CMPT 403 – Bonus Material

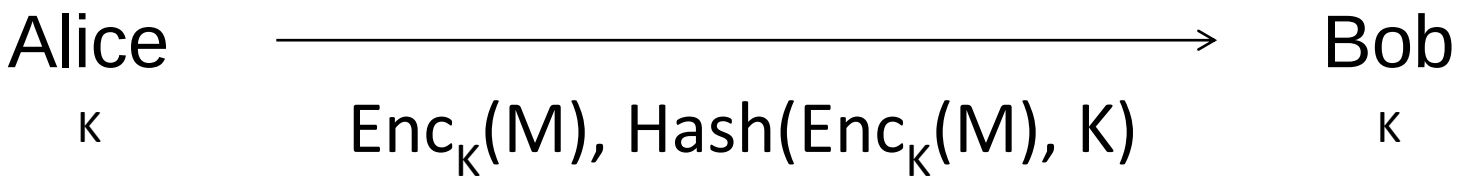
Deniable Messaging

Repudiability

- PKE + PKI allows authentication
- But having our identities provably attached to our message isn't always desirable
  - Connecting identity with behavior compromises privacy
- Repudiability/Deniability: Messages sent in this channel cannot be proven by any other party to have originated from the sender
- Can we design a cryptographic protocol to allow authentication, but also allow repudiability?
  - That is to say, Bob believes Alice is Alice, but Bob cannot prove to anyone else that Alice is Alice

Repudiability

- Consider a SKE setup:



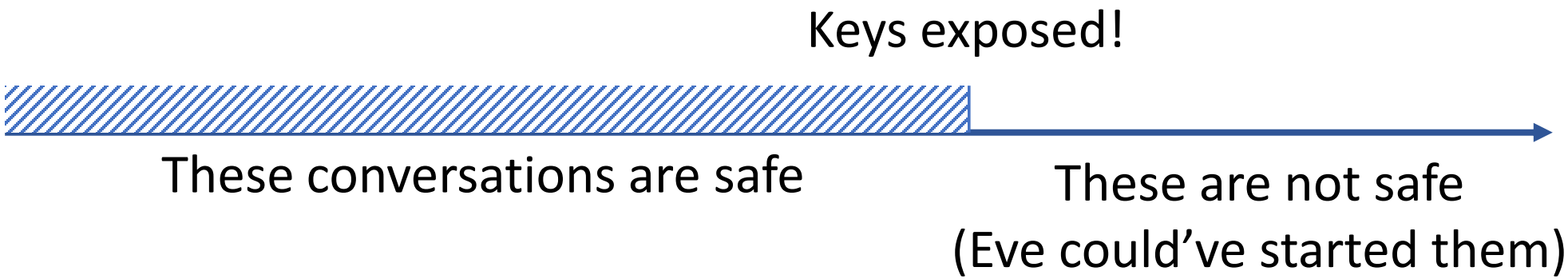
- Bob can check the MAC to ensure that whomever sent this must have the secret key
- Bob knows he himself did not write *M*, so Alice did
- But Bob cannot prove Alice wrote *M* to anyone else, since Bob could've written *M*

Forgeability

- A forgeable ciphertext is a ciphertext that *anyone*, not just Alice or Bob, could have written
  - Even the intercepting attacker could have created this message
- This can be achieved with malleable encryption
  - Recall: Ciphertexts encrypted with malleable encryption can be edited to produce predictable changes in the plaintext
- This can also be achieved by revealing the key

