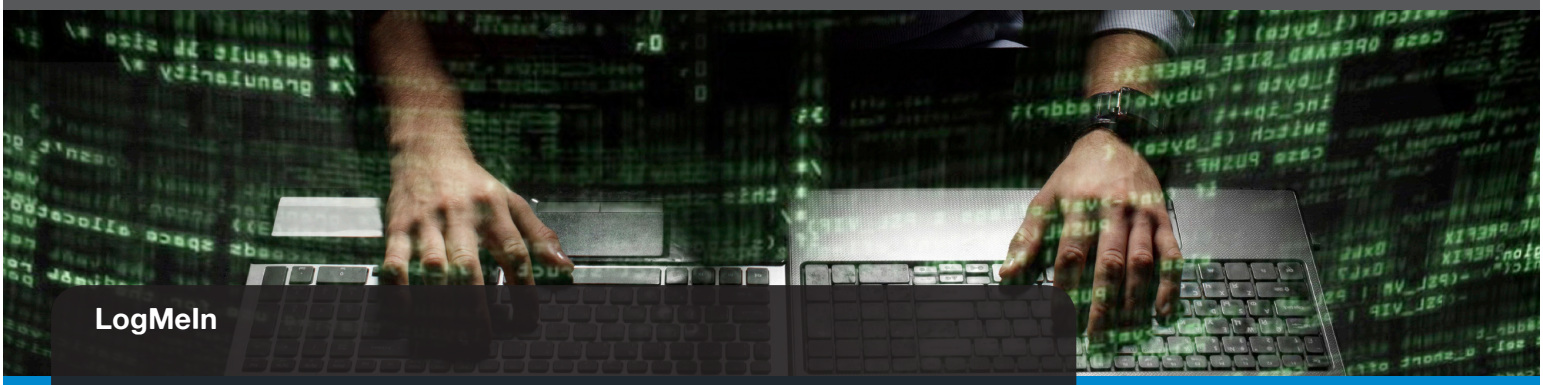


LogMeIn

Hamachi Security

An Overview

Hamachi by LogMeIn

Table of Contents

PUBLISHED BY
LogMeIn, Inc.
320 Summer Street Suite 100
Boston, MA 02210

Overview

This document provides an overview and protocol-level details of the security architecture employed by LogMeIn Hamachi.

Audience

This document is written for security professionals and assumes advanced knowledge of applied cryptography and security practices.

Abbreviations

AE
Authentication Exchange
CE
Configuration Exchange
KE
Key Exchange
SA
Security Association (IPsec specification)
SPI
Security Provider Index (IPsec specification)
ESP
Encapsulated Security Payload protocol (IPsec specification)

Security in LogMeIn Hamachi v2.x

Hamachi v2.x – Summary of Changes

- The client-server authentication is password-based; the server stores the password in a salted hashed form
  - However the initial login (the enrollment) still uses RSA to authenticate the server
- The client does not register or store its public key on a server; server never sees client's public key
- The client-to-client authentication can utilize RSA, PSK or PKI certificates
- The tunnel is never used for bulk traffic transfer unless it is secured with a client-to-client key exchange
- Client-to-client tunnel security and configuration protocol is redesigned

Hamachi v2.x – The Enrollment

The Hamachi v2.x client account is identified by a username. The username is an opaque string automatically assigned by the server when the account is created. The username cannot be changed and it is not guaranteed to be human-readable.

The server also generates a random password for an account. Default password size is 64 symbols. Both the username and the password are passed back to the client in a response to its enrollment request.

The server's enrollment response also includes RSA signature of the handshake hash. This allows the client to ensure it creates an account with the server it wants (and not some random server posing as a Hamachi backend).

Hamachi v2.x – The Password

The server stores the password in a hashed form. Furthermore, to prevent dictionary attacks against leaked password hashes, the server employs salted hashing.
Salting is a process of appending an extra string of random or somewhat random data to the password prior to hashing it. This effectively renders password recovery through the use pre-computed hash tables infeasible.
The hash value of the password is further referred to as a client authentication token.
The client is allowed to change the authentication token at will (given its control session is an authenticated state). The client application also has an option to not store the token in a disk configuration file, in which case the user will be prompted for a password before the login.

