Initiator Code	Responder Code	Comments

1 — **InitHello { sidi, epki, sctr, pidi_ct, auth }** — 2

Line	Variables ← Action	Variables ← Action	Line	Comment
IHI1	ck ← lhash("chaining key init", spkr)	ck ← lhash("chaining key init", spkr)	IHR1	Initialize the chaining key, and bind to the responder's public key.
IHI2	sidi ← random_session_id();			The session ID is used to associate packets with the handshake state.
IHI3	eski, epki ← EKEM::keygen();			Generate fresh ephemeral keys, for forward secrecy.
IHI4	mix(sidi, epki);	mix(sidi, epki)	IHR4	InitHello includes sidi and epki as part of the protocol transcript, and so we mix them into the chaining key to prevent tampering.
IHI5	sctr ← encaps_and_mix<SKEM>(spkr);	decaps_and_mix<SKEM>(sskr, spkr, sctr)	IHR5	Key encapsulation using the responder's public key. Mixes public key, shared secret, and ciphertext into the chaining key, and authenticates the responder.
IHI6	pidi_ct ← encrypt_and_mix(pidi);	spki, psk ← lookup_peer(decrypt_and_mix(pidi_ct))	IHR6	Tell the responder who the initiator is by transmitting the peer ID.
IHI7	mix(spki, psk);	mix(spki, psk);	IHR7	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.
IHI8	auth ← encrypt_and_mix(empty())	decrypt_and_mix(auth)	IHR8	Add a message authentication code to ensure both participants agree on the session state and protocol transcript at this point.

4 — **RespHello { sidr, sidi, ecti, scti, biscuit_ct, auth }** — 3

Line	Variables ← Action	Variables ← Action	Line	Comment
RHI1		sidr ← random_session_id()	RHR1	Responder generates a session ID.
RHI2	ck ← lookup_session(sidi);		RHR2	Initiator looks up their session state using the session ID they generated.
RHI3	mix(sidr, sidi);	mix(sidr, sidi);	RHR3	Mix both session IDs as part of the protocol transcript.
RHI4	decaps_and_mix<EKEM>(eski, epki, ecti);	ecti ← encaps_and_mix<EKEM>(epki);	RHR4	Key encapsulation using the ephemeral key, to provide forward secrecy.
RHI5	decaps_and_mix<SKEM>(sski, spki, scti);	scti ← encaps_and_mix<SKEM>(spki);	RHR5	Key encapsulation using the initiator's static key, to authenticate the initiator, and non-forward-secret confidentiality.
RHI6	mix(biscuit_ct)	biscuit_ct ← store_biscuit();	RHR6	The responder transmits their state to the initiator in an encrypted container to avoid having to store state.
RHI7	decrypt_and_mix(auth)	auth ← encrypt_and_mix(empty());	RHR7	Add a message authentication code for the same reason as above.

5 — **InitConf { sidi, sidr, biscuit_ct, auth }** — 6

Line	Variables ← Action	Variables ← Action	Line	Comment
ICI1		biscuit_no ← load_biscuit(biscuit_ct)	ICR1	Responder loads their biscuit. This restores the state from after RHR6.
ICI2		encrypt_and_mix(empty());	ICR2	Responder recomputes RHR7, since this step was performed after biscuit encoding.
ICI3	mix(sidi, sidr);	mix(sidi, sidr);	ICR3	Mix both session IDs as part of the protocol transcript.
ICI4	auth ← encrypt_and_mix(empty);	decrypt_and_mix(auth);	ICR4	Message authentication code for the same reason as above, which in particular ensures that both participants agree on the final chaining key.
ICI5		assert(biscuit_no > biscuit_used);	ICR5	Biscuit replay detection.
ICI6		biscuit_used ← biscuit_no;	ICR6	Biscuit replay detection.
ICI7	enter_live();	enter_live();	ICR7	Derive the transmission keys, and the output shared key for use as WireGuard's PSK.