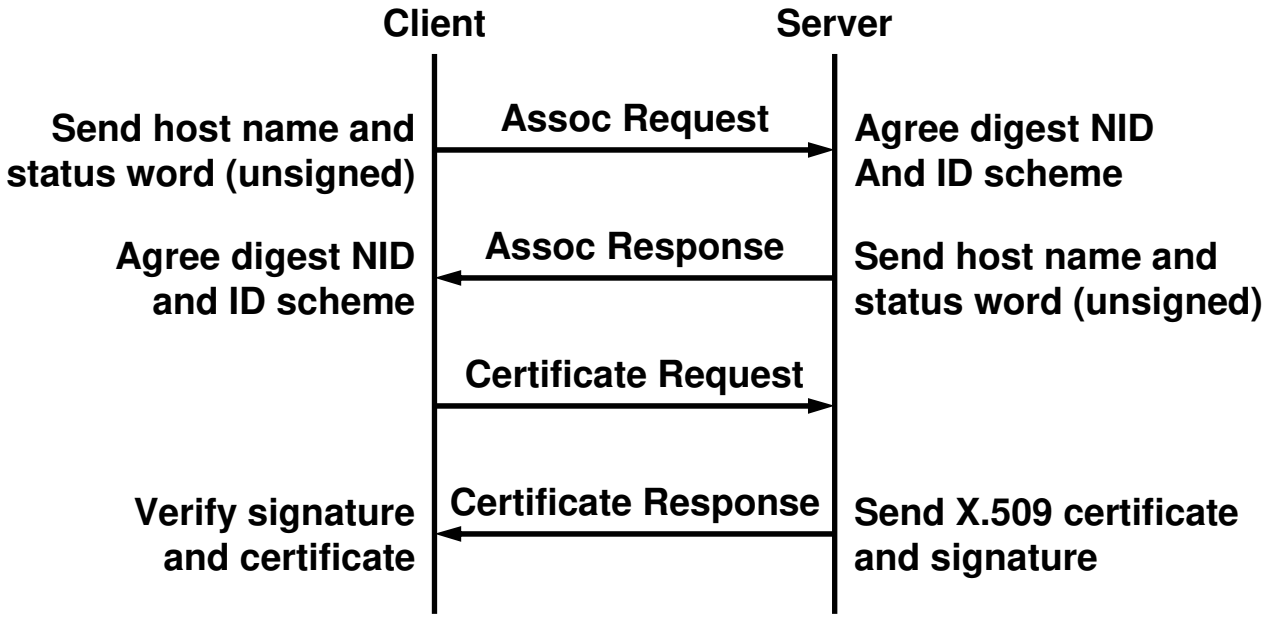


- Host status word is constructed at initialization time.
  - Client and server exchange status words with offered identity schemes
  - Both client and server agree on the same scheme
- Digest/Signature NID
  - 16 bits for Network ID of the message digest/signature encryption scheme
- Client
  - 8 bits available for client Autokey protocol operations
- Host
  - 8 bits available for host Autokey operations and offered identity scheme

Autokey protocol exchanges

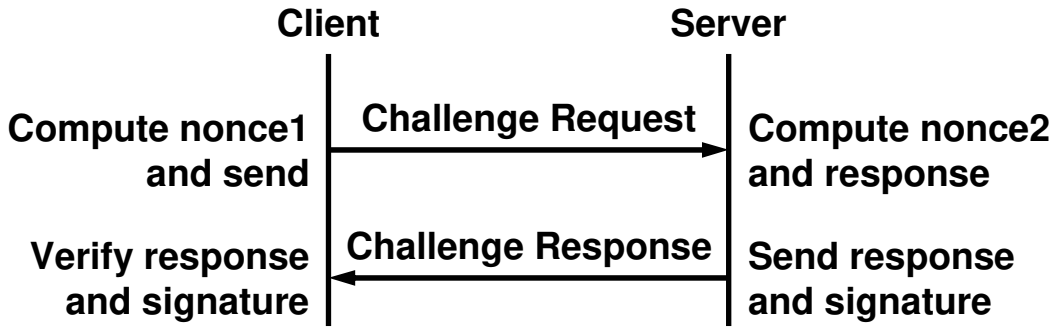
- Parameter Exchange (ASSOC message - not signed)
  - Exchange host names; agree on digest/signature and identity schemes. Optional: verify host name/address using reverse-DNS.
- Certificate Exchange (CERT message)
  - Obtain and verify certificates on the trail to a trusted root certificate.
- Identity Exchange (IFF, GQ and MV messages)
  - Verify server identity using agreed identity scheme (TC, IFF, GQ, MV).
- Values Exchange (COOKIE and AUTO messages)
  - Obtain and verify the cookie, autokey values and leapseconds table, depending on the association mode (client-server, broadcast, symmetric).
- Signature Exchange (SIGN message)
  - Request the server to sign and return a client certificate. The exchange is valid only when the client has synchronized to a proventic source and the server identity has been confirmed.

