Practical Cryptographic Systems

Signal Protocol

Instructors: Matthew Green and Alishah Chator

Logistics stuff

- Midterm at end of month
- A2 coming out tonight

News?

Review

- Last time:
  - Protocols (TLS, problems in SSLv2)
  - Attacks on SSL/TLS historically, TLS1.3!
- Today:
  - Signal protocol and forward secrecy
  - Wrapping up Protocols

TLS Review: The good and the bad

- The Good:
  - Wide scale deployment
  - Lots of security analysis
- The Bad?

Learning from TLS

Learning from TLS

- Negotiating is hard
- Backwards Compatibility is tricky
- Supporting multiple cipher suites can be risky

Learning from TLS

- Negotiating is hard
- Backwards Compatibility is tricky
- Supporting multiple cipher suites can be risky

What if we didn’t do any of these things?


Signal

Signal

- End-to-End (E2E) Encrypted Messaging Service
  - Why do we care about E2E?
- At least 40 million monthly active users
- Unsurprisingly, uses the Signal Protocol
- Supports features such as disappearing messages, fingerprint comparisons, (and soon to have MobileCoin)

History of Signal

- 2010: Release of TextSecure and RedPhone
- 2011: Whisper Systems acquired by Twitter
- 2013: Moxie Marlinspike leaves to form Open Whisper Systems
- 2014: E2E messaging added to TextSecure
- 2015: TextSecure and RedPhone merge to become Signal
- 2018: Moxie and Brian Acton form the Signal Foundation
- 2021: The great WhatsApp exodus

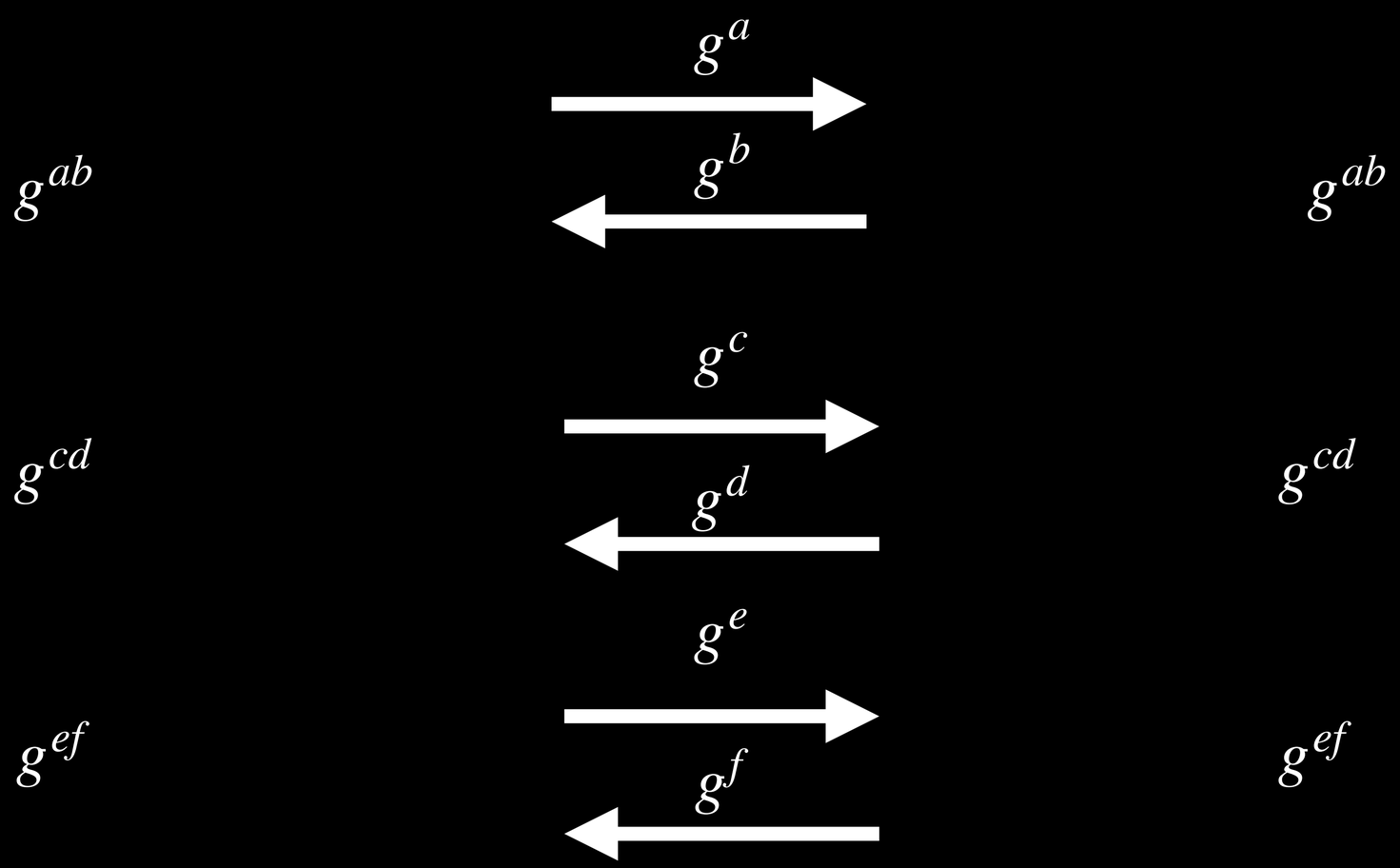
Review: Forward Secrecy

- Sometimes referred to as perfect forward secrecy
- Property that past messages are not exposed by a future compromise
- Why would we want this property for messaging?