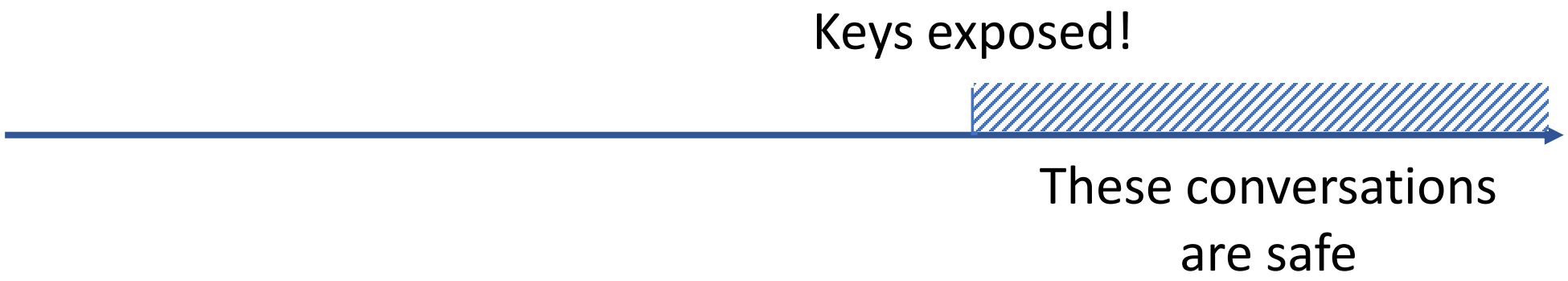
Forward secrecy

- We want to limit damage if keys are exposed
- A (long-term) key in a cryptosystem has **forward secrecy** if leaking that key does not expose *past* conversations
- To achieve this, we ensure that:
  - Long-term keys are only used for signing
  - Encryption is done only with short-term (session) keys



Break-in Recovery

- A cryptosystem has break-in recovery if future conversations after the point of compromise are safe
  - Also known as future secrecy
  - If we only use short-term keys, we have break-in recovery; but we need long-term keys to bootstrap trust



Double Ratchet Algorithm

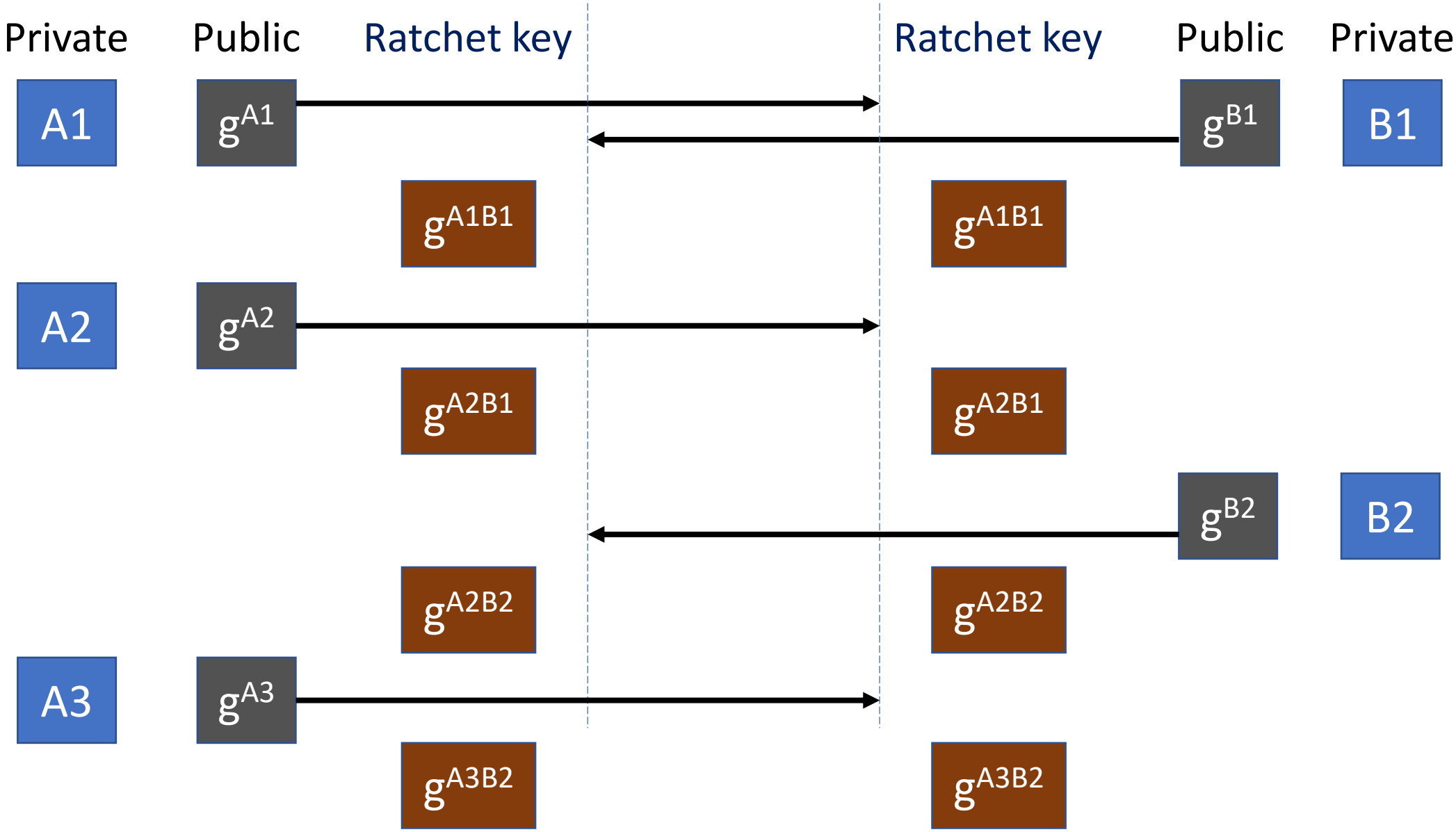
- Used in the Signal Protocol
  - WhatsApp, possibly Facebook Messenger and Skype
- Based on the Off-the-Record Messaging algorithm
- Achieves repudiation, forward secrecy, and break-in recovery
- Based on two sets of ratchets:
  - The **Diffie-Hellman ratchet** generates ratchet keys
  - The **symmetric key ratchet** generates message keys based on ratchet keys
  - A ratchet key can be used to generate several message keys from the same sender

