Pond – a non-instant messaging protocol by Adam Langley

Chitchanok Chuengsatiansup

Technische Universiteit Eindhoven

Pond

- Forward-secure asynchronous messaging (not email!)
- Communicate with authorized users (no spam!)
  - authorization done using PANDA key exchange
- Attempt to resist network traffic observation
  - connections made at random intervals (dummy or real send/fetch)
  - messages padded to a fixed length
- More details see https://pond.imperialviolet.org/

How Pond Works

- Servers
  - work as a "cut-out"
    - never make connections
    - no one controls servers
  - provide availability
    - users need not to be online simultaneously

How Pond Works

- Servers
  - work as a "cut-out"
    - never make connections
    - no one controls servers
  - provide availability
    - users need not to be online simultaneously
- Users:
  - have 3 types of connection
    - key exchange: PANDA server (over Tor)
    - send/receive message: Tor server
    - large file transfer: Tor server
  - communicate only with servers
    - own server to receive messages or upload files
    - recipient's server to send messages
    - sender's server to download files
  - make connections periodically
    - time between each connection is exponentially distributed

Messages

- Stored on server until they are fetched
- Auto-erased from users' side after a fixed amount of time (currently 1 week)
- Padded to a fixed size
- Attachment also allowed
  - small: included in a message
  - large: sent many small pieces or uploaded to server
- Encouraged to be acknowledged
  - reply to a message
  - send an acknowledgment
- Server does not learn who the sender is

Crypto Components

- TOR
- TLS (for PANDA key exchange)
- Curve25519, Ed25519, Salsa20, Poly1305, HMAC-SHA256, Rijndael (with 256-bit block)
- BBS: group signature
- DH ratchet: encryption
  https://github.com/trevp/axolotl/wiki
  https://github.com/agl/pond/blob/master/client/ratchet/
  - forward secure through symmetric-key updating
  - future secure through DH ratchet key updating
- All assumed to be good
- Implemented in GO by Adam Langley

DH Ratchet: keys

- consists of sender and receiver version
- generated using HMAC-SHA256 from DH key

DH Ratchet: keys

- consists of sender and receiver version
- generated using HMAC-SHA256 from DH key

- DH-ratchet key: (A_i, B_j)
  - generated unrelatedly from previous key
- chain key: $CK_{(A_i,B_j)}$
  - derived from DH-ratchet key
  - used for forward-secrecy updating
- message key: $MK_{(A_i,B_j)}$
  - derived from chain key
  - newly generated per message

DH Ratchet: keys

- consists of sender and receiver version
- generated using HMAC-SHA256 from DH key

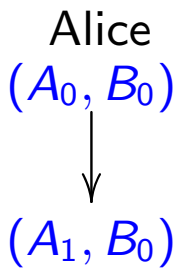
- DH-ratchet key: (A_i, B_j)
  - generated unrelatedly from previous key
- chain key: $CK_{(A_i,B_j)}$
  - derived from DH-ratchet key
  - used for forward-secrecy updating
- message key: $MK_{(A_i,B_j)}$
  - derived from chain key
  - newly generated per message
- header key, next header key
  - used to encrypt DH-ratchet key sent to receiver

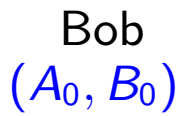
DH Ratchet: flow

Alice
(A_0, B_0)

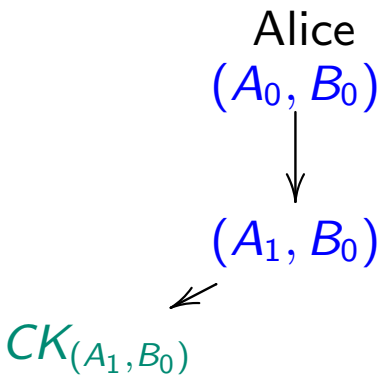
Bob
(A_0, B_0)