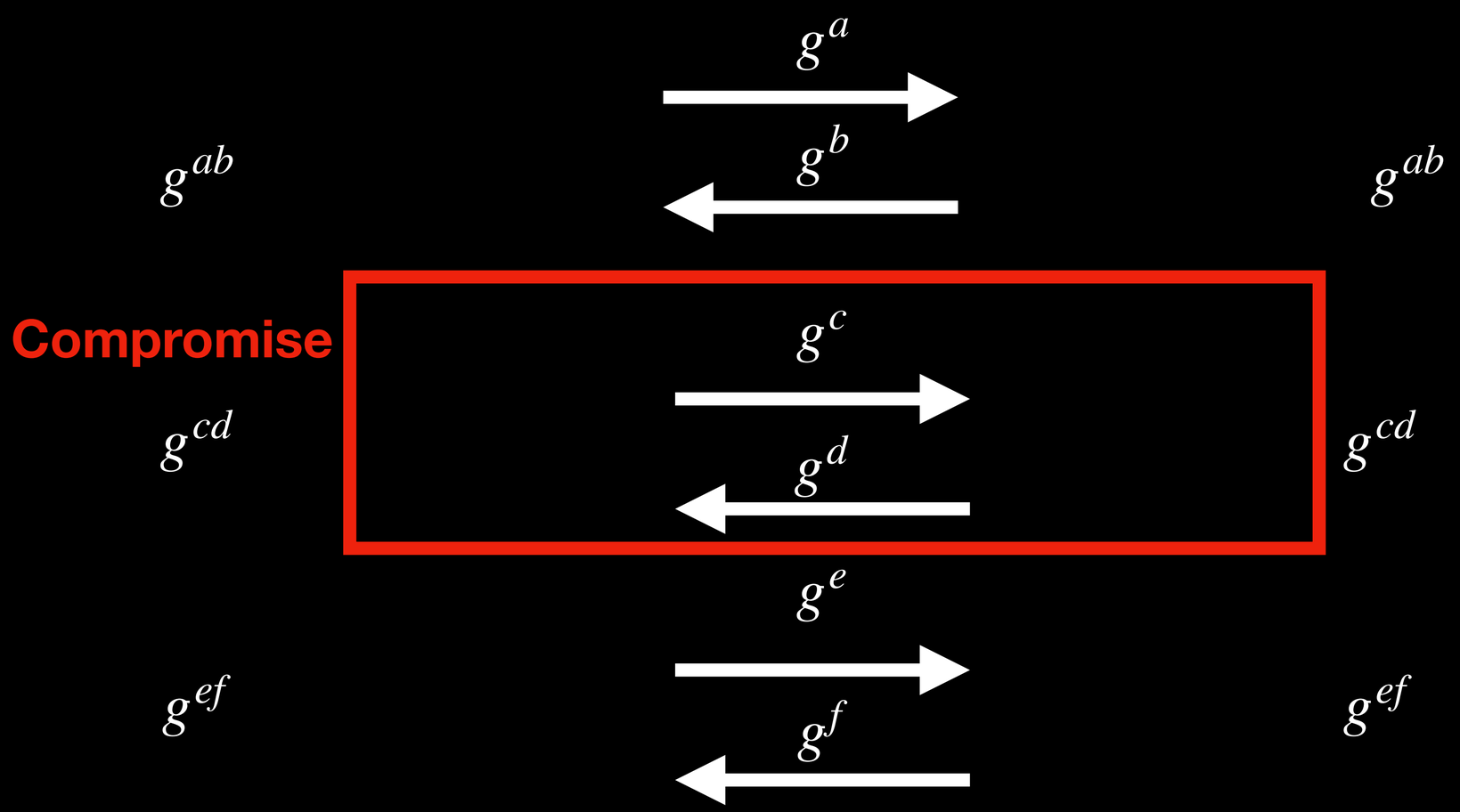
Post Compromise Security

- Property that a protocol can regain privacy after the attacker leaves the system
- Also known as Self Healing Protocols, Backwards Secrecy, Future Secrecy
- How might we achieve this?