Double Ratchet Algorithm

Diffie-Hellman Ratchet

- Consider DH:
  - Generator g
  - Alice's private key is x, public key is g^x
  - Bob's private key is y, public key is g^y
  - Shared secret becomes g^{xy}
- In the Diffie-Hellman Ratchet, a sequence of shared secrets is generated
- A new shared secret is generated whenever someone who has just received a message wants to send a message
- Ratchet keys will be generated from those shared secrets

Diffie-Hellman Ratchet



Double Ratchet Algorithm

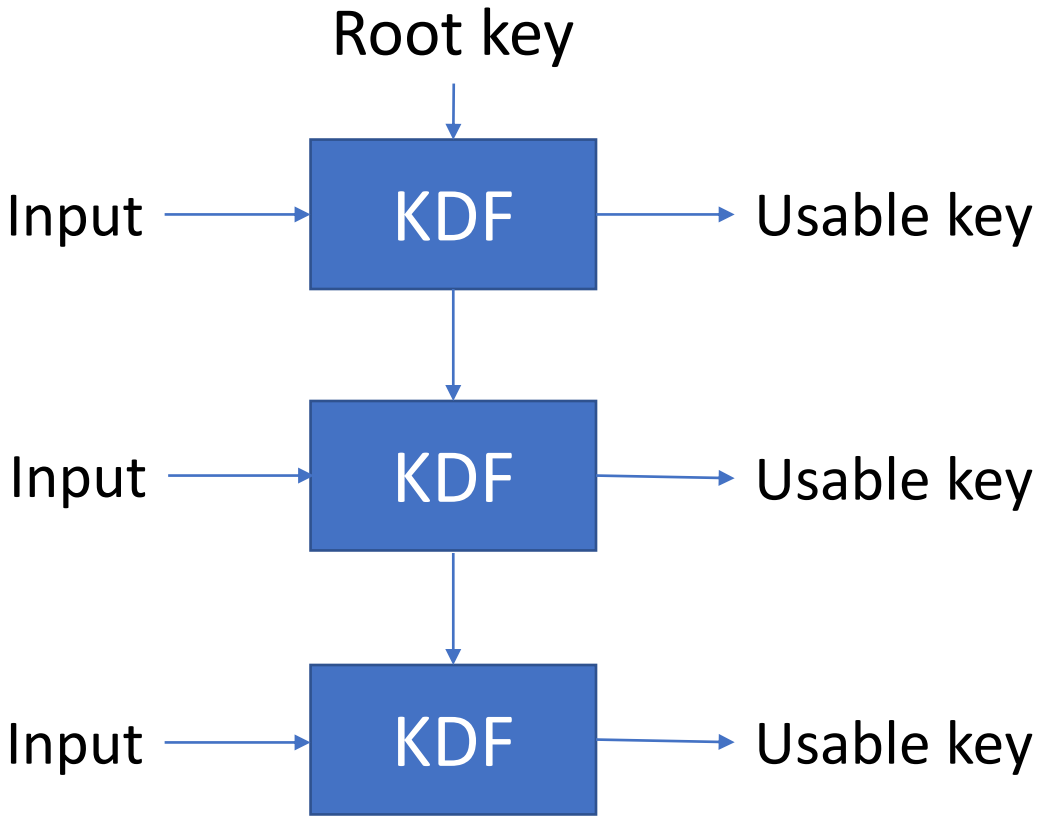
Diffie-Hellman Ratchet

- What happens if a private key is compromised later?
  - Then exactly 2 ratchet keys are compromised
  - If it is B5, then they would be g^{A5B5}, g^{A6B5} (if Alice talks first)
  - No other past or future ratchet keys, or messages depending on those keys, are compromised: **forward secrecy** and **future secrecy**
- The ability to encrypt and create HMACs using the ratchet key would also provide repudiability, as long as we avoid signatures
  - In reality, we refrain from using the ratchet key directly to further reduce the attacker's attack surface

