CS 465

Signal

Daniel Zappala, created from documentation at https://signal.org/docs/

Signal

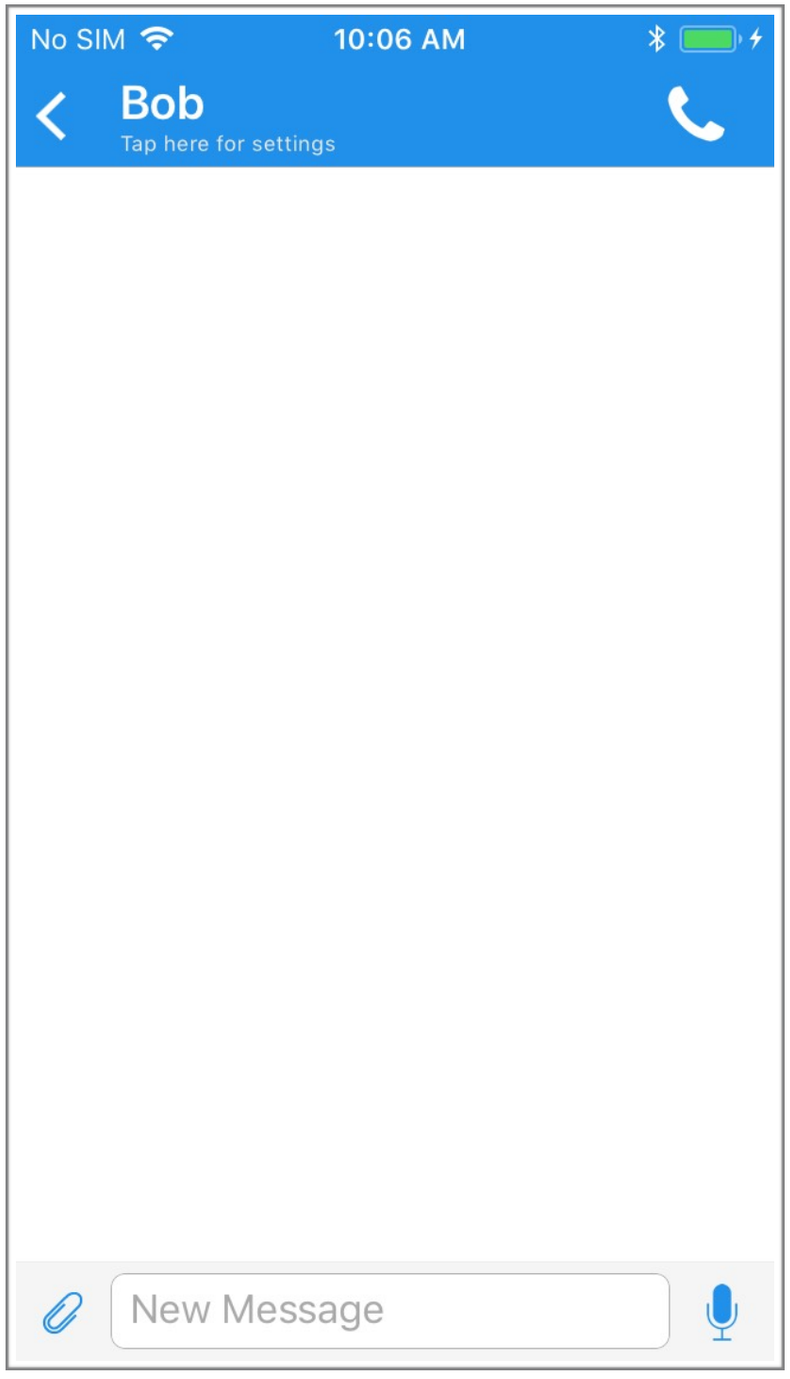
- Secure messaging app
  - messages automatically encrypted to recipients
  - mostly looks like any other messaging app

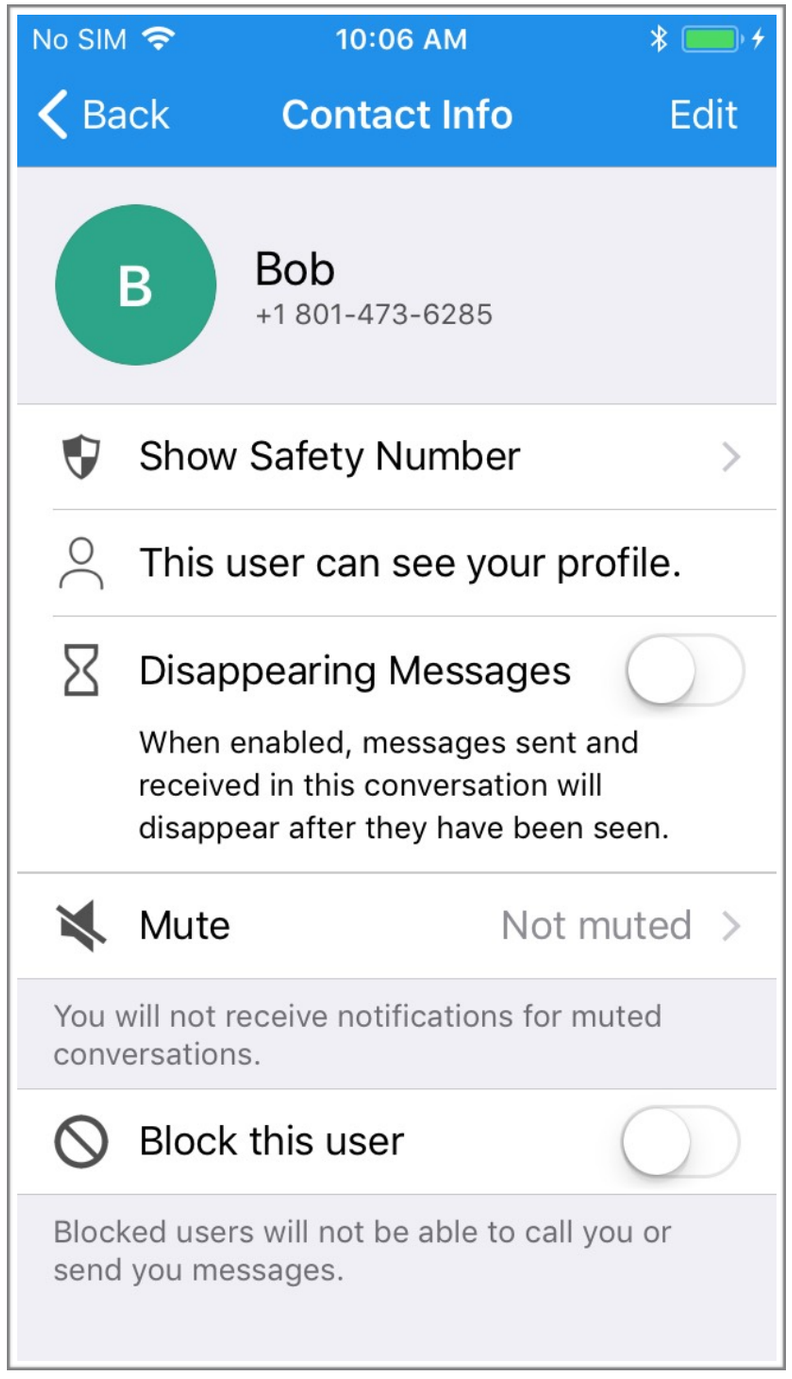
Signal Authentication

- Alice gets Bob's public key from the Signal server and uses this key to bootstrap an encrypted conversation

- If you want to be sure you have the right key

Authentication Ceremony





Authentication Ceremony

- If you want to learn more about making the authentication ceremony usable:

  - Elham Vaziripour, Justin Wu, Mark O'Neill, Daniel Metro, Josh Cockrell, Timothy Moffett, Jordan Whitehead, Nick Bonner, Kent Seamons, and Daniel Zappala, Action Needed! Helping Users Find and Complete the Authentication Ceremony in Signal, USENIX Symposium on Usable Privacy and Security (SOUPS), August 2018.