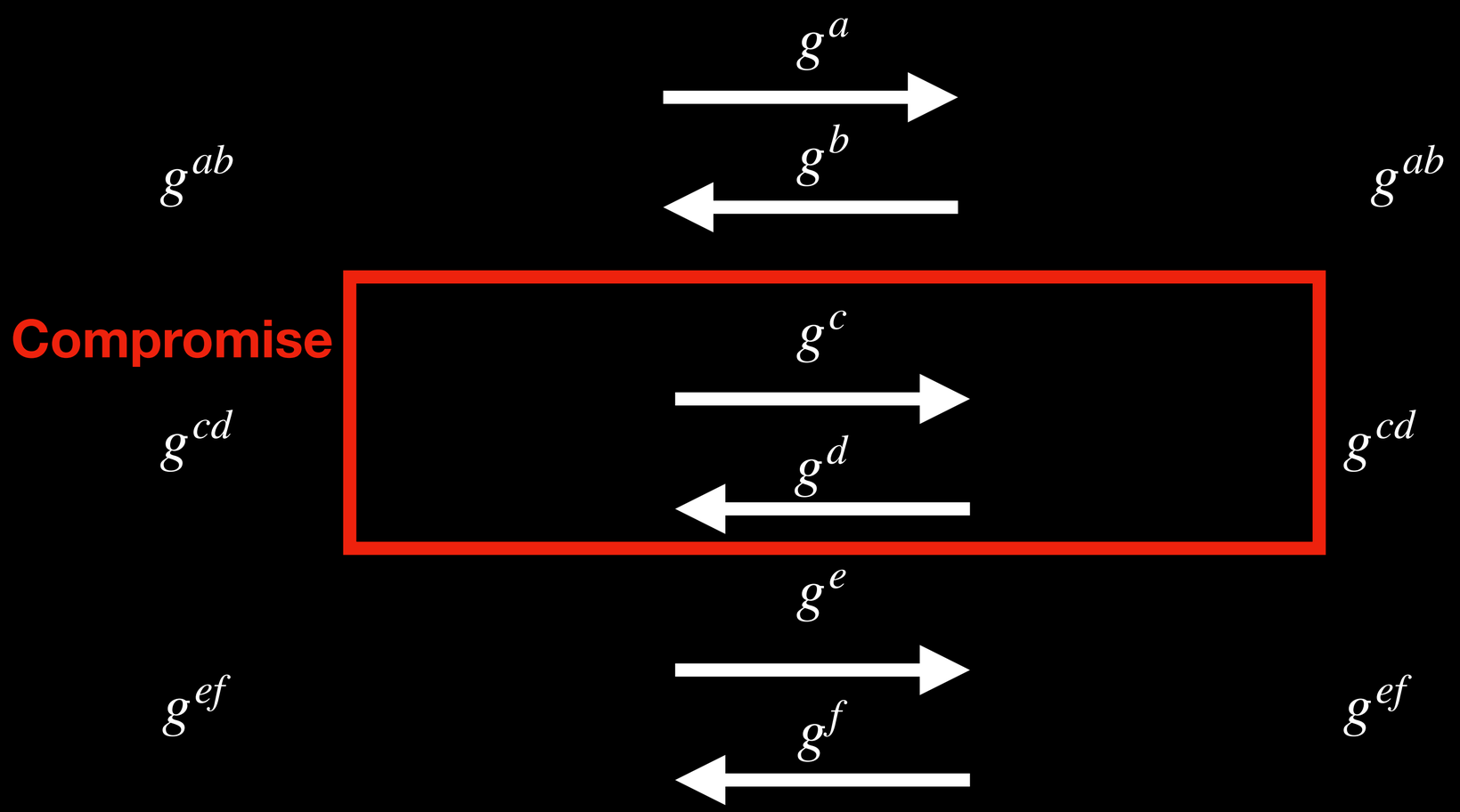
Online DH



Online DH



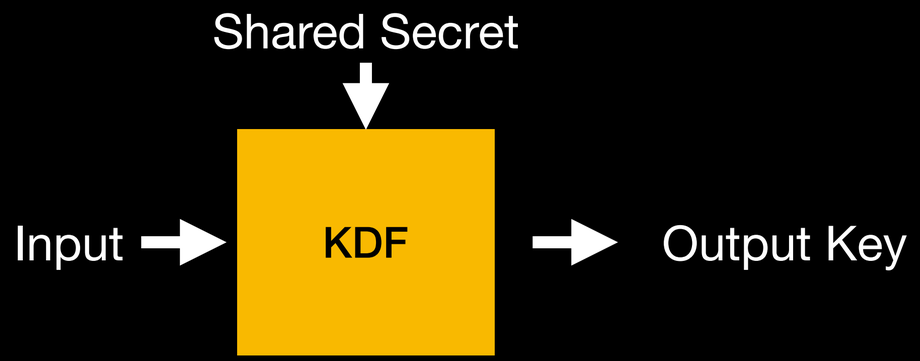
Online DH



But what if we don’t want to stay online?

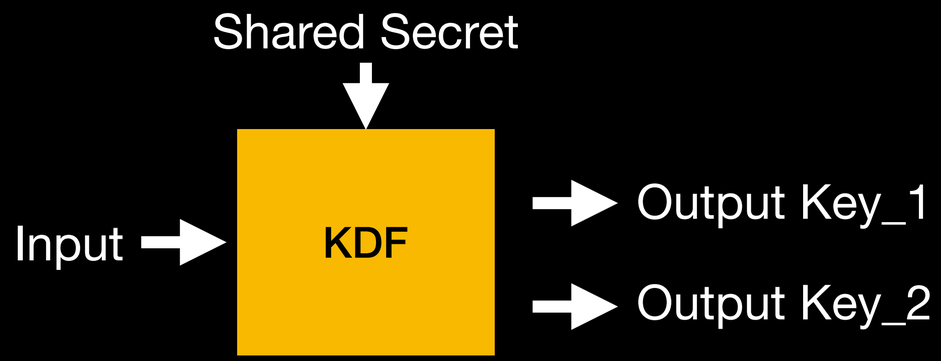
Key Derivation Functions

- Used to take a shared secret and some additional input and expand into a key
- Needs to be One-Way
  - Often instantiated using PRFs (concretely MACs) (why are MACs one-way?)



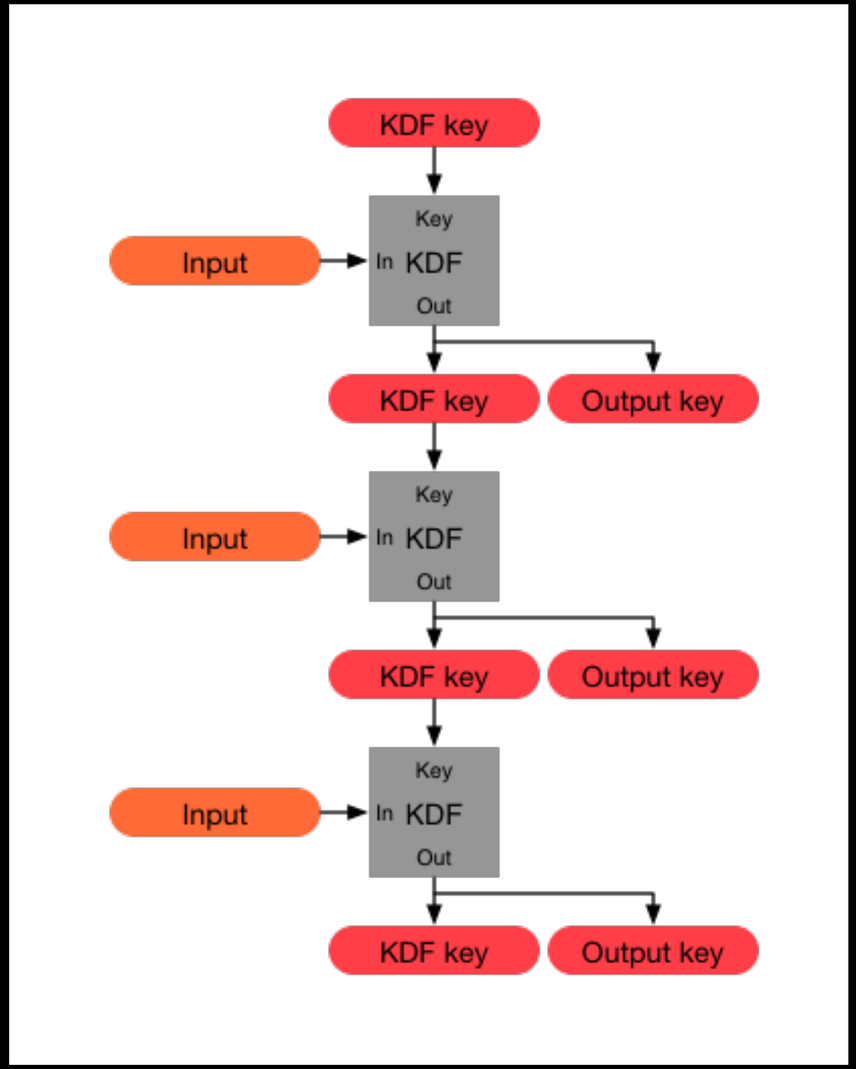
Key Derivation Functions (KDF)

- Used to take a shared secret and some additional input and expand into a key
- Needs to be One-Way
  - Often instantiated using PRFs (concretely MACs) (why are MACs one-way?)



KDF Chains

- We can feed the output of our KDF into subsequent KDFs
- Why would we want to do this?
  - Forward Secrecy?
  - Async?
  - Post Compromise Security?