Double Ratchet Algorithm

Symmetric Key Ratchet

Based on Key Derivation Function Chains:

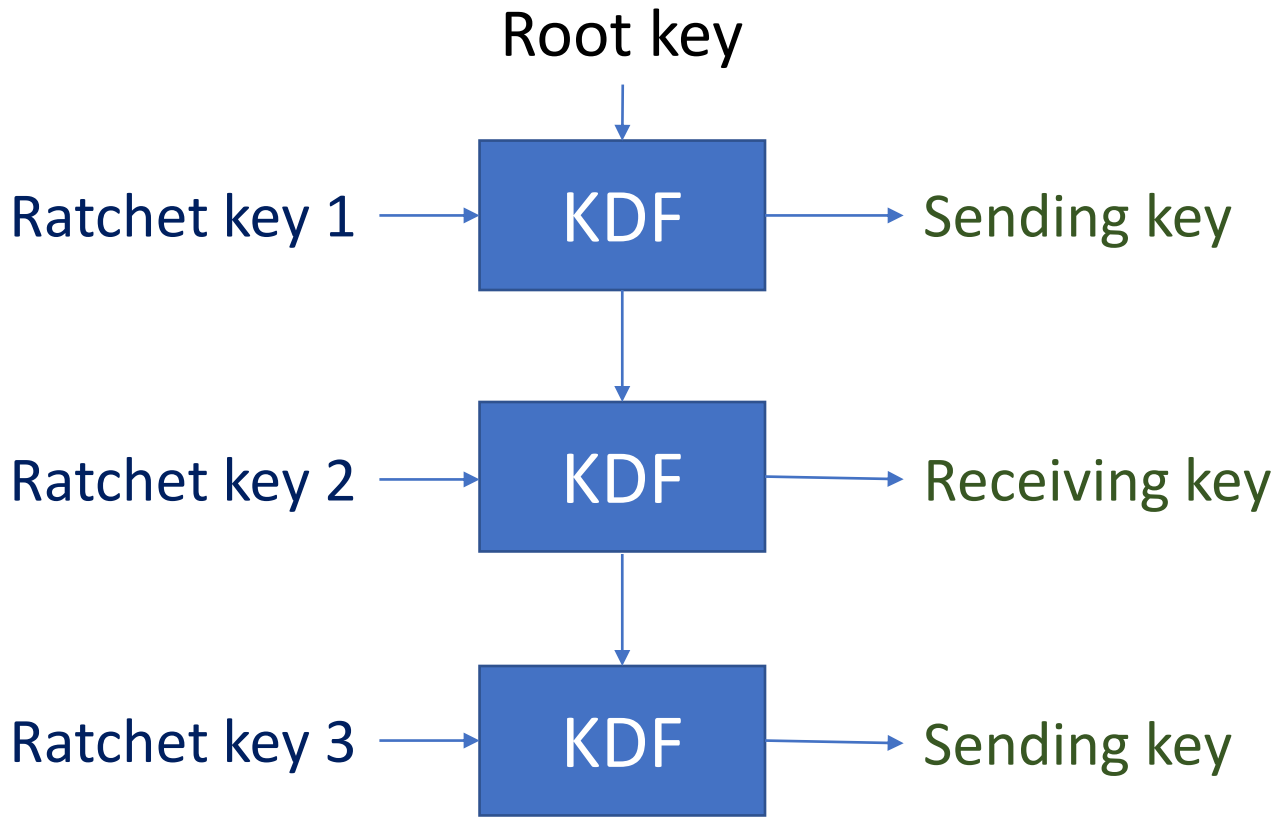


The point is to create usable temporary keys that can be leaked without compromising other keys.