  - Elham Vaziripour, Justin Wu, Mark O'Neill, Ray Clinton, Jordan Whitehead, Scott Heidbrink, Kent Seamons, Daniel Zappala, Is that you, Alice? A Usability Study of the Authentication Ceremony of Secure Messaging Applications, USENIX Symposium on Usable Privacy and Security (SOUPS), July 2017. Errata, Corrected version.

Signal Protocol

- Extended Triple Diffie-Hellman (X3DH)
  - derive a shared key (and exchange first encrypted message)
- Double Ratchet
  - send subsequent encrypted messages
  - derive new keys for every message so that earlier keys cannot be calculated from later ones and later ones cannot be derived from earlier ones
- If a particular key is compromised for a given message, then previous and subsequent messages are not at risk

Extended Triple Diffie-Hellman (X3DH)

- establish a shared secret between two parties
- mutually authenticate using public keys
- designed for asynchronous communication (Bob is offline when Alice wants to talk to him)

X3DH: Publishing Keys

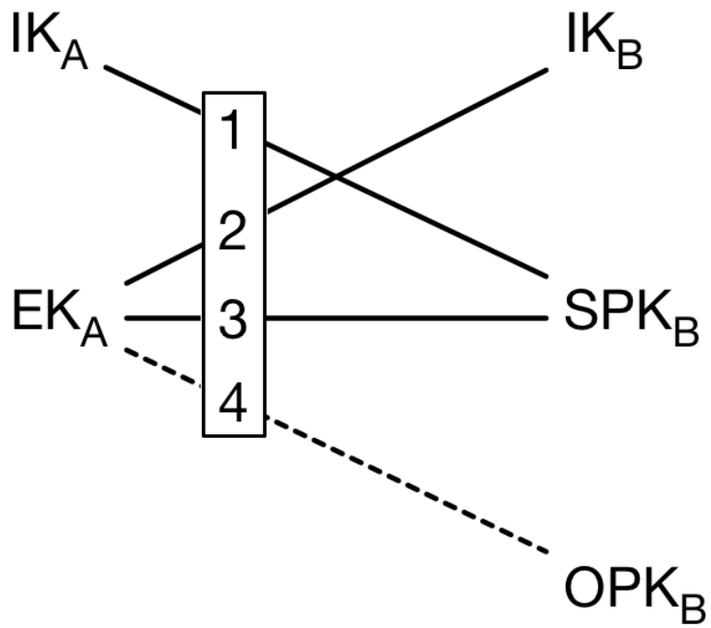
- Bob generates
  - identity key IK_B
  - signed prekey SPK_B
  - prekey signature $Sig(IK_B, Encode(SPK_B))$
  - a set of one-time prekeys OPK_B^1, OPK_B^2, OPK_B^3, …
- Uploads these keys to the Signal server
  - Bob uploads a new signed prekey and prekey signature periodically (e.g. once a month or once a week) — deletes old ones to get forward secrecy
  - one-time prekeys are deleted by the server as they are used (one per session with a contact)

X3DH: Sending the Initial Message (1)

- Alice contacts the server and asks for Bob's prekey bundle
  - Bob's identity key IK_B
  - Bob's signed prekey SPK_B
  - Bob's prekey signature $Sig(IK_B, Encode(SPK_B))$
  - (optionally) one of Bob's one-time prekeys OPK_B
    - server provides a prekey if one exists and then deletes it
- Alice verifies the signature on the prekey and aborts if it doesn't validate
- Alice generates ephemeral key pair with public key EK_A

X3DH: Sending the Initial Message (2)

- Alice computes
  - DH1 = DH(IK_A, SPK_B)
  - DH2 = DH(EK_A, IK_B)
  - DH3 = DH(EK_A, SPK_B)
  - SK = KDF(DH1 || DH2 || DH3)
- and if a one-time prekey is included:
  - DH4 = DH(EK_A, OPK_B)
  - SK = KDF(DH1 || DH2 || DH3 || DH4)



- Remember DH: Alice computes $\mathbf{g^{ba} = (g^b)^a \bmod p}$, where g^b is Bob's public value, a is her private value, g and p are public parameters.
- KDF is the HKDF key derivation function, specified by IETF
- DH1 and DH2 provide mutual authentication, DH3 and DH4 provide forward secrecy
- After calculating SK, Alice deletes private key for EK_A and DH outputs

X3DH: Sending the Initial Message (3)

- Alice calculates associated data
  - AD = Encode(IK_A) || Encode(IK_B)
- Alice sends message to Bob with
  - Alice's identity key IK_A
  - Alice's ephemeral key EK_A
  - Identifiers stating which of Bob's prekeys Alice used
  - An initial ciphertext encrypted with some AEAD encryption scheme using AD as associated data and using an encryption key which is either SK or the output from some cryptographic PRF keyed by SK
- Initial ciphertext is usually the first text message sent to Bob

X3DH: Receiving the Initial Message

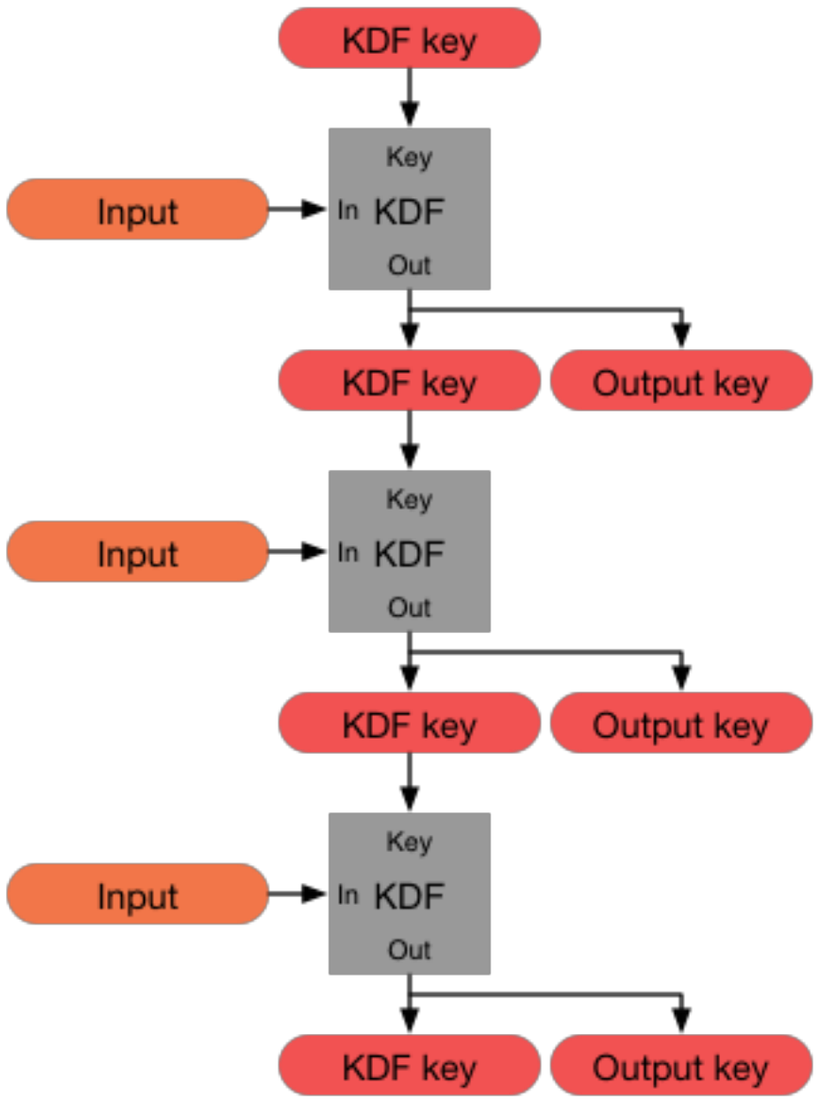
- Bob retrieves Alice's identity key IK_A and ephemeral key EK_A from the message
- Bob loads his identity private key and the private keys for the signed prekey and one-time key that Alice used
- Bob can now repeat the DH calculations and derive the same SK as Alice, then delete the DH outputs
- Bob constructs the AD byte sequence
- Bob decrypts and authenticates the ciphertext — if it fails he aborts the protocol and deletes SK
- Bob deletes the one-time prekey (providing forward secrecy)