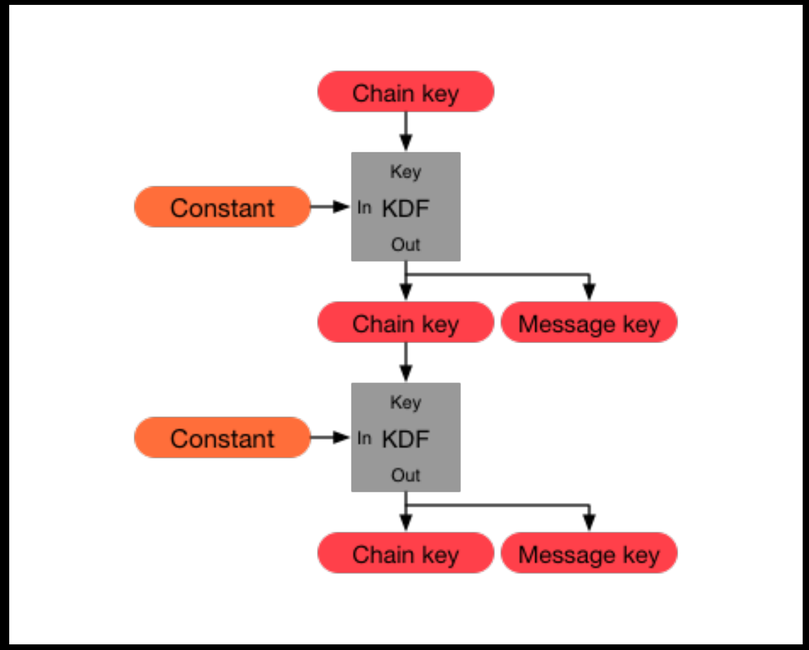


Putting it all together

- KDF chains can give us Forward Secrecy and Async
- We know we can get Post Compromise Security from continuous DH
  - Not async though!
- We can use previously discussed primitives such as AEADs (AES-GCM) to ensure messages sent are hidden and not tampered with
- Signal uses 3 KDF chains: A root chain, sending chain, and receiving chain
  - Why a separate sending and receiving chain?
  - We call this the Double Ratchet: The root chain is the DH Ratchet and the Send/Receiving chains are the Symmetric Ratchet

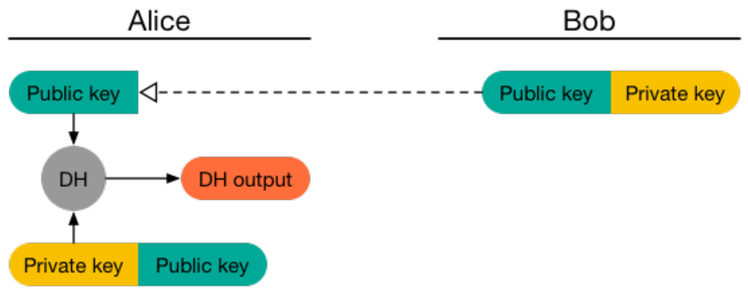
The Symmetric Ratchet

- Each message sent is encrypted by a unique message key
- We know this gives us FS and Async
- What about the chain key?



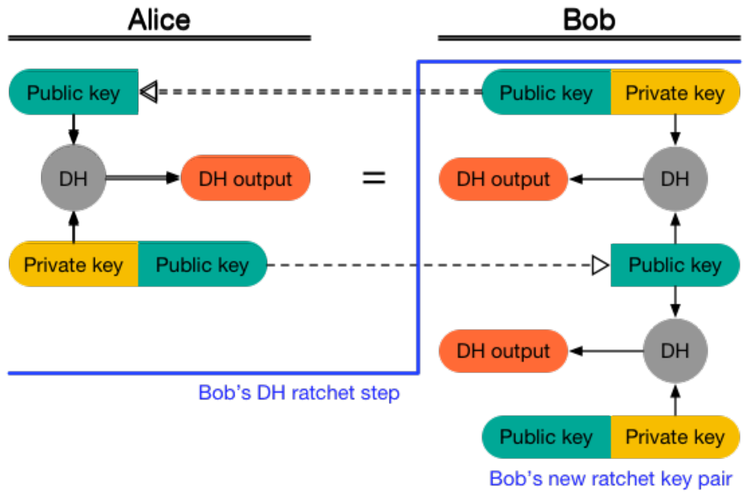
Diffie-Hellman Ratchet

- Used to update the chain keys
- Gives us PCS
- Why is this Async?



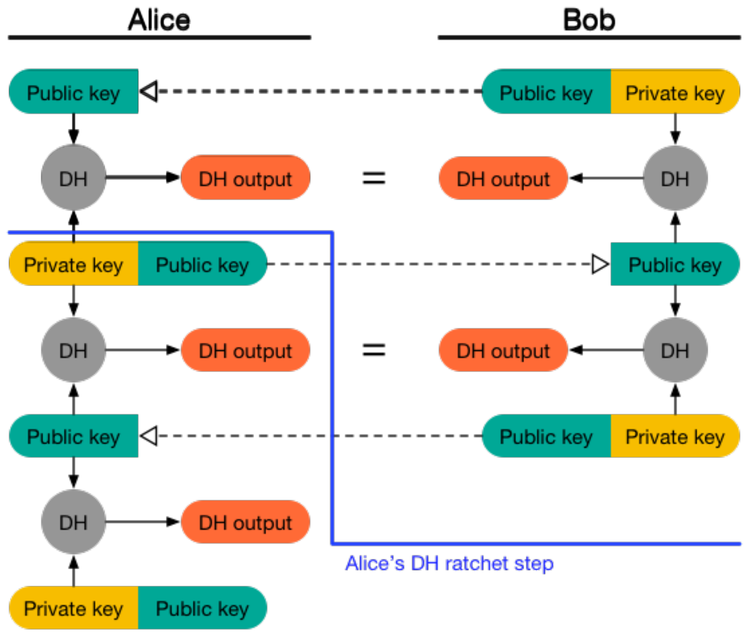
Diffie-Hellman Ratchet

- Used to update the chain keys
- Gives us PCS
- Why is this Async?