Double Ratchet Algorithm

Symmetric Key Ratchet

First, the ratchet keys produces sending/receiving keys:

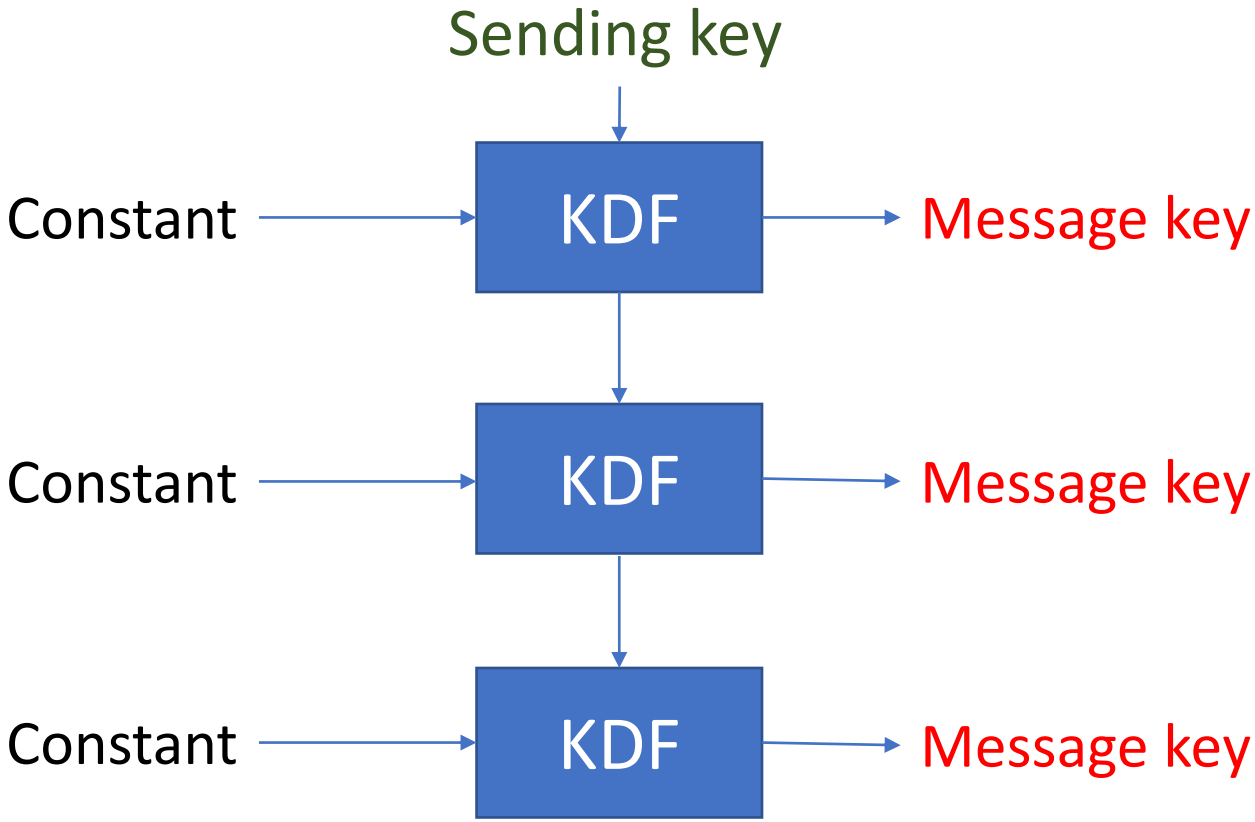


(Alice's side) First ratchet key is Alice's first sending key; Bob's would start with a receiving key

Double Ratchet Algorithm

Symmetric Key Ratchet

Each sending/receiving key starts its own symmetric key KDF chain:



Each message key is used for only one message.

Double Ratchet Algorithm

Review

- KDF chains generates a series of keys, each key based on the previous root key and an input
- The DH ratchet generates and procedurally updates ratchet keys
  - A new chain is started whenever one side switches from receiving to sending
- The ratchet keys are used as input to the DH KDF chain to generate sending and receiving chain keys
- Chain keys are used as the bootstrapping root key for symmetric key DF chains to generate message keys

Double Ratchet Algorithm

- Benefits of using two ratchets:
  - Each message key can be deleted after one use; sending/receiving keys can be deleted after all relevant messages are sent/received
  - Handling of out-of-order/dropped messages is possible
  - Limits compromise of messages from key leakage
    - Sending/receiving keys can compromise multiple messages
    - Ratchet key plus a previous root key for the same
    - Each message key can leak one message

Double Ratchet Algorithm

- Can we also achieve forgeability?
  - Possibly, by releasing MAC keys (not decryption keys)
- A message participant can still *collude* with an outsider to prove messages sent by the other participant are real
  - It is possible to resolve this problem (“strong deniability”)
- This does not work for group messaging
  - The property that an HMAC indirectly proves identity does not follow for group messaging