Double Ratchet

- send subsequent encrypted messages
- derive new keys for every message so that earlier keys cannot be calculated from later ones and later ones cannot be derived from earlier ones

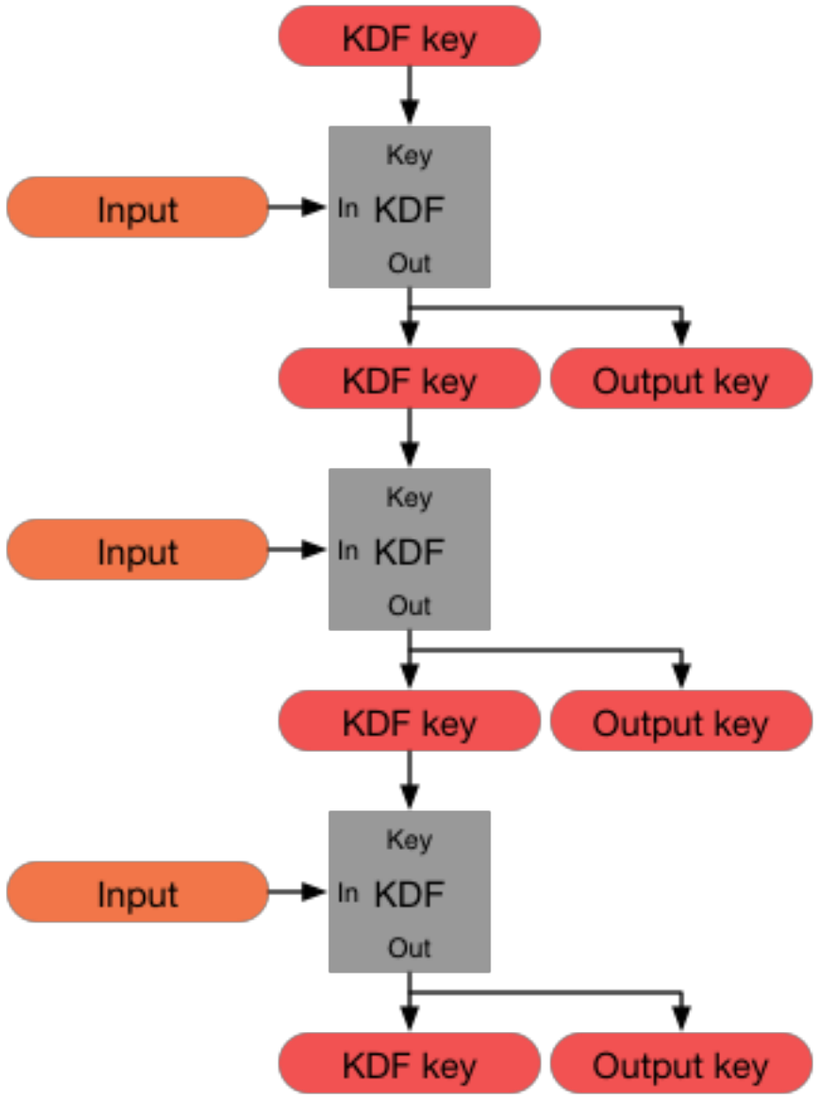
Double Ratchet: KDF Chains

- KDF is a cryptographic function
  - takes two parameters: (1) a secret and random KDF key, (2) input data
  - returns output data
- if key is not secret and random, KDF will still return a cryptographic hash of the inputs
- can use HKDF
- KDF chain splits the output into a new KDF key and an output key



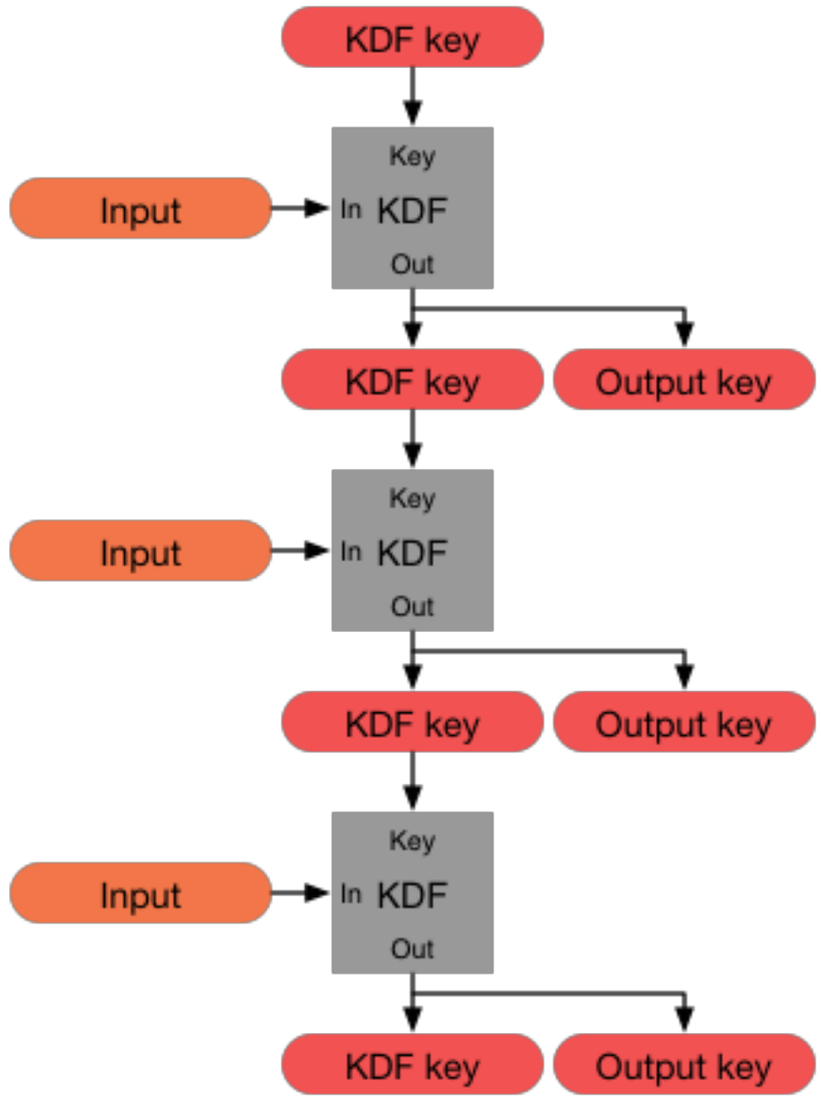
Double Ratchet: KDF Chains

- **Resilience:** Output keys appear random to an adversary that doesn't know the KDF keys, even if adversary controls inputs
- **Forward security:** Output keys from the past appear random to an adversary who learns the KDF key at some point in time
- **Break-in recovery:** Future output keys appear random to an adversary who learns the KDF key at some point in time, provided that future inputs have added sufficient entropy