Parameter and certificate exchanges



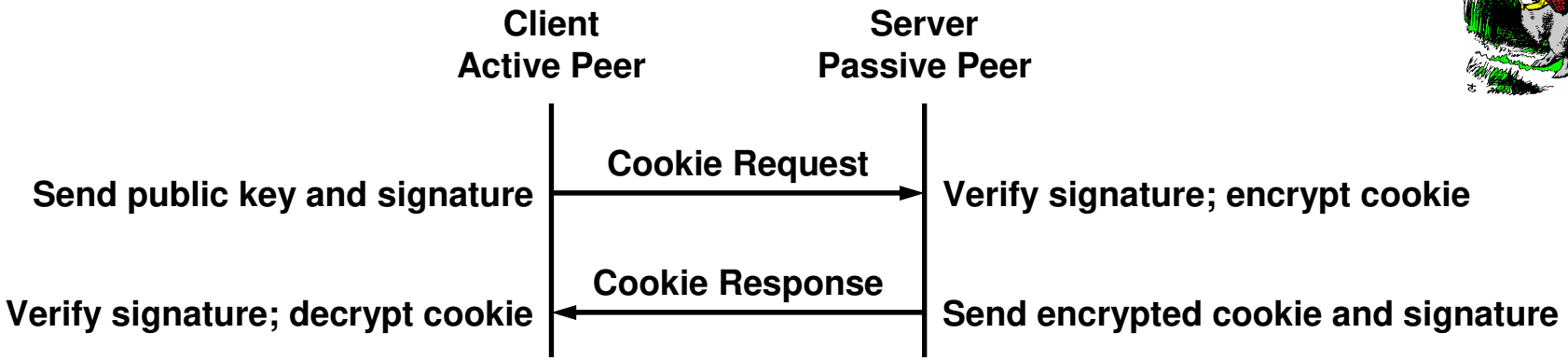
- Initial exchange of host status words defines server message digest and signature encryption algorithm and identity scheme.
- The Certificate Request/Response cycle repeats as needed.
  - Primary (stratum 1) certificate is explicitly trusted and self-signed.
  - Secondary certificates are signed by the next lower stratum server and validated with its public key.

Identification exchange



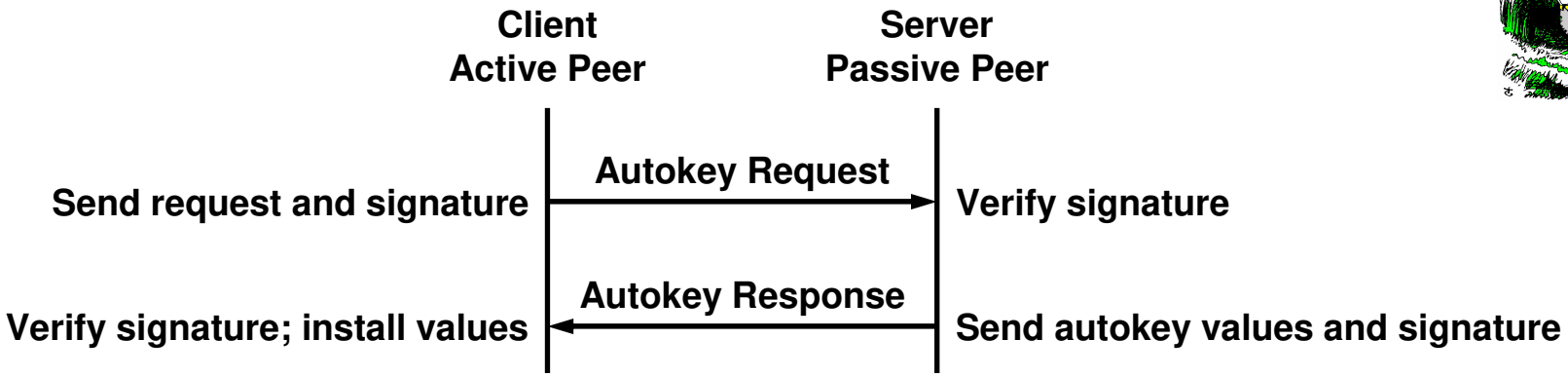
- This is a challenge-response scheme
  - Client Alice and server Bob share a common set of parameters and a private group key *b*.
  - Alice rolls random nonce *r* and sends to Bob.
  - Bob rolls random nonce *k*, computes a one-way function *f*(*r*, *k*, *b*) and sends to Alice.
  - Alice computes some function *g*(*f*, *b*) to verify that Bob knows *b*.
- The signature prevents message modification and binds the response to Bob's private key.
- An interceptor can see the challenge and response, but cannot determine *k* or *b* or how to construct a response acceptable to Alice.

