Three desirable properties for private messaging

- **Repudiability**: I can deny that a message is written by me; no one can prove to a third party that it is written by me
  - How can this co-exist with message authenticity?
- **Forward secrecy:** If I leak my keys, conversations before the leakage time are still secure
  - This was achieved with short-term encryption keys
- **Break-in recovery:** If I leak my keys, conversation after the leakage time are still secure
  - This cannot be achieved with the above setup; it is broken by the signature scheme bootstrapping process

Double Ratchet Algorithm

- Used in the Signal Protocol
  - WhatsApp, Telegram, Facebook Messenger, 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

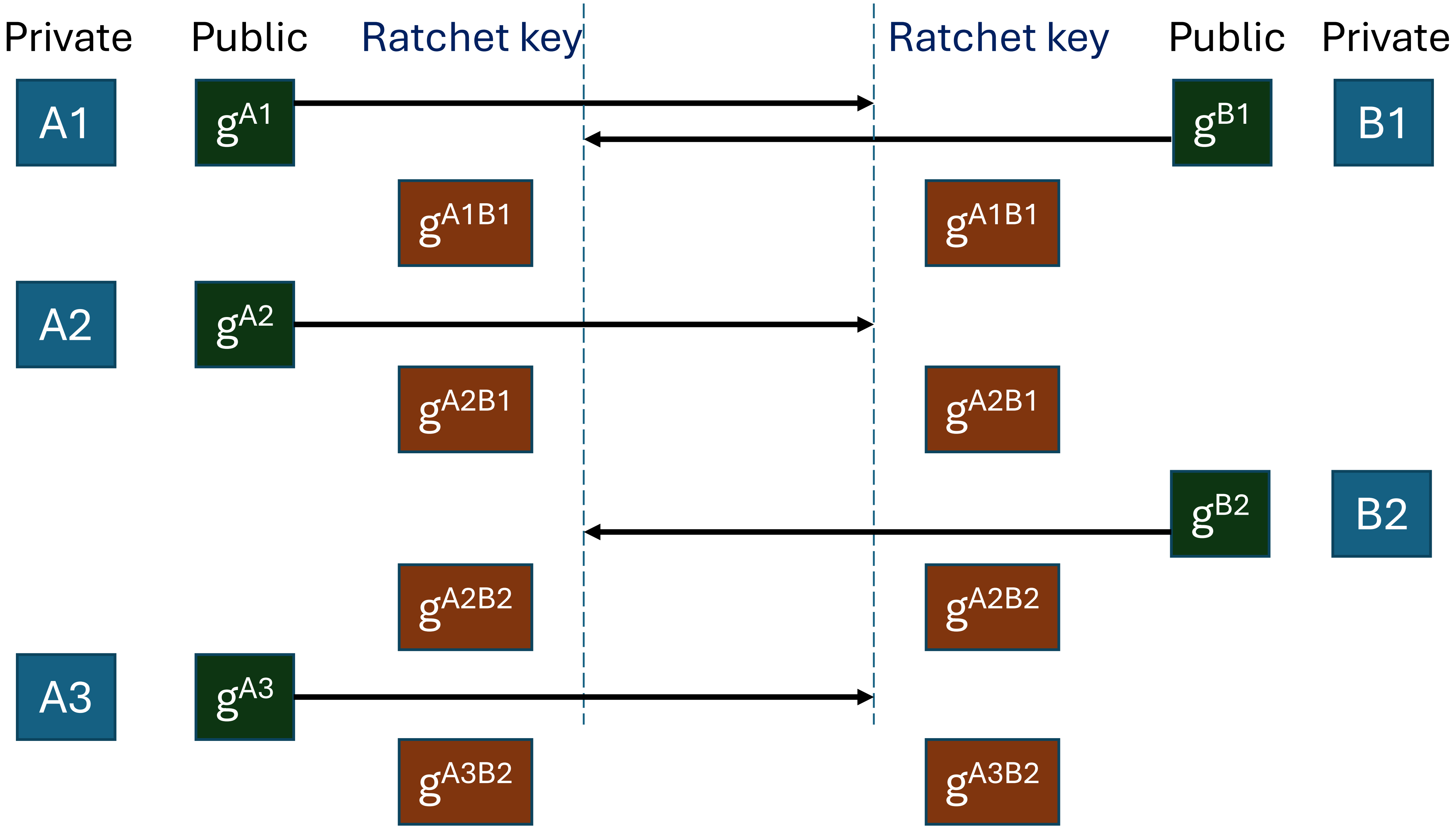
<u>Diffie-Hellman Ratchet</u>

- 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

Alice

Diffie-Hellman Ratchet

Bob



Double Ratchet Algorithm

Diffie-Hellman Ratchet