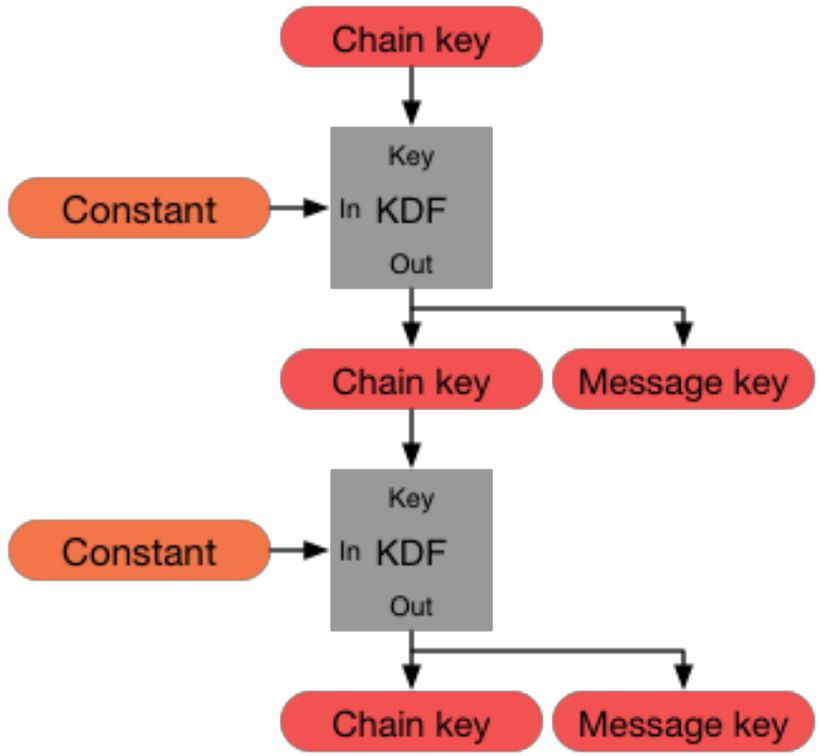
Double Ratchet: KDF Chains

- Each party stores three KDF chains:
  - root chain
  - sending chain
  - receiving chain
- Alice's sending chain is the same as Bob's receiving chain



Double Ratchet: Symmetric-key ratchet

- Each message sent is encrypted with a message key, which is the output from the sending KDF chain
- The sending/receiving chain has a constant input (no break-in recovery)
- Message key is deleted after each use (encryption or decryption)
- Calculating the next chain key and message key = ratchet



Double Ratchet: Diffie-Hellman ratchet

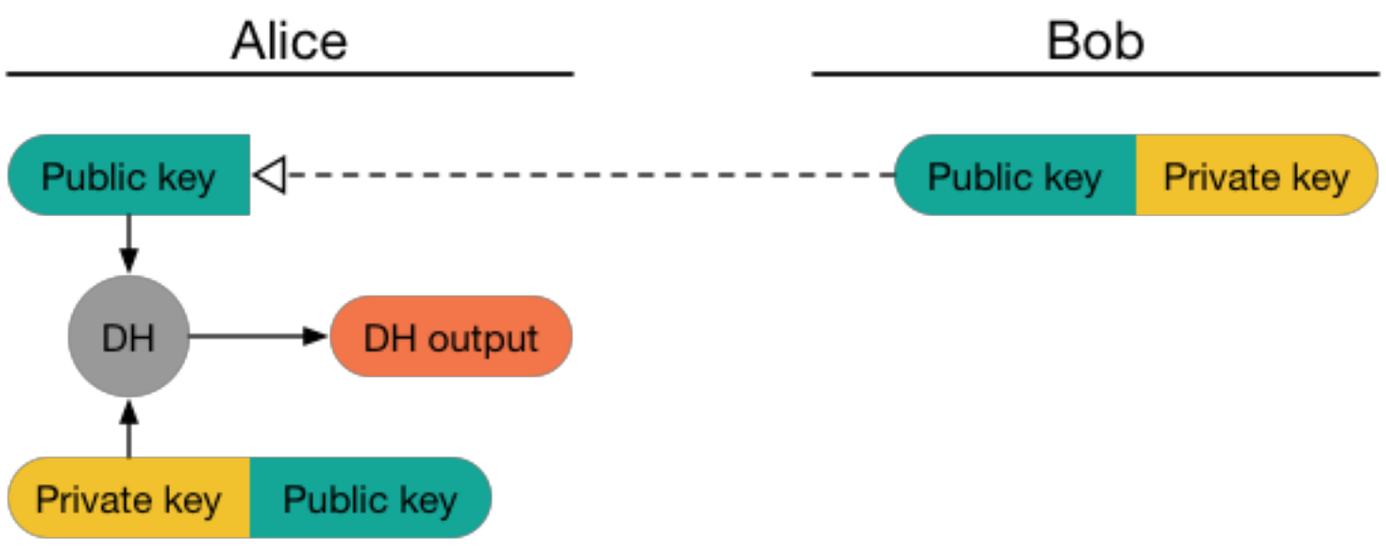
- If attacker steals sending/receiving chains from one party, then they can compute all future message keys and decrypt all future messages

- Diffie-Helman ratchet computes new chain keys resulting in a *double ratchet*

- Each time messages are exchanged, they include a new DH public key, creating new DH parameters, and these are used to create new chain keys