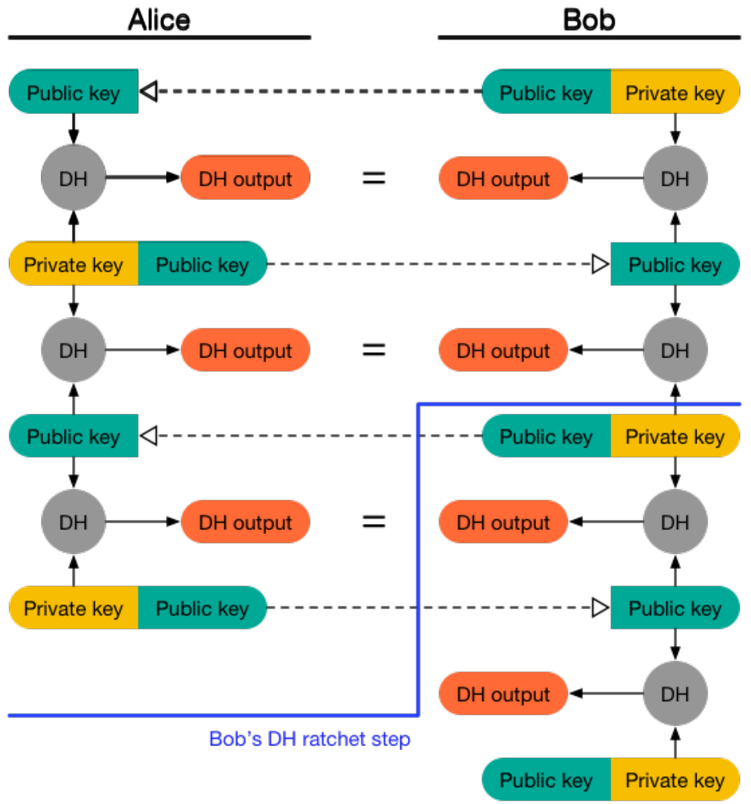
Diffie-Hellman Ratchet

- Used to update the chain keys
- Gives us PCS
- Why is this Async?



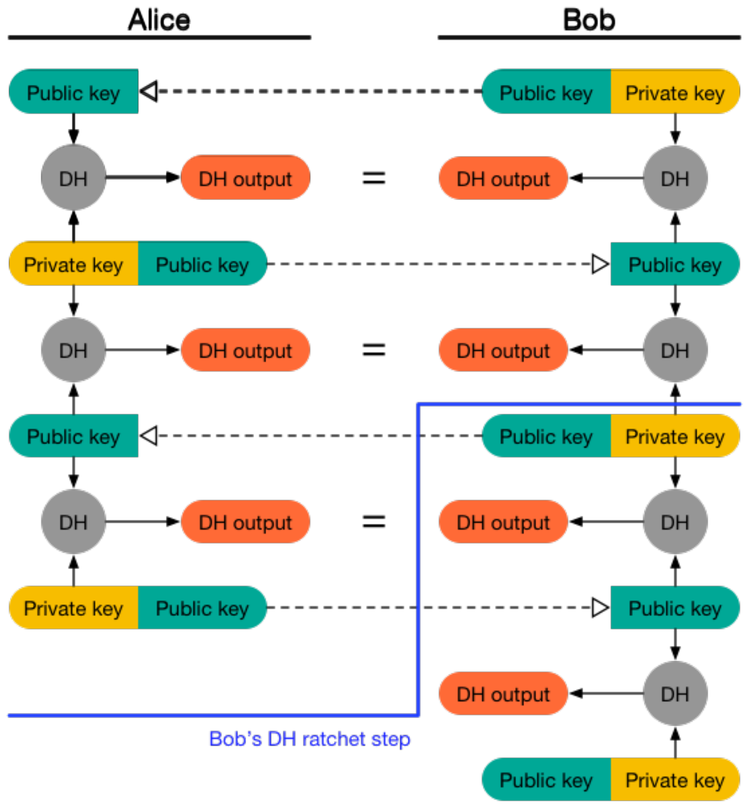
Diffie-Hellman Ratchet

- Used to update the chain keys
- Gives us PCS
- Why is this Async?



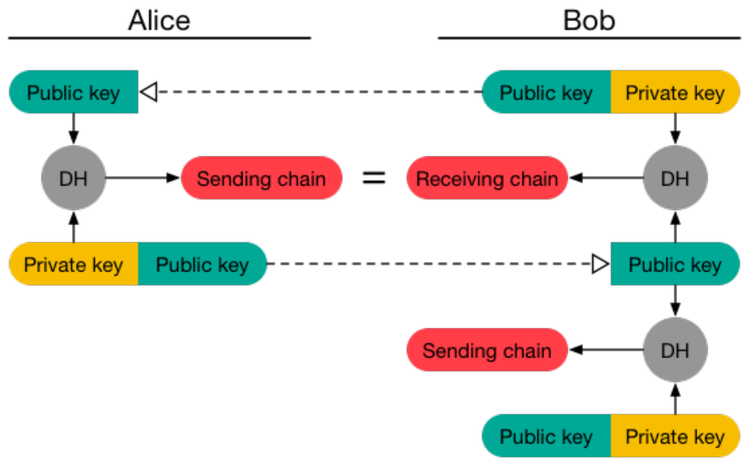
Diffie-Hellman Ratchet

- Used to update the chain keys
- Gives us PCS
- Why is this Async?
- After 2 ratchet steps we have PCS



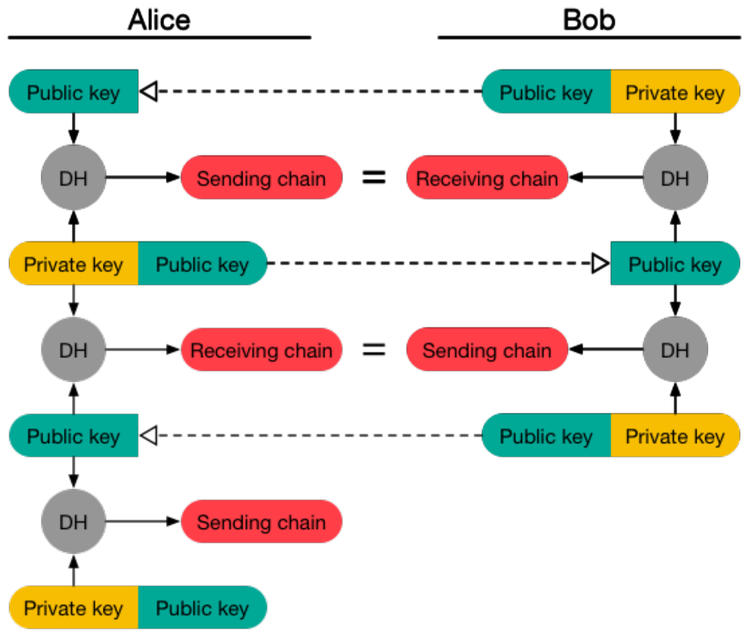
Diffie-Hellman Ratchet

- Used to update the chain keys
- Gives us PCS
- Why is this Async?
- After 2 ratchet steps we have PCS
- In reality, each party is introducing a new chain