- We now have **key update** without the need of long-term keys
  - Only the first exchange is signed with identity keys
- 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)
- **Forward secrecy**: Conversations using previous keys are not compromised
- **Break-in recovery**: Conversations using future keys are not compromised

Repudiability

- Consider a SKE setup:

Alice → Bob

K $\quad \text{Enc}_K(M), \text{Hash}(\text{Enc}_K(M), K) \quad$ K

- 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*
- *The important thing is to avoid signatures*
- Diffie-Hellman Ratchet achieves repudiability by using only a secret key to send messages and HMACs

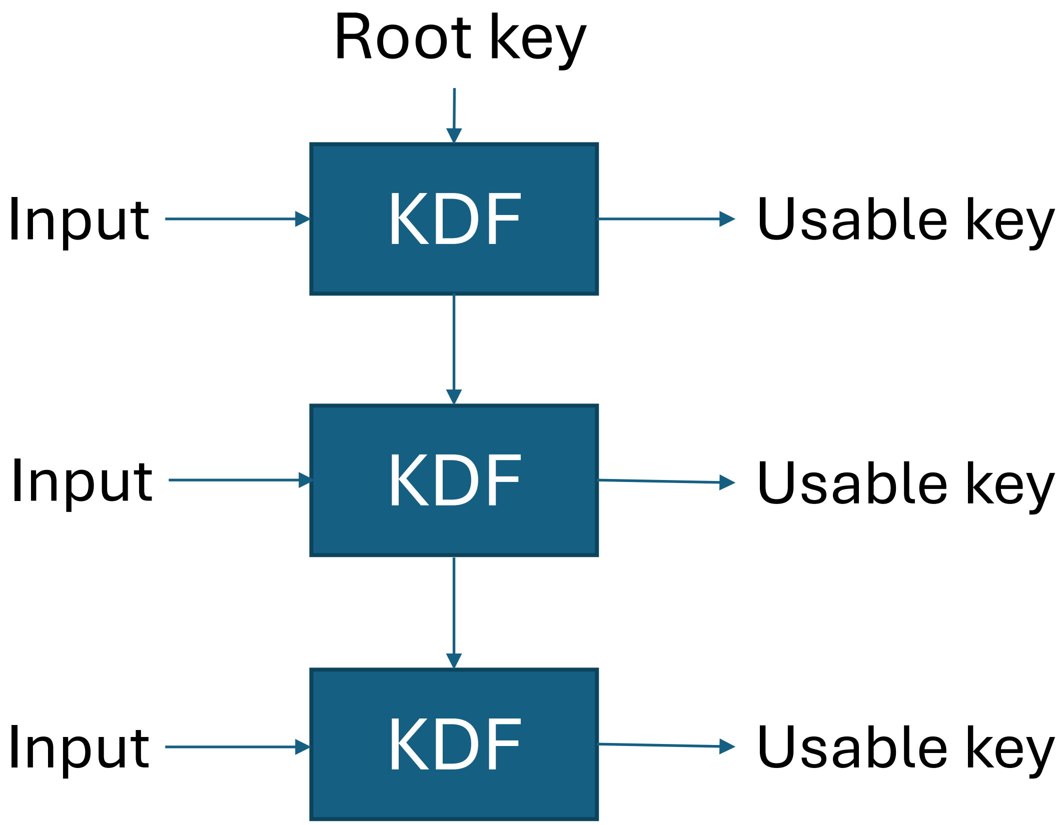
A remaining weakness

- In practice, message can be lost or re-ordered
- This means we cannot keep advancing ratchet keys – we need to store old keys for an indefinite time
- To solve this, we use a second ratchet, known as the *symmetric key ratchet*