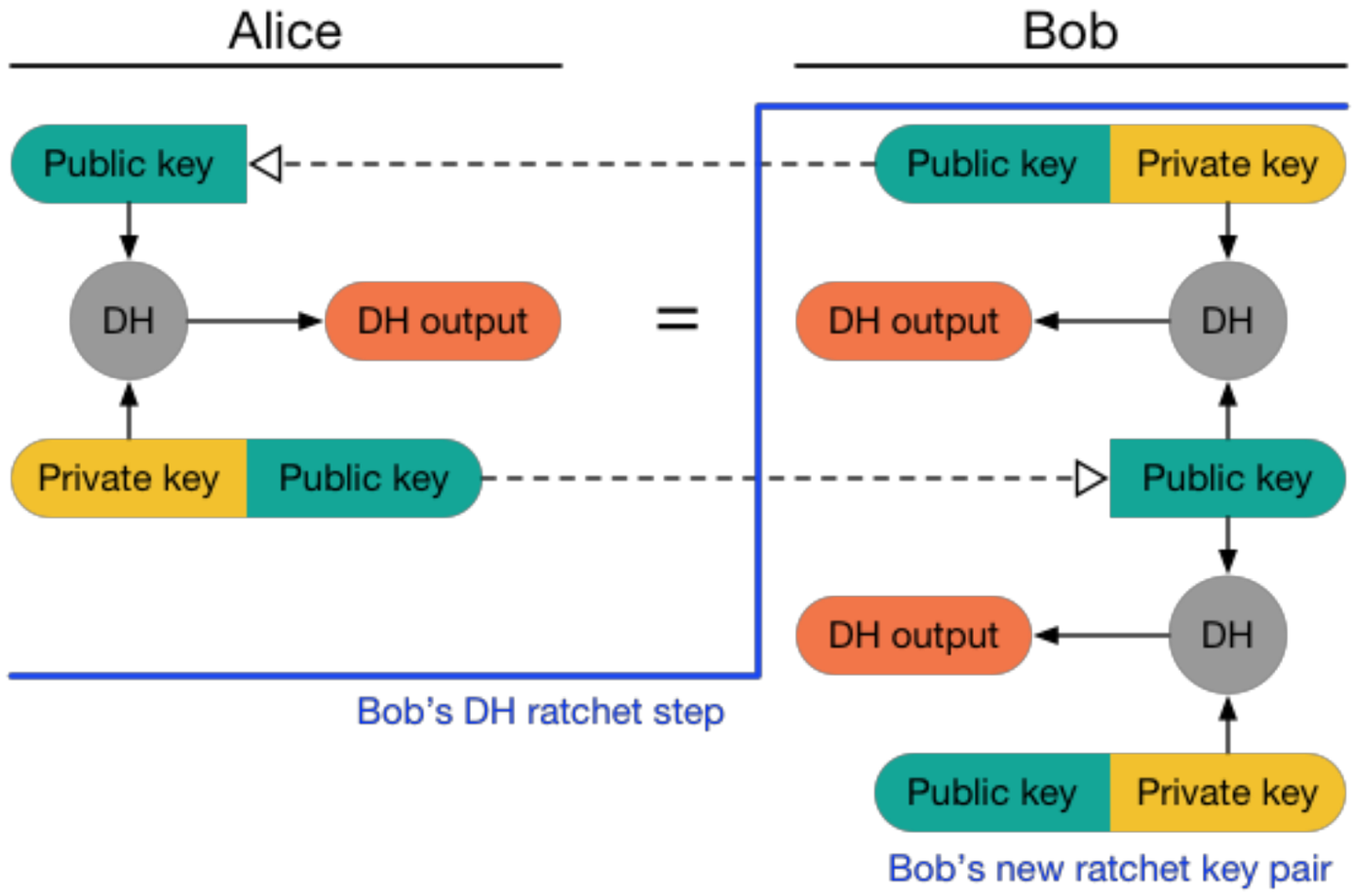
Double Ratchet: Diffie-Hellman ratchet

- Initial step



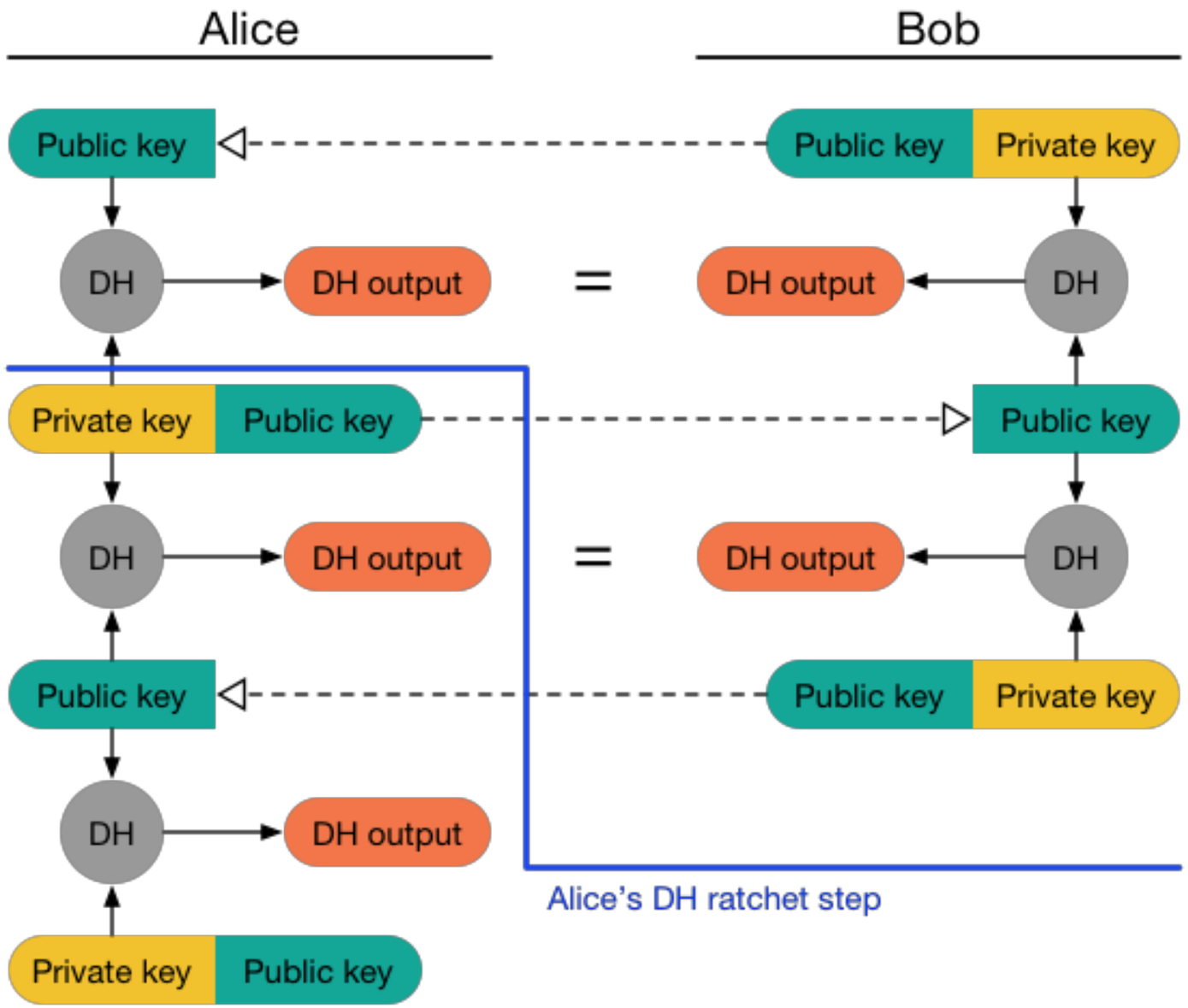
Double Ratchet: Diffie-Hellman ratchet

- Alice's initial message



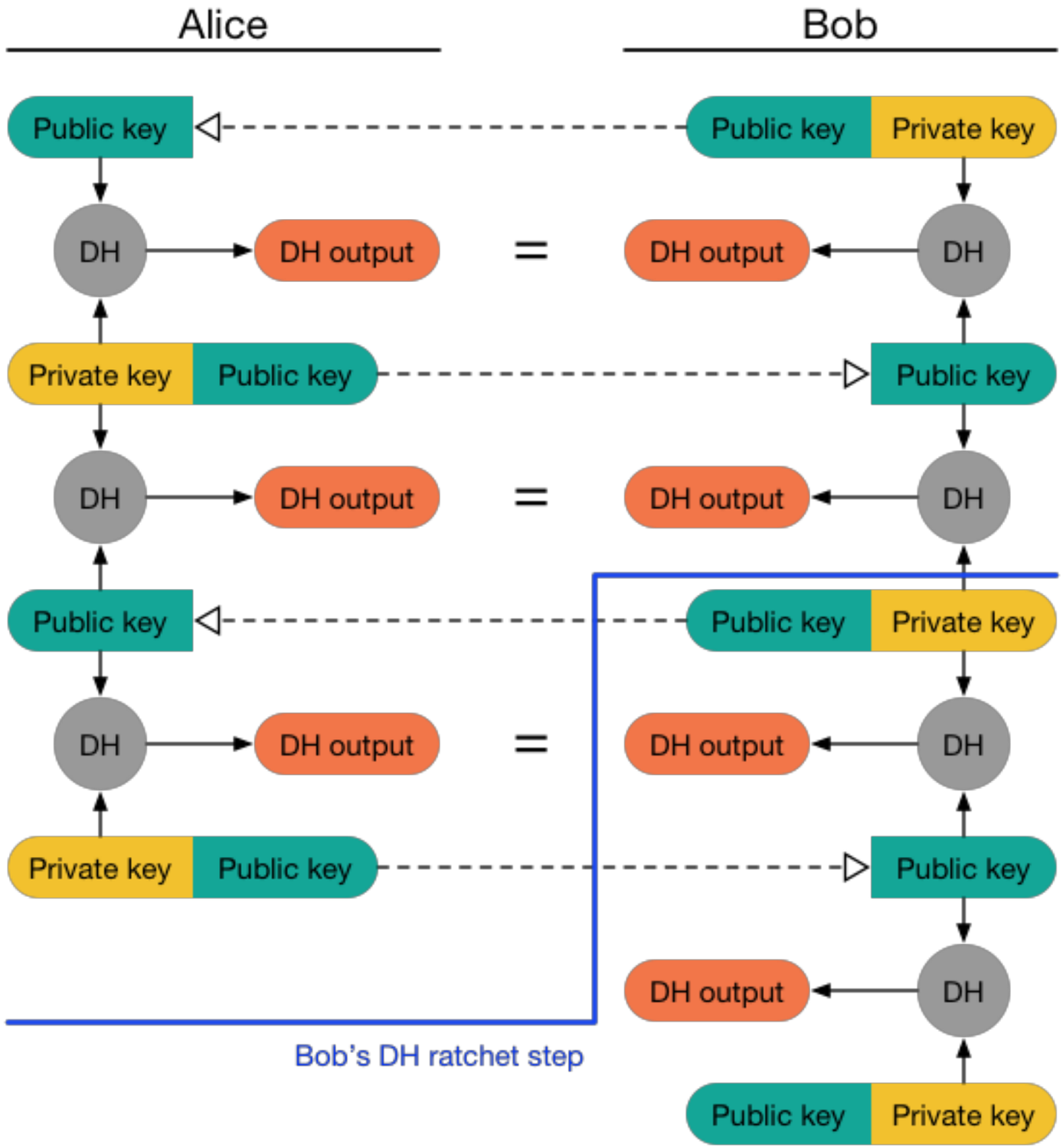
Double Ratchet: Diffie-Hellman ratchet

- Message sent by Bob