Hamachi v2.x – The Login

The Hamachi v2.x login is essentially the same as Hamachi v1.x login in a pre-shared key mode.

The client sends the username and the HMAC of the handshake traffic. HMAC is computed using the client's authentication token as a password.

The server locates the client's account record, verifies the HMAC, and responds with its own value of the handshake HMAC.

Hamachi v2.x – Client-server security

Hamachi v2.x introduces no major changes in the way the client-server control connection is secured.

Hamachi v2.x – Client-to-client security

The Hamachi v2.x client implements a revised version of the tunnel security setup.

A combination of the cipher, the MAC, their keys, and other traffic processing options is referred to as Security Association (SA).

Simply put, the SA includes all the information needed to tunnel or de-tunnel the traffic to/from the peer. The Security Provider Index (SPI) value is an ID of an SA and it is included in all packets exchanged by the peers.

Each tunnel has two SAs; one SA per direction. Both SA and SPI are established terms from the IPsec specification.

Hamachi v2.x tunnels are comprised of one or more links. A link provides a simple connectivity service between the clients. A link may be direct or relayed, it may be UDP or TCP based, it may or may not be established with a help of the Hamachi server.

Hamachi v2.x – Client-to-Client Security – Exchanges

Hamachi v2.x tunnel configuration and setup is based on three exchanges. All three exchanges use the same initiator-responder mechanism and the same packet header format. They all negotiate new SA and they share a logic used to activate new SA. These similarities are intentional as they simplify the implementation.

KE - Key Exchange	KE generates shared secret used to protect the rest of the tunnel traffic
AE - Authentication Exchange	AE authenticates the KE
CE - Configuration Exchange	CE adjusts tunnel options such as traffic compression and encryption

General Exchange Structure

- The Initiator of the exchange selects a new random 32 bit SPI value and sends an “offer” message:

```
<exchange-type> | SPI_i |     0 | <exchange-specific data>
```

- The responder decides whether to accept or reject the offer. The offer is rejected by aborting the exchange as described below.
- If the responder accepts the exchange it creates its own SPI value and prepares a new SA and replies with:

```
<exchange-type> | SPI_r | SPI_i | <exchange-specific data>
```

At this point the responder has two SAs – old and new. It continues to use the old SA for the outbound traffic. Inbound traffic is checked against both the new and old SAs. Once the new SA is used, the old SA is discarded.

- The initiator receives a response. Similar to the responder, it has an option of aborting an exchange if it does not like the response.
- If the initiator accepts the response, it creates a new SA.
- At this point the initiator also has two SAs and it starts processing all outbound traffic using the newer one. Inbound traffic is checked against both new and old SAs. Once the new SA is used, the old SA is discarded.
- The initiator periodically re-sends the request until it receives a reply or exhausts a number of retransmissions. The exchange is initiator-driven.
- The responder aborts the exchange by sending:

```
<exchange-type> |     0 | SPI_i | <exchange-specific data>
```

It can only abort the exchange as a response to one of the initiator’s messages.

- The initiator aborts the exchange by sending:

```
<exchange-type> |     0 | SPI_r | <exchange-specific data>
```

And the responder acknowledges it with:

```
<exchange-type> |     0 | SPI_i | <exchange-specific data>
```

Symmetrical Exchange

A *symmetrical exchange* occurs when both sides simultaneously initiate an exchange. Clients know they are in this situation if they receive an OFFER in response to their own OFFER. Instead of trying to complete a symmetrical exchange, clients implement a simple initiator-responder election mechanism instead. Each client compares its username to the username of the peer. The client with a "smaller" username (as per certain ordering criteria) becomes a responder, and its peer an initiator. The initiator drops the peer's request and awaits a reply to his own. The responder does the opposite: it cancels its own request and replies to the peer's. See also Appendix A – Tunnel Exchanges.

