

1

InitHello { sisi, epki, sctr, pidiC, auth }

2

Line	Variables ← Action	Variables ← Action	Line
IHI1	ck ← lhash("chaining key init", spkr)	ck ← lhash("chaining key init", spkr)	IHR1
IHI2	sidi ← random_session_id();		
IHI3	eski, epki ← EKEM::keygen();		
IHI4	mix(sidi, epki);	mix(sidi, epki)	IHR4
IHI5	sctr ← encaps_and_mix<SKEM>(spkr);	decaps_and_mix<SKEM>(sskr, spkr, ct1)	IHR5
IHI6	pidiC ← encrypt_and_mix(pidi);	spki, psk ← lookup_peer(decrypt_and_mix(pidiC))	IHR6
IHI7	mix(spki, psk);	mix(spki, psk);	IHR7
IHI8	auth ← encrypt_and_mix(empty())	decrypt_and_mix(auth)	IHR8

Comment

Initialize the chaining key, and bind to the responder's public key.

The session ID is used to associate packets with the handshake state.

Generate fresh ephemeral keys, for forward secrecy.

InitHello includes sisi and epki as part of the protocol transcript, and so we mix them into the chaining key to prevent tampering.

Key encapsulation using the responder's public key. Mixes public key, shared secret, and ciphertext into the chaining key, and authenticates the responder.

Tell the responder who the initiator is by transmitting the peer ID.

Ensure the responder has the correct view on spki. Mix in the PSK as optional static symmetric key, with epki and spkr serving as nonces.

Add a message authentication code to ensure both participants agree on the session state and protocol transcript at this point.

4

RespHello { sidr, sisi, ecti, scti, biscuit, auth }

3

Line	Variables ← Action	Variables ← Action	Line
RHI1		sidr ← random_session_id()	RHR1
RHI2	ck ← lookup_session(sidi);		RHR2
RHI3	mix(sidr, sidi);	mix(sidr, sidi);	RHR3
RHI4	decaps_and_mix<EKEM>(eski, epki, ecti);	ecti ← encaps_and_mix<EKEM>(epki);	RHR4
RHI5	decaps_and_mix<SKEM>(sski, spki, scti);	scti ← encaps_and_mix<SKEM>(spki);	RHR5
RHI6	mix(biscuit)	biscuit ← store_biscuit();	RHR6
RHI7	decrypt_and_mix(auth)	auth ← encrypt_and_mix(empty());	RHR7

Comment

Responder generates a session ID.

Initiator looks up their session state using the session ID they generated.

Mix both session IDs as part of the protocol transcript.

Key encapsulation using the ephemeral key, to provide forward secrecy.

Key encapsulation using the initiator's static key, to authenticate the initiator, and non-forward-secret confidentiality.

The responder transmits their state to the initiator in an encrypted container to avoid having to store state.

Add a message authentication code for the same reason as above.

5

InitConf { sisi, sidr, biscuit, auth }

6

Line	Variables ← Action	Variables ← Action	Line
ICI1		biscuit_no ← load_biscuit(biscuit);	ICR1
ICI2		encrypt_and_mix(empty());	ICR2
ICI3	mix(sidi, sidr);	mix(sidi, sidr);	ICR3
ICI4	auth ← encrypt_and_mix(empty());	decrypt_and_mix(auth);	ICR4
ICI5		assert(biscuit_no > biscuit_used);	ICR5
ICI6		biscuit_used ← biscuit_no;	ICR6
ICI7	enter_live();	enter_live();	ICR7

Comment

Responder loads their biscuit. This restores the state from after RHR6.

Responder recomputes RHR7, since this step was performed after biscuit encoding.

Mix both session IDs as part of the protocol transcript.

Message authentication code for the same reason as above, which in particular ensures that both participants agree on the final chaining key.

Biscuit replay detection.

Biscuit replay detection.

Derive the transmission keys, and the output shared key for use as WireGuard's PSK.