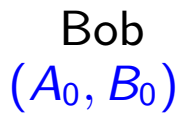
DH Ratchet: flow



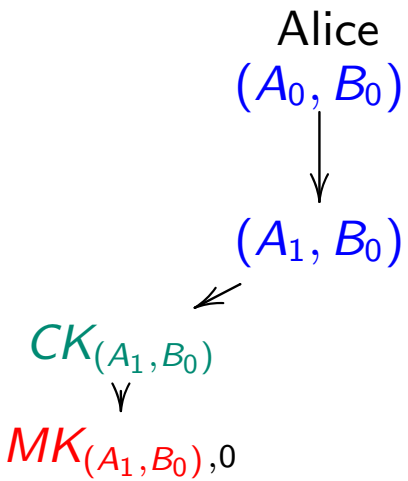


DH Ratchet: flow



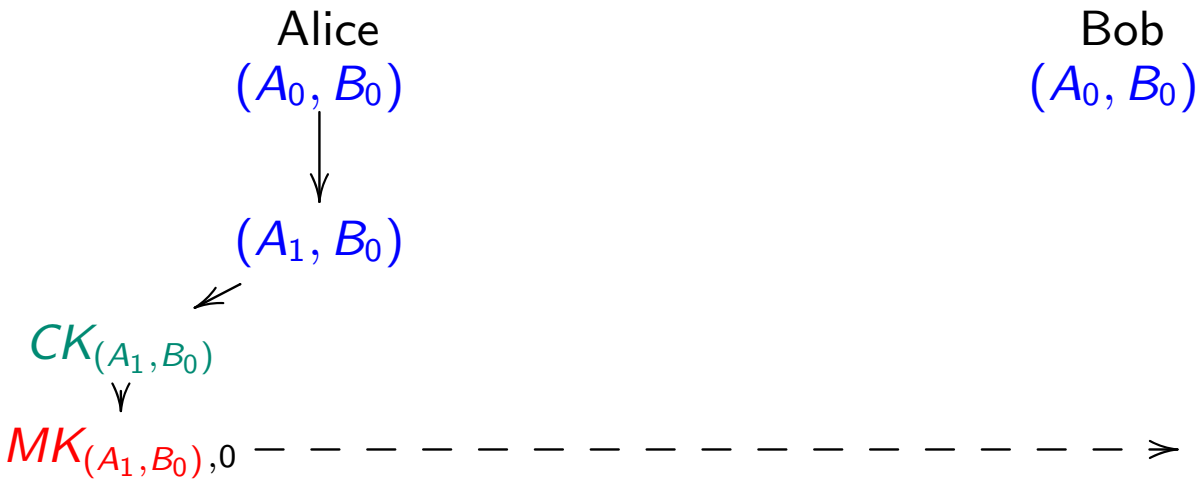