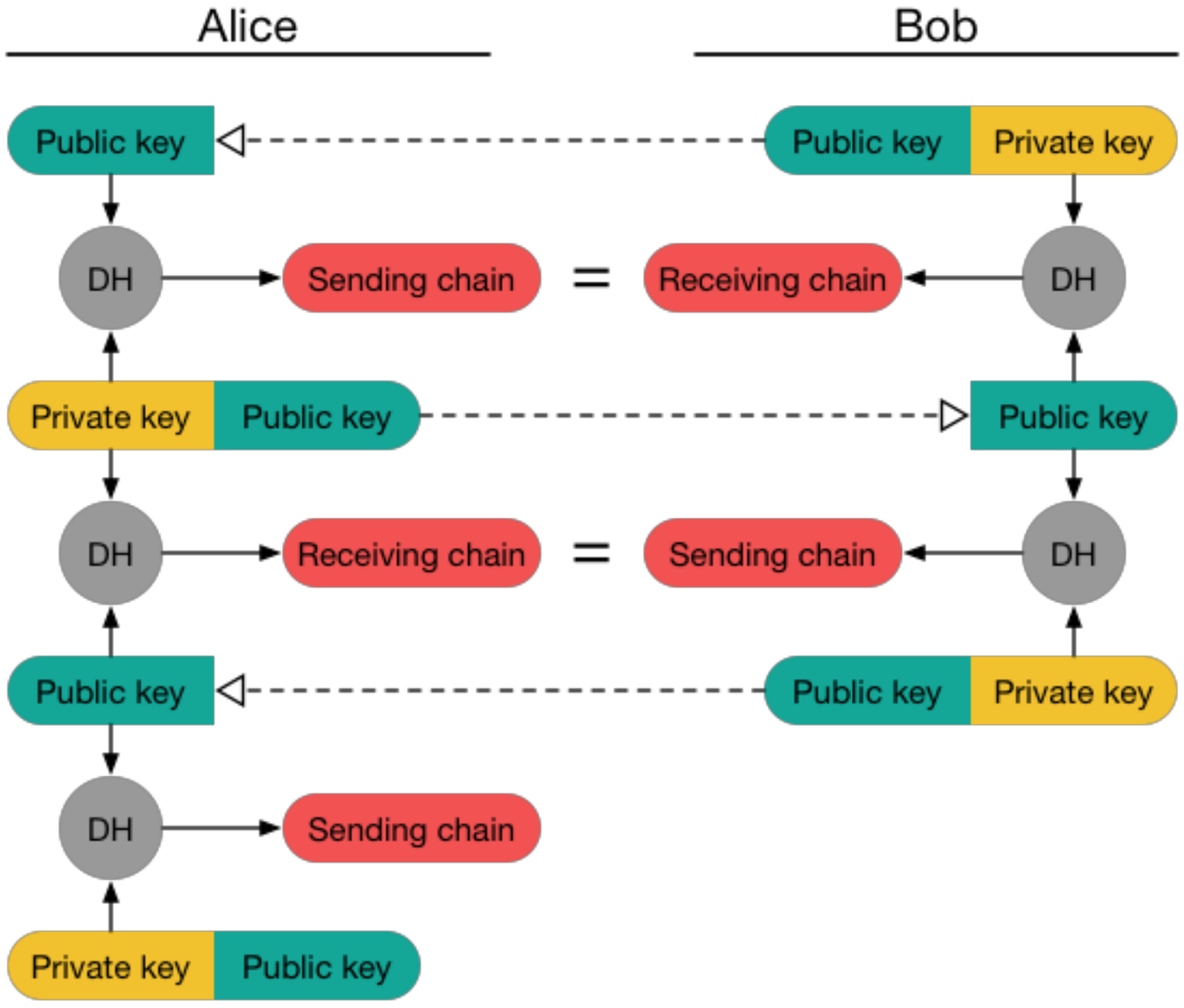
Double Ratchet: Diffie-Hellman ratchet

- Message sent by Alice



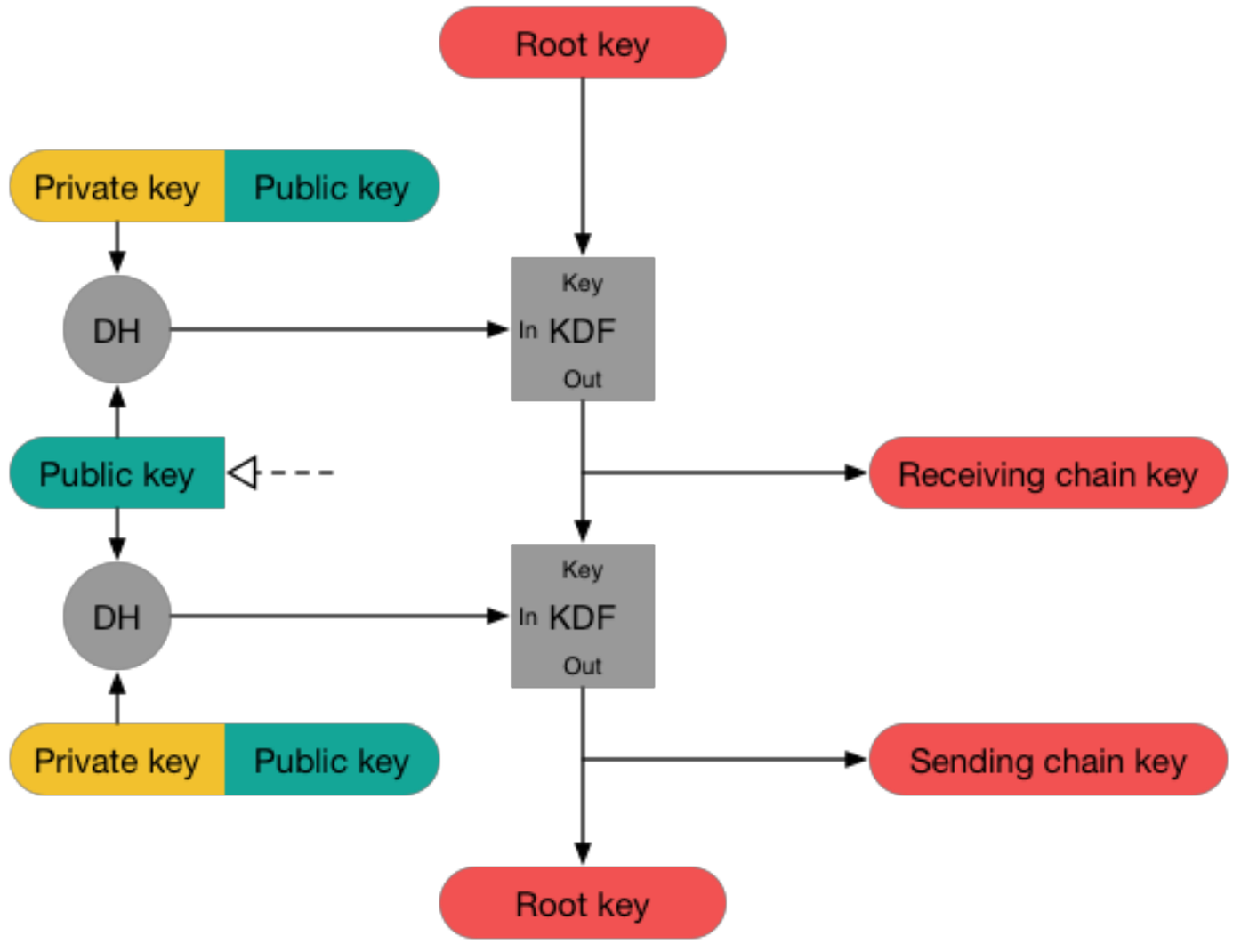
Double Ratchet: Diffie-Hellman ratchet

- Each exchange leads to a new sending/receiving chain for one party



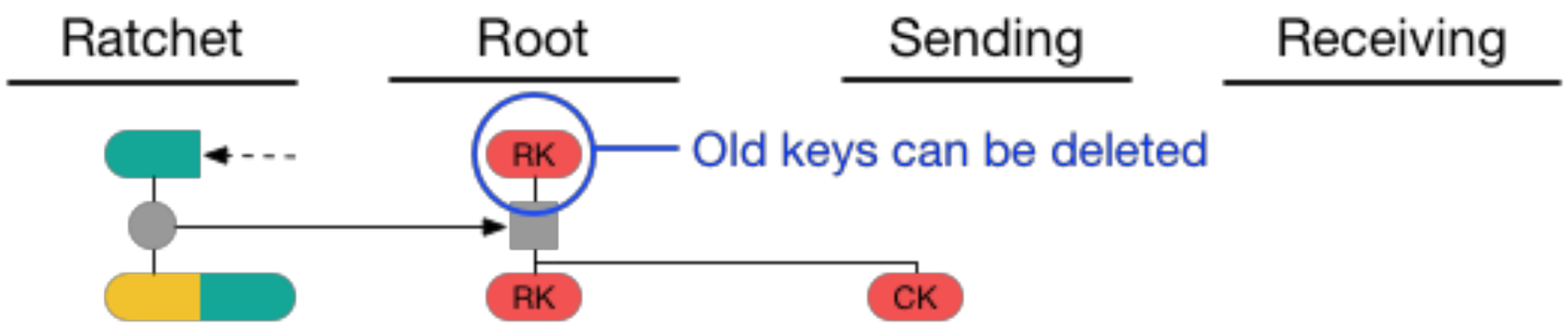
Double Ratchet: Diffie-Hellman ratchet