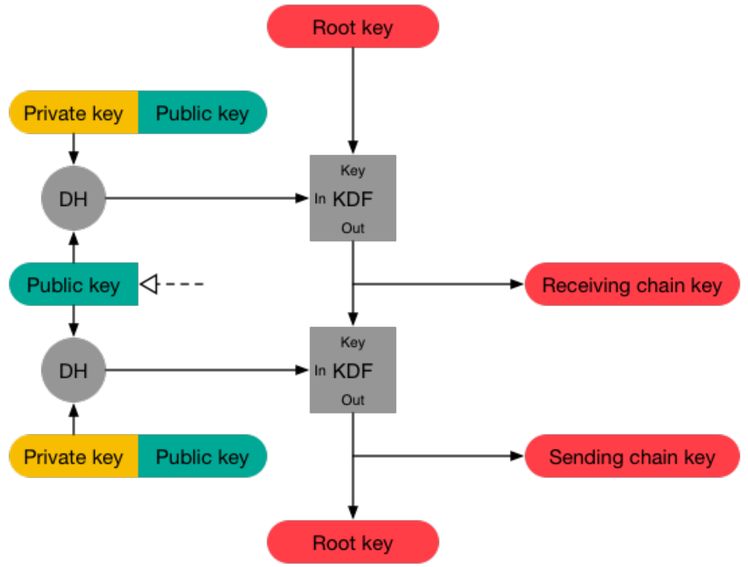
Diffie-Hellman Ratchet

- Used to update the chain keys
- Gives us PCS
- Why is this Async?
- After 2 ratchet steps we have PCS
- In reality, each party is introducing a new chain



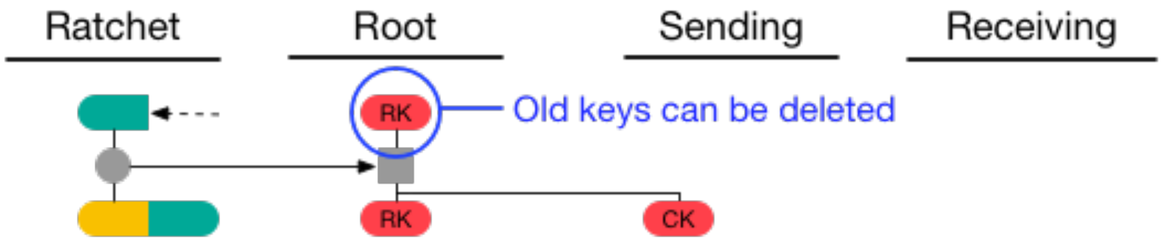
Diffie-Hellman Ratchet

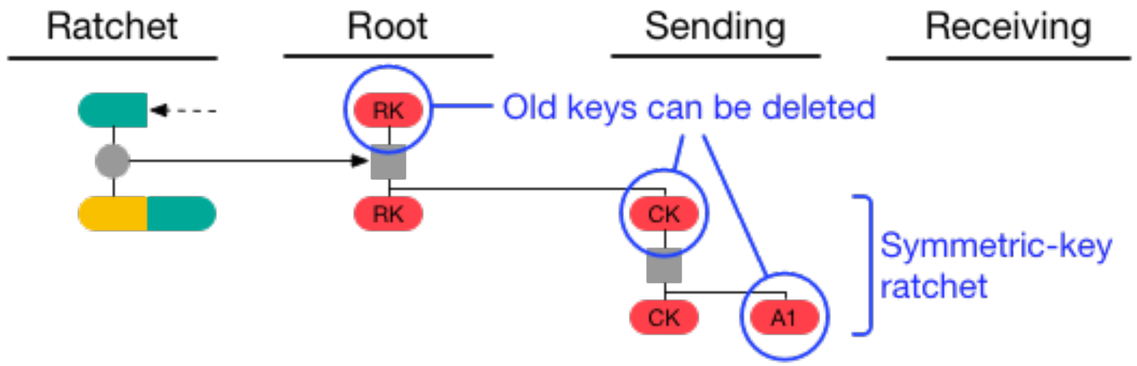
- Used to update the chain keys
- Gives us PCS
- Why is this Async?
- After 2 ratchet steps we have PCS
- In reality, each party is introducing a new chain
- DH Keys are being fed into a KDF chain as well