Double Ratchet Algorithm

Symmetric Key Ratchet

Based on Key Derivation Function Chains:



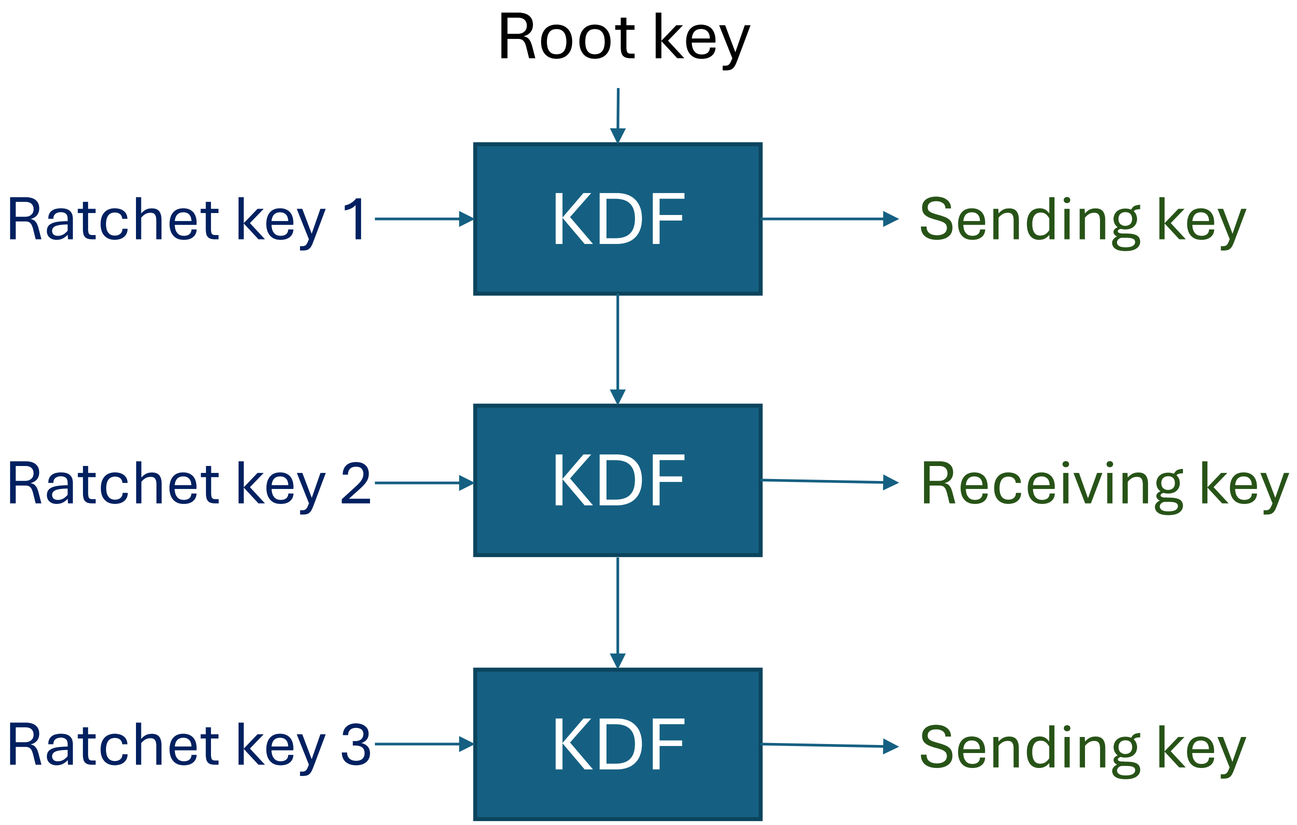
e.g. h(Input1 || Input2) = (Usable key || Output1)