Cookie exchange



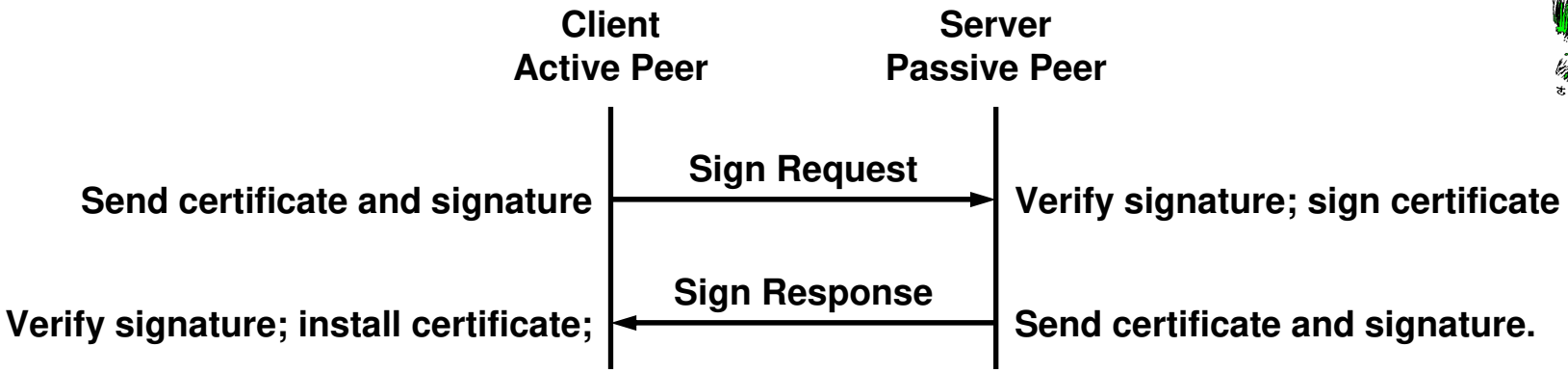
- Client sends public key to server without signature when not synchronized.
- Symmetric active peer sends public key and signature to passive peer when synchronized.
- Server cookie is encrypted from the hash of source/destination addresses, zero key ID and server private value.
- Symmetric passive cookie is a random value for every exchange.
- Server private value is refreshed and protocol restarted once per day.

Autokey exchange



- Server generates key list and signature calculated to last about one hour.
- Client sends request to server without signature when not synchronized.
- Server replies with the last index number and key ID on the list.
- Broadcast server uses AUTO response for the first message after regenerating the key and ASSOC response for all other messages.