- Chains are actually updated at each ratchet using the root chain



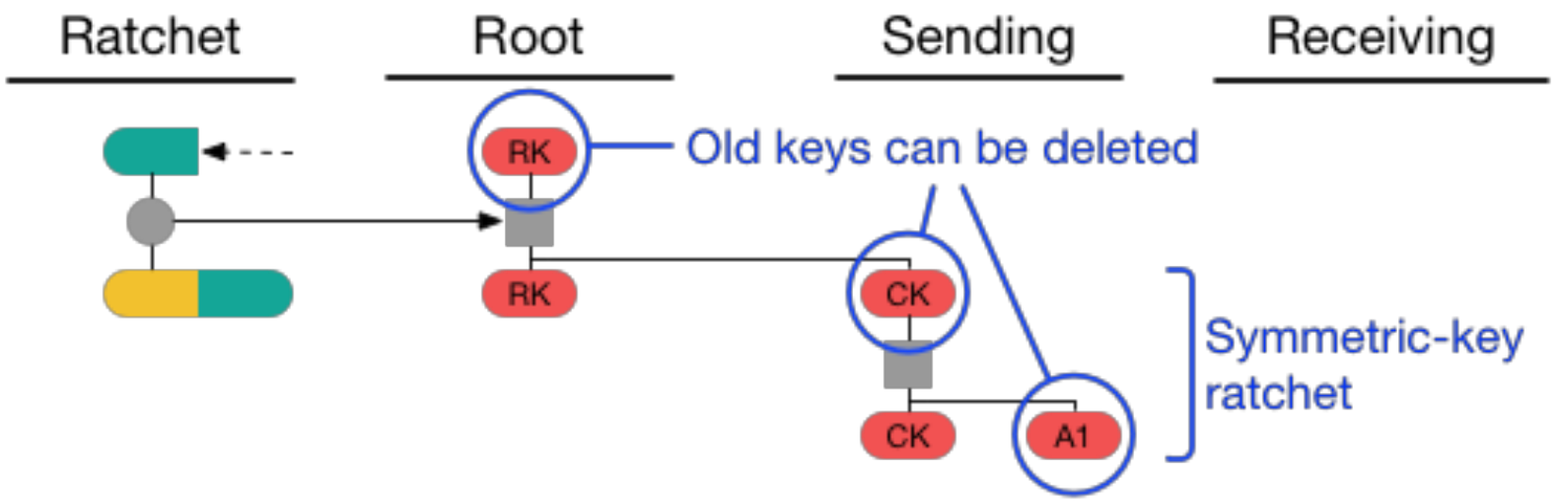
Double Ratchet: Step 1

- Alice initialized with Bob's ratchet public key and shared secret = root key
- Alice generates a new ratchet key pair, calculates DH, gives this as input to root KDF, gets a new root key and sending chain key