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

Double Ratchet Algorithm

<u>Symmetric Key Ratchet</u>

First, the ratchet keys produces sending/receiving keys:

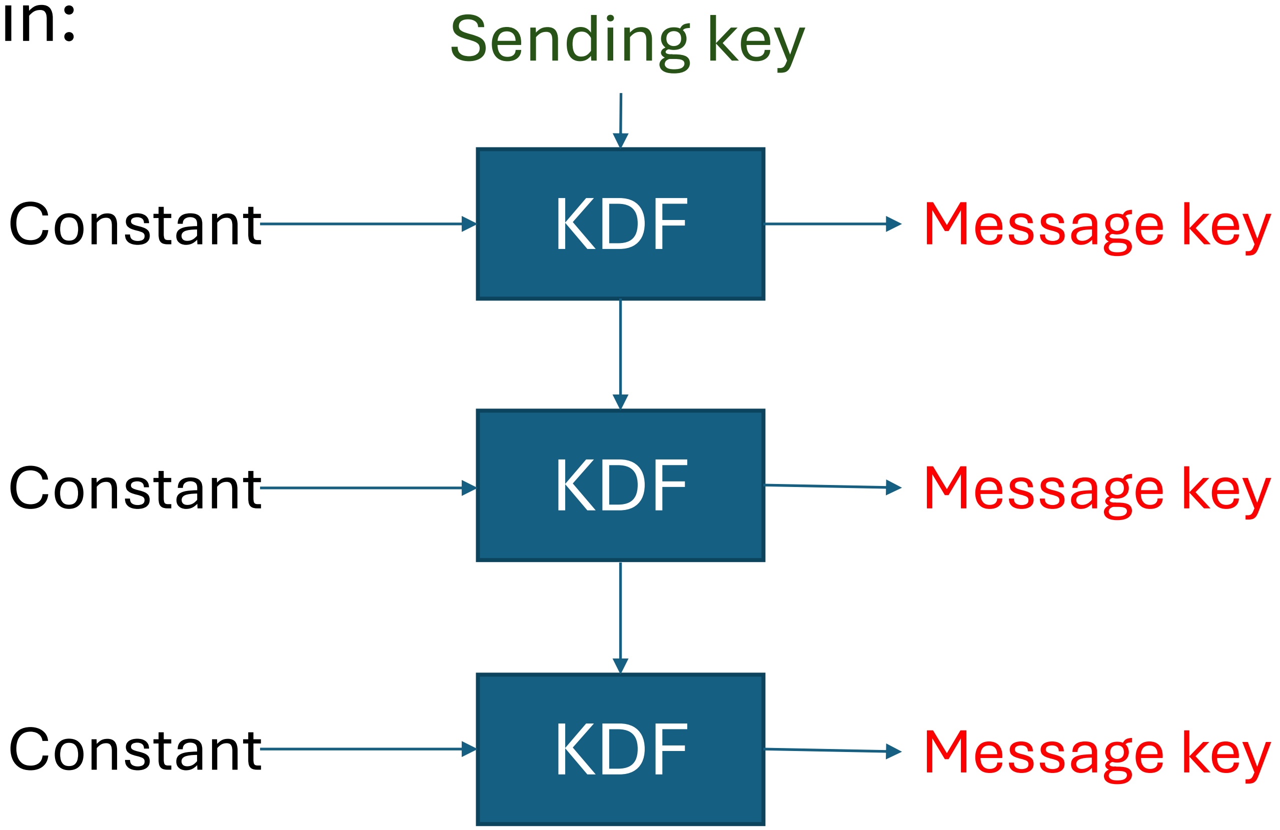


(Alice's side) First ratchet key is Alice's first sending key; Bob's would start with a receiving key

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.

Message keys can now be stored (and potentially leaked) without affecting security.

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

- Stronger property than repudiability: forgeability
  - *Anyone* could have created the message, not just Alice and Bob
- Can we also achieve forgeability?
  - Possibly, by releasing MAC keys (not decryption keys)
- This does not work for group messaging
  - The property that an HMAC indirectly proves identity does not follow for group messaging