Sign certificate exchange

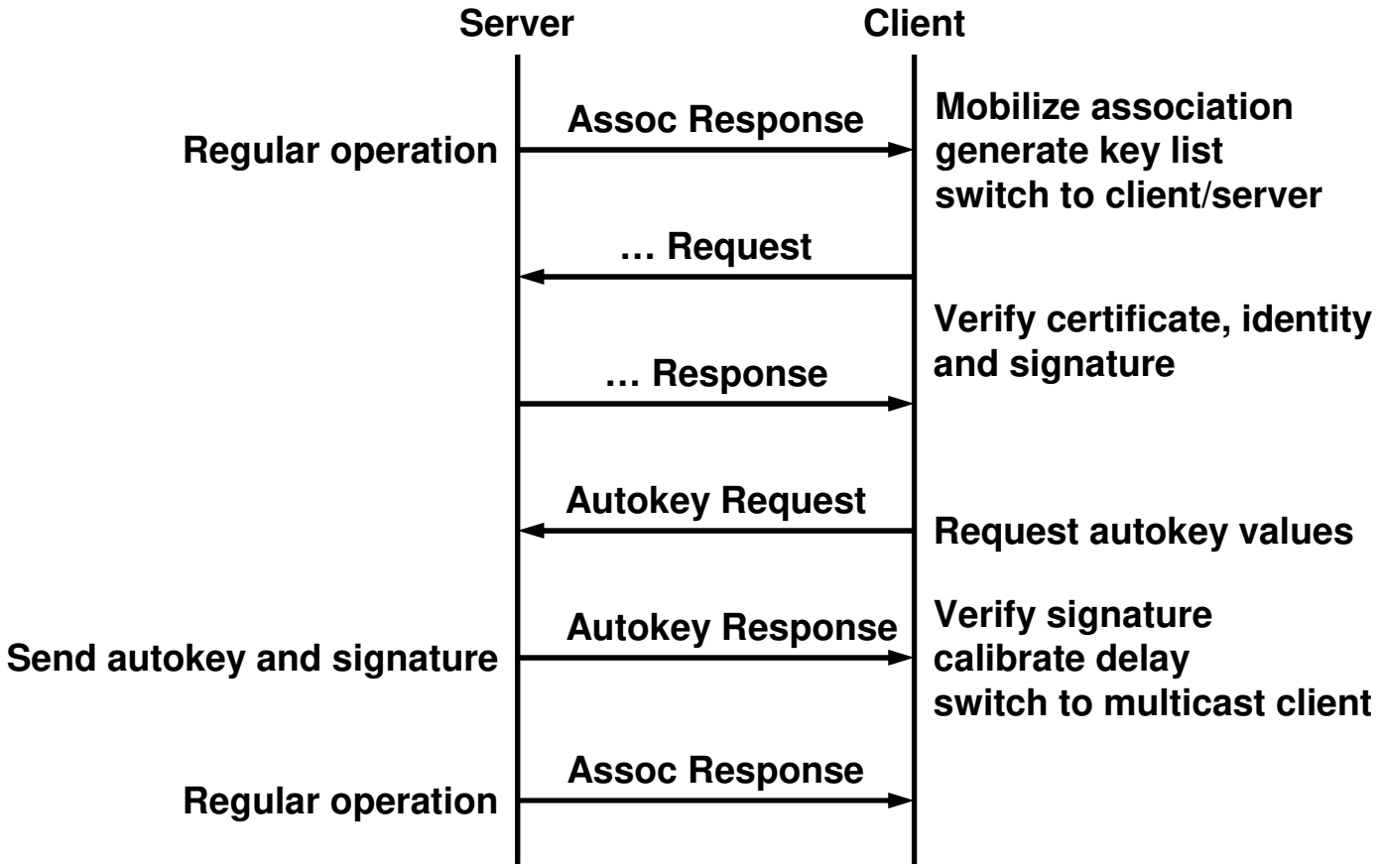


- This is used to authenticate client to server, with server acting as de facto certificate authority using encrypted credential scheme TBD.
- Client sends certificate to server with or without signature.
- Server extracts request data and signs with server private key.
- Client verifies certificate and signature.
- Subsequently, client supplies this certificate rather than self-signed certificate, so clients can verify with server public key.

Broadcast/multicast mode

- The broadcast server sends messages at fixed intervals.
  - The first message sent after regenerating the key list includes the autokey values and signature.
  - Other messages include the server association ID, but no signature. This is used as a handle for clients to request the autokey values if necessary.
  - These messages are considered public values, so the cookie value is zero.
- When a multicast client receives the first message, it temporarily switches to client/server mode in order to calibrate the network propagation delay and authenticate the server.
- The client first obtains the parameters and verifies the certificate, identity and signature as in client/server mode, then obtains the autokey values and signature.
- When the propagation delay is calibrated, the client switches back to broadcast client mode and makes no further transmissions.