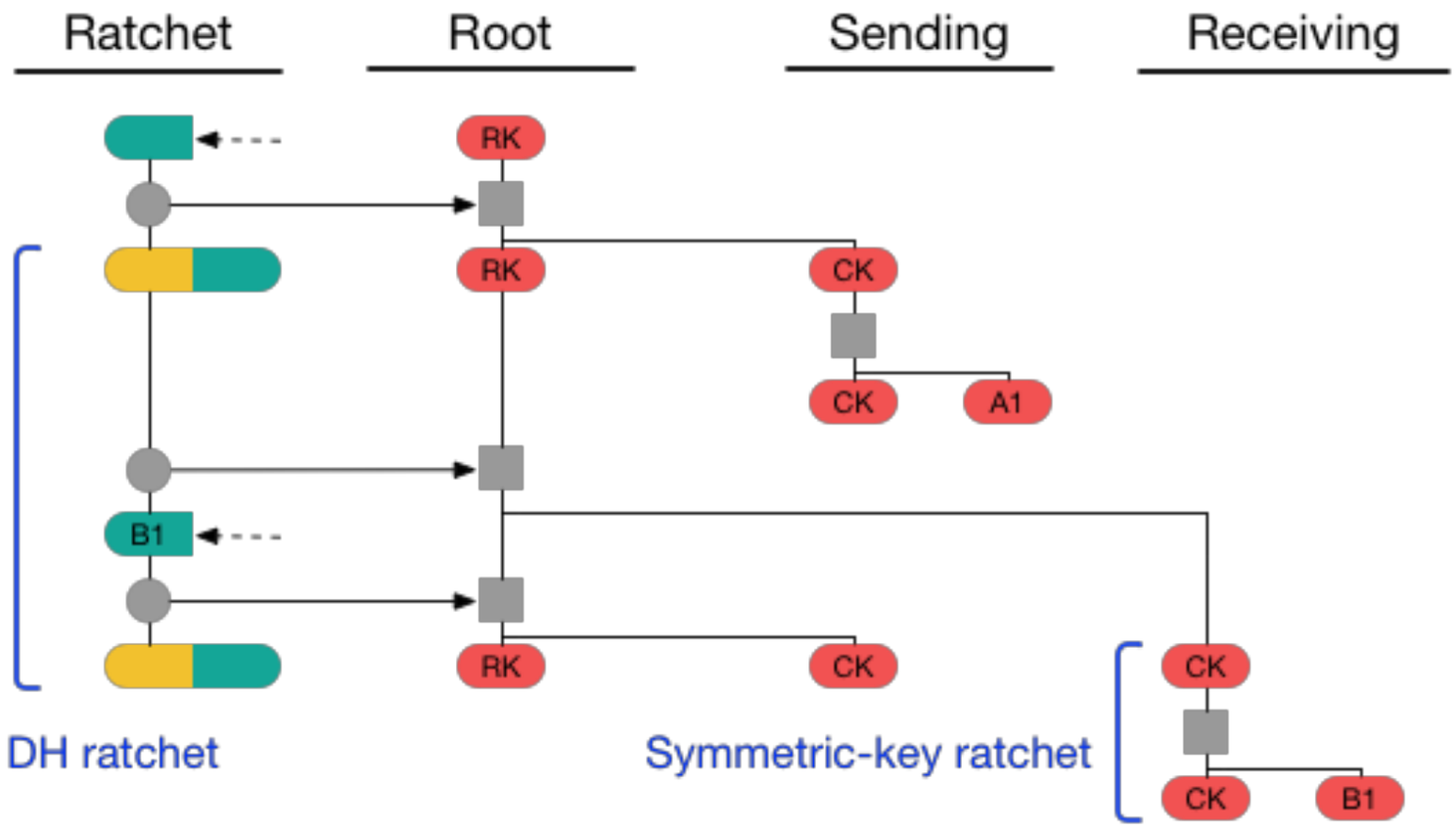
Double Ratchet: Step 2

- Alice sends her first message, and ratchets the sending chain, generating a new message key A1 that she uses to encrypt the message



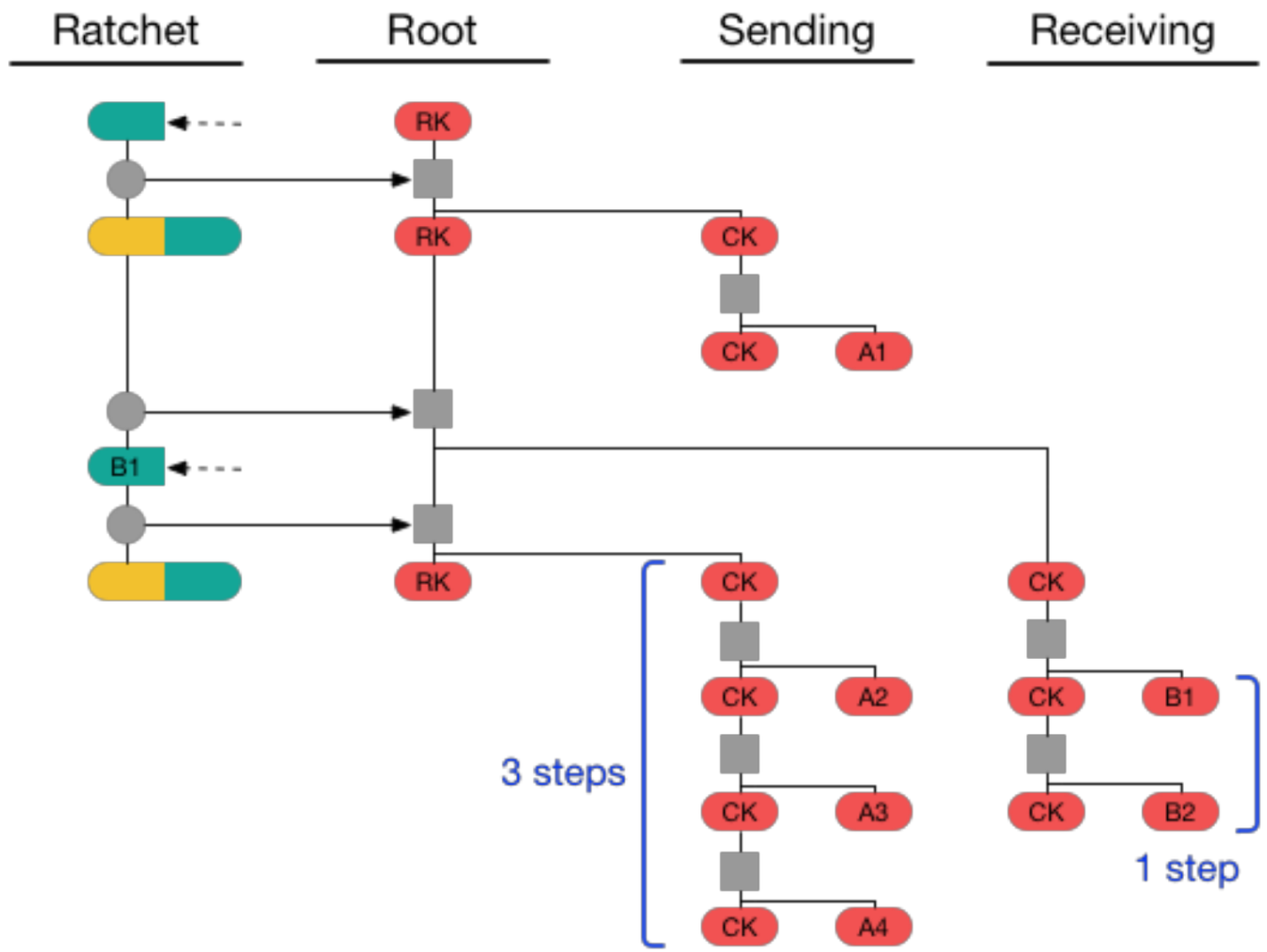
Double Ratchet: Step 3

- Response from Bob contains B1, new ratchet public key, so she does a DH ratchet, which generates a new receiving chain (for Bob), then ratchets the symmetric key to decrypt the message



Double Ratchet: Step 4

- Alice sends A2, receives B2 with Bob's old ratchet key, sends A3 and A4. Alice ratchets sending chain 3 times and receiving chain once



Double Ratchet: Step 5

- Alice gets B3, B4 with Bob's new ratchet key, then sends A5