Hamachi v2.x – Client-to-client security – Initial SA

When an individual link is first established, it may or may not have an Initial SA set up. For example, when the server establishes a direct or relayed link, it provides both clients with a pre-generated keying material. This enables clients to establish an SA.

On the other hand, if the link is manually configured, the clients do not have any shared secret data, so the link does not have an initial SA.

This SA, when present, is used for traffic authentication only; and not its encryption. Its sole purpose is to provide DoS attack protection for the key exchange that follows.

The use of initial SA is somewhat unique to Hamachi v2.x. Traditional IKE-based VPN systems are susceptible to DoS attacks against the initial handshake, and this is considered a known deployment risk.

Initial SA is referred to as Weak SA. The name is due to the fact that it is derived from a server-supplied keying material. This theoretically grants the server full access to any client-to-client traffic protected under this SA. As a result, the SA is not used for anything but preliminary authentication of the key exchange traffic.

Summary

If the server provides an initial keying material for a link, the Weak SA is established. This is a link SA and it is used for traffic authentication only. The SA exists until the KE is executed between the clients, at which point a tunnel SA is established. The latter is then used to process the traffic on all links, thus making all individual link SAs obsolete.

Hamachi v2.x – Client-to-Client Security – Overview

Once the Hamachi v2.x tunnel gets its first operational link, the clients execute a key exchange. The KE generates an SA that is based on a secret known only to the clients. In other words, it excludes the server "from a loop".

Initial key exchange is followed by an Authentication Exchange (AE). It confirms the identities of both clients and verifies that KE was not a subject to a Man-in-the-Middle attack.

The key exchange may be re-run later on, which is also known as re-keying. It typically is not followed by an AE, because the authenticity of the exchange parties is simply inherited from the existing SA.

Overview of the Tunnel Security Setup Process

1 Link A is up.

   If the server provides keying material, Weak SA is set up for this link.

2 Client initiates KE with a peer

   If KE messages are sent over A, then its Weak SA is used to authenticate them.

3 KE is completed.

   This creates Anonymous SA, which a shared tunnel SA.

   Weak SAs on all links are discarded.

4 Client initiates AE with a peer.

   The traffic is protected by Anonymous SA.

5 AE is completed.

   This creates Strong SA (for the lack of better name) and completes the setup.

Hamachi v2.x – Client-to-Client Security – KE Details

The key exchange is modeled loosely after JFK and IKEv2 protocols and is a request-response protocol. The flow of the exchange is as follows.

1 A generates new SPI value, new Nonce , private DH key and public DH exponent

2 A sends SPI, Nonce and public DH exponent to A

3 A receives the packet, generates its own SPI value, Nonce, private DH key and public DH exponent

4 A generates shared DH key and uses it as a keying material to create new SA

5 A sends its SPI, Nonce and public DH exponent to A

6 A receives the packet, generates shared DH key and creates an SA.

   Client A may facilitate a switch to a new SA by sending a ping-like application level message. This will be processed under the new SA; thereby causing A to switch the SA when it receives the message. In turn, A will echo the packet; this response will cause A to switch its SA.

A may choose to delay expensive operations such as DH secret computation until it receives the first packet from SA with a new SPI value. This provides a protection against CPU-bound DoS attacks. Also note that this sort of attack is only possible when the KE messages are not authenticated (when there are no link or tunnel SAs).

Both the request and the response use the same format:

`Type`	8 bits	Message type, 0xC1 for KE message
`SPI_x`	16 bits	Sender's new SPI value
`SPI_y`	16 bits	Recipient's new SPI value
`Suite`	32 bits	An ID of a pre-configured set of crypto algorithms and their parameters used by SA and KE itself. All clients must support suite 1, which is described below
`Nonce_x`	var	Sender's vector of random bytes, used for generating SA's keying material.
`DH_gx`	var	Sender's public DH exponent. The modulus and the base are defined by a suite being used.

Additionally, both clients compute a key exchange hash, which is hash of the following:

```
SPI_x, SPI_y, Suite, Nonce_x, Nonce_y, DH_gx, DH_gy
```

Where

..._x is a client's own value and
..._y is a peer's value.