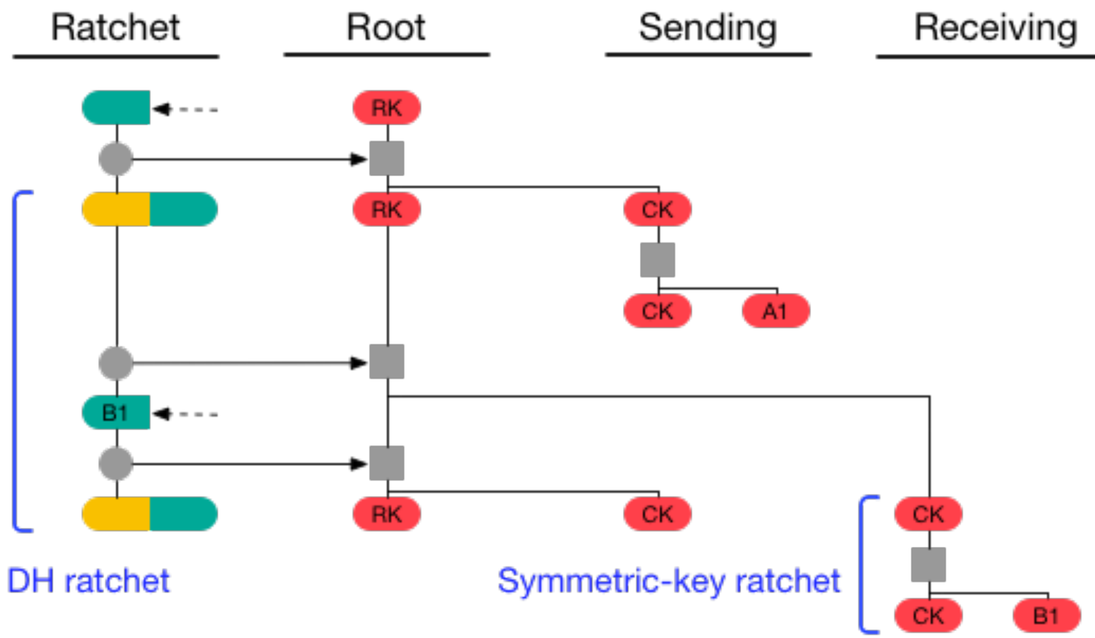


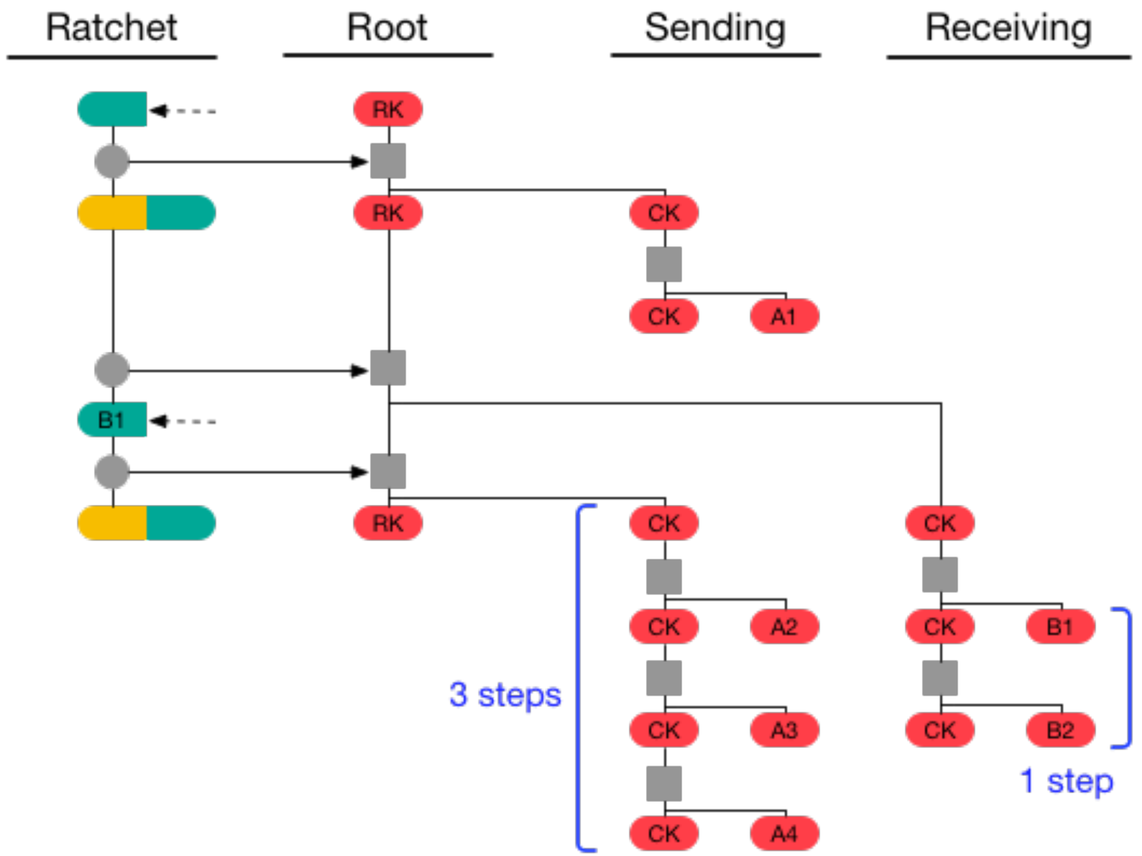
The Double Ratchet

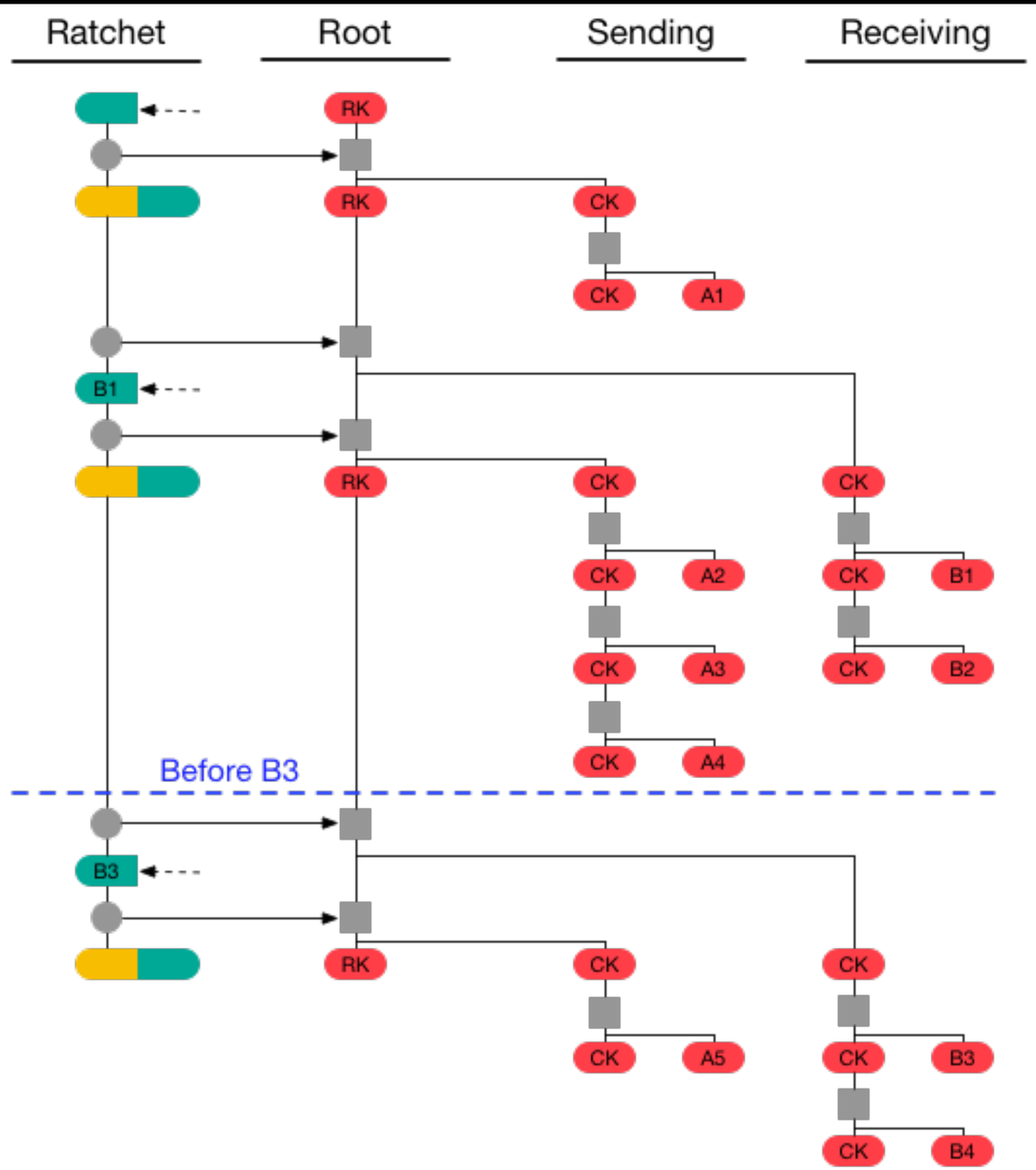
- Symmetric ratchet step is applied to the respective chain every time a message is sent or recieved
- DH ratchet step is only applied when a new public key is received