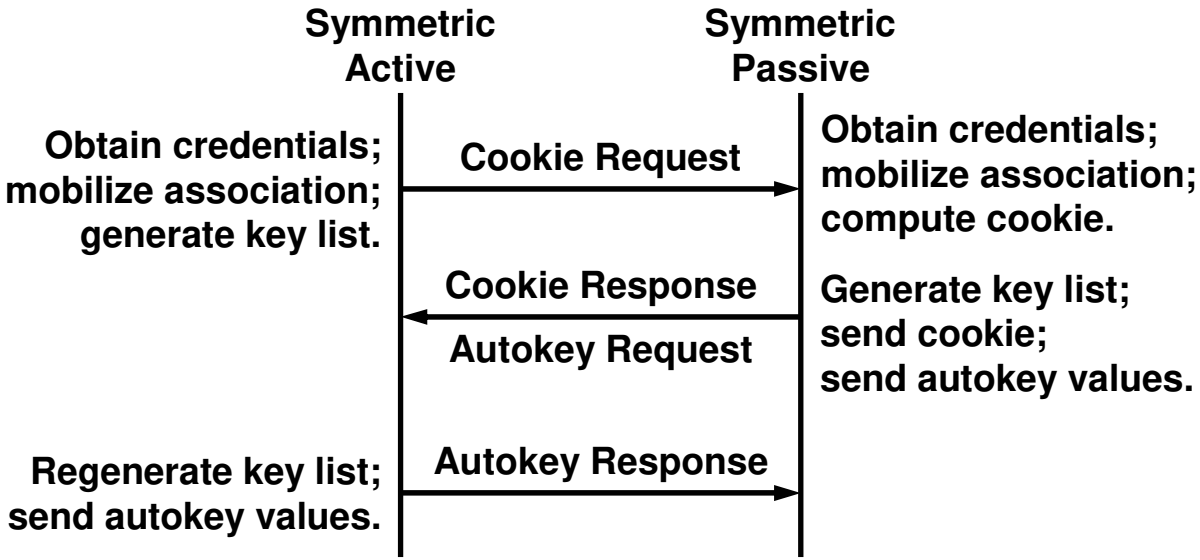
Broadcast/multicast mode protocol



Symmetric modes

- Symmetric peers can each synchronize the other, depending on which one has the lowest synchronization distance.
  - One of the peers must be active; the other can be active or passive. Each peer computes a cookie and generates key lists independently.
  - The passive peer is presumably already synchronized to a proventic source. It mobilizes an association upon arrival of the first message from the active peer and begins a parameter exchange.
  - The active peer proceeds through the various exchanges until synchronized to the passive peer. The passive peer continues the parameter exchange.
  - When the active peer is synchronized, the passive peer proceeds through the various exchanges until synchronized to the active peer.
  - When the passive peer is synchronized, both peers continue using the key lists as necessary. An AUTO response with

Symmetric modes protocol



TAI leapsecond table

- The UTC leapsecond table contains the historic epoches, in NTP seconds, of leapsecond insertions since UTC began in 1972
  - An authoritative copy is on NIST NTP servers in `pub/leap-seconds`
  - It can be retrieved directly from NIST using FTP
  - It can be retrieved from a server or peer during the Autokey dance
  - If both peers have the table, only the most recent is used
  - NTP provides the seconds offset relative to TAI to the kernel
- Application program interface
  - The ntp_gettime() system call returns the current time and seconds offset relative to TAI
  - Currently, only FreeBSD, Linux and locally modified SunOS and Tru64 (Alpha) have modified kernels to support this interface

Further information

- Network Time Protocol (NTP): http://www.ntp.org/
  - Current NTP Version 3 and 4 software and documentation
  - FAQ and links to other sources and interesting places
- David L. Mills: http://www.eecis.udel.edu/~mills
  - Papers, reports and memoranda in PostScript and PDF formats
  - Briefings in HTML, PostScript, PowerPoint and PDF formats
  - Collaboration resources hardware, software and documentation
  - Songs, photo galleries and after-dinner speech scripts
- FTP server ftp.udel.edu (`pub/ntp` directory)
  - Current NTP Version 3 and 4 software and documentation repository
  - Collaboration resources repository
- Related project descriptions and briefings
  - See “Current Research Project Descriptions and Briefings” at http://www.eecis.udel.edu/~mills/status.htm