This hash is later used with an Authentication Exchange.

Hamachi v2.x – Client-to-Client Security – AE Details

Authentication exchange is used to confirm the identities of the clients and to check the integrity of the key exchange. It follows the KE that generates Anonymous SA.

There are three types of authentication methods supported by the Hamachi v2.x clients:

- RSA keys
- Pre-shared key (passwords)
- PKI certificates

The protocol flow of AE does not depend on specifics of the method and it is as follows:

1 A checks its configuration and selects authentication method

2 A sends SA SPIs, auth method ID, its arguments, and an authentication hash to A

Authentication hash is computed from key exchange hash and authentication data

3 A receives the packet, checks the auth hash and a creates new SA

All parameters of new SA except for SPIs are copied from existing SA

4 A sends SA SPIs, auth method ID, its arguments and its own auth hash to A

A may use an auth method that is different from the one used by A

5 A receives the packet, checks the auth hash and creates new SA

Client A may reject the exchange in there's no matching authentication method or for some other reason. In this case, it replies with a reject message:

Type	8 bits	Message type, 0xC2 for AE message
<zero>	16 bits	
SPI_y	16 bits	Initiator's SPI value
ErrorCode	32 bits	the cause of the failure

The client may be configured to send zero as an ErrorCode for all AE failures.

RSA Keys

This is the method employed by Hamachi v1.x clients. The authentication model is similar to that of an SSH protocol. The first time a client authenticates a peer, it adds peer's public key to its key repository and marks it as unverified. The user is then expected to check key's fingerprint via an out-of-band channel. Once validated, the key is then marked as verified or trusted.

The client then also warns the user if the peer switches to using a different key as this might be a manifestation of a Man-in-the-Middle attack against AE.

The ID of this authentication method is 1. The packet format is as follows:

Type	8 bits	Message type, 0xC2 for AE message	
SPI_x		16 bits	Sender's SPI value
SPI_y	16 bits	Recipient's SPI value	
AuthMethod	8 bits	Authentication method ID, 0x01 for RSA	
PublicKey	var	Sender's public key	
Signature	var	Sender's RSA signature of an auth hash	

Authentication hash is computed as follows:

```
SPI_x, SPI_y, ID_x, ID_y, AuthMethod, PublicKey, KE_hash
```

Where ID_x and ID_y are the usernames of the local and remote client respectively. Note that when computing initiator's hash, the SPI_y value is going to be zero.

Pre-shared Key

This method is supported to allow RSA-free deployments. Both clients are configured with the matching pre-shared tunnel keys (passwords) and these are used to compute HMAC over the authentication hash. The HMAC is then sent over to the peer for verification:

Type	8 bits	Message type, 0xC2 for AE message	
SPI_x		16 bits	Sender's SPI value
SPI_y	16 bits	Recipient's SPI value	
AuthMethod	8 bits	Authentication method ID, 0x02 for PSK	
AuthHash	var	HMAC as per above	

PKI Certificate

The third authentication method is based on certified public keys. It allows establishing a validity of (previously unknown) peer's public key identity through trust in a 3rd party – a Certificate Authority.

This method is structured the same way as RSA one, except the public key certificate is included into an AE packet instead of a public key itself.

Type	8 bits	Message type, 0xC2 for AE message	
SPI_x		16 bits	Sender's SPI value
SPI_y	16 bits	Recipient's SPI value	
AuthMethod	8 bits	Authentication method ID, 0x03 for Cert	
PublicKeyCert	Var	Sender's public key certificate	
Signature	var	Sender's RSA signature of an auth hash	

Hamachi v2.x – Client-to-Client Configuration

Hamachi v2.x clients are capable of dynamically re-configuring the tunnel to toggle the use of encryption, compression and potentially other settings.

Each traffic processing option has three settings – ON, OFF and ANY. The last option indicates a willingness of a peer to accept whatever the other side wants to use.