Ratchet
Root
Sending
Receiving
RK
Old keys can be deleted
RK
CK


Ratchet
Root
Sending
Receiving
RK
RK
Old keys can be deleted
CK
CK
A1
Symmetric-key
ratchet

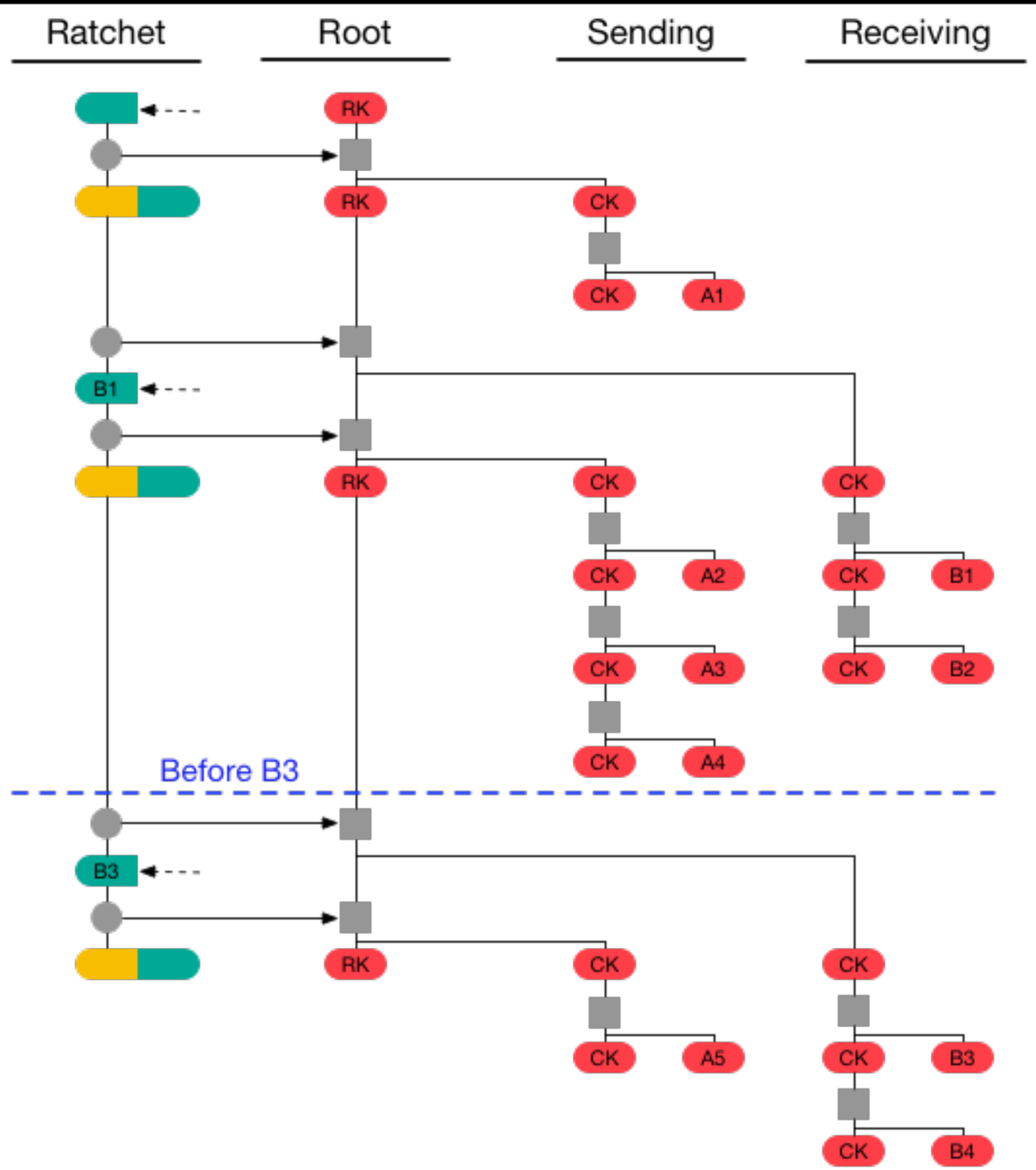

Ratchet
Root
Sending
Receiving
RK
RK
CK
CK
A1
B1
RK
CK
CK
CK
B1
DH ratchet
Symmetric-key ratchet


Ratchet
Root
Sending
Receiving
RK
RK
CK
CK
A1
B1
RK
CK
CK
CK
A2
CK
B1
3 steps
CK
A3
CK
B2
1 step
CK
A4


Ratchet
Root
Sending
Receiving
RK
RK
CK
CK
A1
B1
RK
CK
CK
CK
A2
CK
B1
CK
A3
CK
B2
CK
A4
Before B3
B3
RK
CK
CK
A5
CK
CK
B3
CK
B4

Out of Order Message Handling

- In order to support old messages, message headers include the messages index in the current chain as well as the length of the previous chain
- This lets us know exactly which messages were skipped
- We can hold onto these keys for a while in case those messages show up


Ratchet
Root
Sending
Receiving
RK
RK
CK
CK
A1
B1
RK
CK
CK
CK
A2
CK
B1
CK
A3
CK
B2
CK
A4
Before B3
B3
RK
CK
CK
A5
CK
CK
B3
CK
B4

Additional Signal Protocol Features

- Pre-keys
  - Initial key setup is tricky so client publish keys to the server that can be fetched as necessary
    - Need to be authenticated and have some amount of ephemeral-ness
  - Contact discovery
    - More on this in a future lecture

- Thats it for Protocols
- Next time: Side channels