For example, if the client wants to enable the compression, it sends an offer to the peer with ON as a compression setting. If peer's own configuration has this setting either at ON or ANY, the compression is turned on. However, if it is set to OFF, there is a conflict and the compression setting is left unmodified.

More formally, a tunnel is re-configured using a configuration exchange (CE). A peer starts CE by sending his version of the tunnel option set. Second peer responds with its own version, at which point both peers look at both options sets and deduce new set of options. In case of conflicting settings, its value copied from the existing SA.

An option set is encoded as a bitfield stored in a 32 bit integer. Each option occupies 2 bits. The value of 01 is ON, 02 – OFF, 03 – ANY. The value of 00 is reserved and should be treated as a protocol error if encountered.

When encoded, bits 0 and 1 are used for an encryption option, bits 2 and 3 – for the compression. Remaining bits are reserved at the moment and should be set to zero.

New option value is deduced from two peer's values as follows:

	ON	OFF	ANY
ON	on	no-change	on
OFF	no-change	off	off
ANY	on	off	no-change

The exchange itself progresses as follows:

1 A generates new SPI, decides on the new tunnel configuration

2 A sends new SPI and the new tunnel config encoded in 32 bit integer to A

3 A receives the packet

4 A generates new SPI, deduces effective tunnel configuration and creates new SA

5 A sends back SPI and its own version of the tunnel config

6 A receives the packet, deduces effective config and creates new SA

Encryption and authentication keys are inherited from existing SAs, only SPIs and traffic processing options are updated.
The activation process for new SAs is the same as that of a Key Exchange.
The handling of a symmetrical exchange start is also the same as with the KE.

The “offer” packet:

Type	8 bits	Message type, 0xC3 for CE message	
SPI_x		16 bits	Sender’s SPI value
<zero>	16 bits	Recipient’s-would-be-SPI-value	
Options	32 bits	Initiator’s configuration option set	

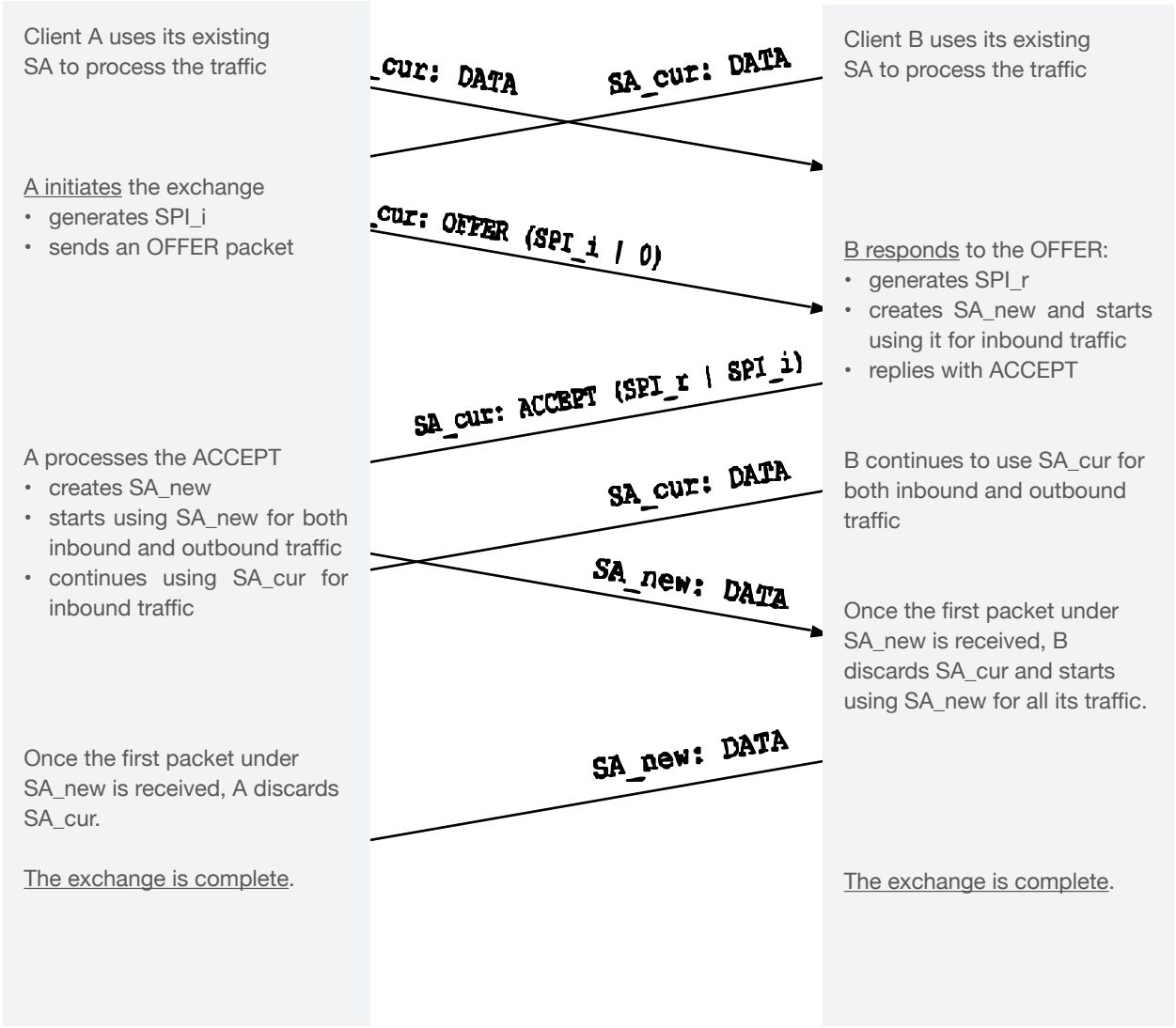
The “accept” packet:

Type	8 bits	Message type, 0xC3 for CE message	
SPI_x		16 bits	Sender’s SPI value
SPI_y	16 bits	Recipient’s SPI value	
Options	32 bits	Responder’s configuration option set	

Appendix A – Tunnel Exchanges

Typical Flow

The following diagram demonstrates a general flow of the events during the exchange. Both sides are assumed to have an existing SA (SA_cur) before the exchange starts.



Packet Loss and Excessive Link Latency

Client A periodically re-sends its OFFER if it does not hear back from A within a timeout period. Once A exhausts all retransmission attempts, it cancels the exchange.

If A responds to the OFFER after A has cancelled the exchange, A sends a REJECT packet, thus forcing A to discard its SA_new. A is expected to respond with its own REJECT to confirm the reception of “A’s” REJECT.

Similar to the OFFER, A retransmits REJECT few times if A does no respond in a timely manner. A aborts the retransmission sequence if another exchange is initiated by either side.

REJECT retransmits are really for “B’s” own benefit – they try to make sure that A knows that the exchange has been cancelled. Client A may in fact not send a single REJECT and the logic described below will still ensure the tunnel is always in a consistent state.

“B’s” SA_new can be viewed as an initially disabled SA that is activated by a packet received from A. This means that if A decides not to use SA_new, A may be required to discard the SA without ever using it.

Specifically, in cases when there is a packet loss in a B-to-A direction, A may never receive any of the ACCEPT packets. Client A will exhaust its retransmission attempts and cancel the exchange. A, however, will generate SA_new.
In this scenario, A is required to discard its SA_new under either of two conditions:

- It receives new OFFER from A
- It makes its own OFFER and receives a response from A (either ACCEPT or REJECT)

This behavior also covers the case when either A rejects “A’s” OFFER or A rejects “B’s” ACCEPT and these REJECT packets are lost.

When can the new exchange be started

The key exchange and the configuration exchange are an offer-accept or an offer-reject sequence. However the authentication exchange may be longer - offer-accept-reject-reject.

As a result of this, the conditions under which the new exchange may be started depend on the exchange type and the role of the peer in a preceding exchange.

The Initiator:

KE, CE, AE – after receiving ACCEPT or REJECT.

The Responder:

KE, CE – after sending ACCEPT or REJECT.

AE – after receiving REJECT or the first packet processed under the new SA