DH Ratchet: flow

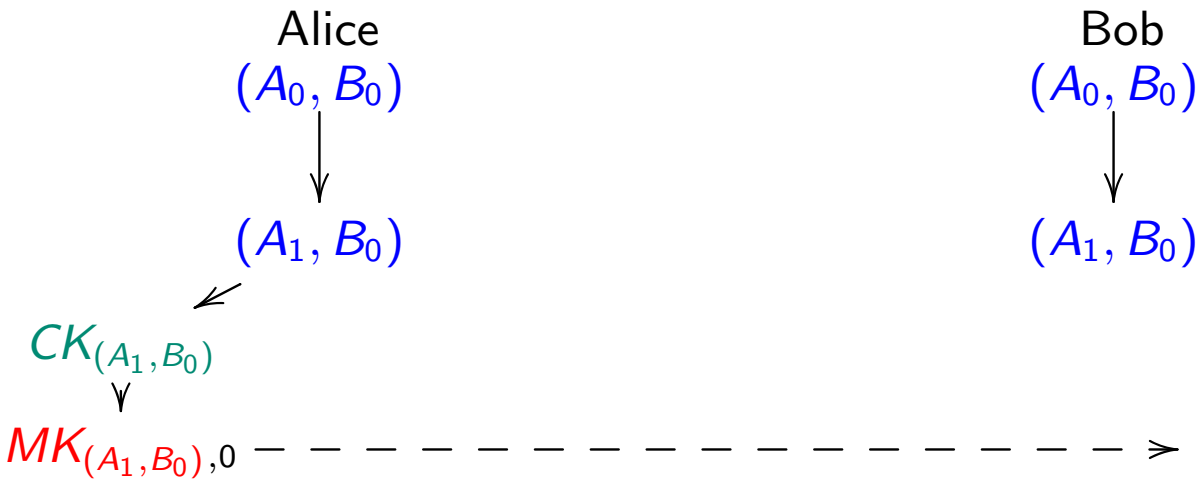


Bob
(A_0, B_0)

DH Ratchet: flow



DH Ratchet: flow

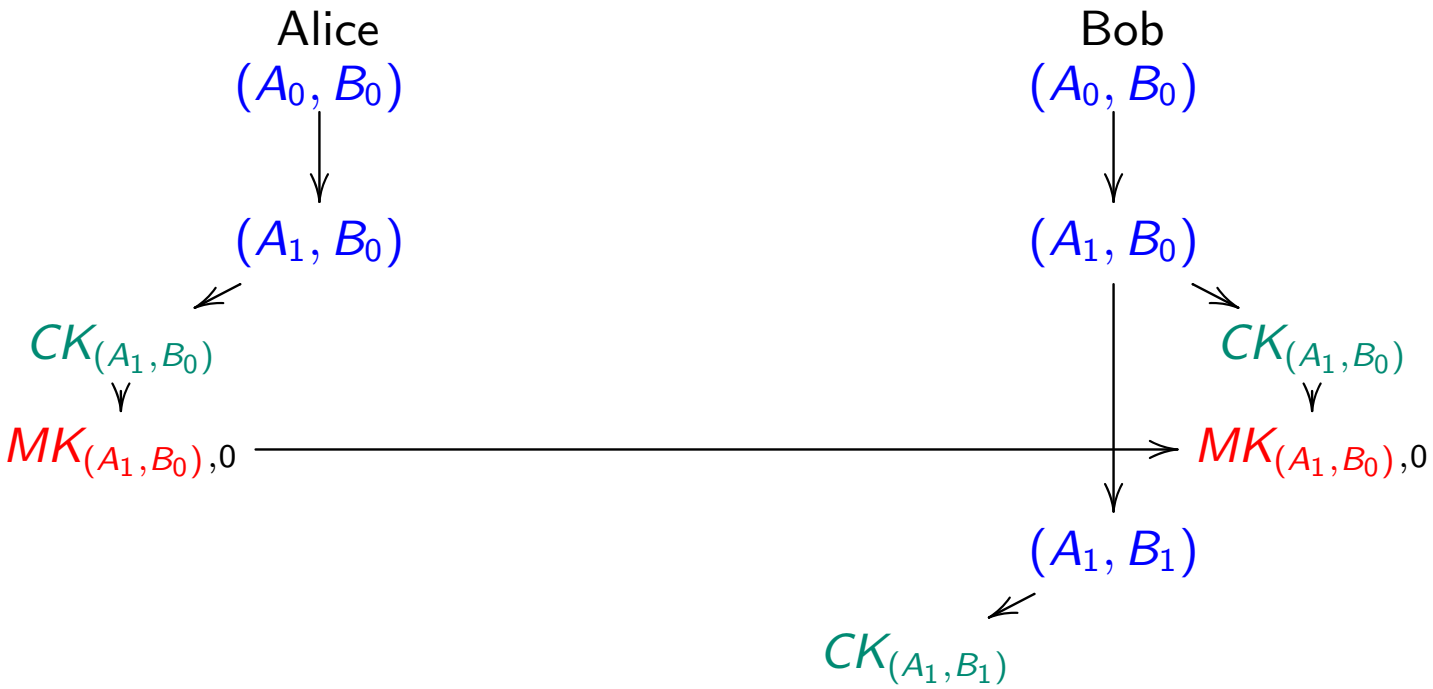


DH Ratchet: flow

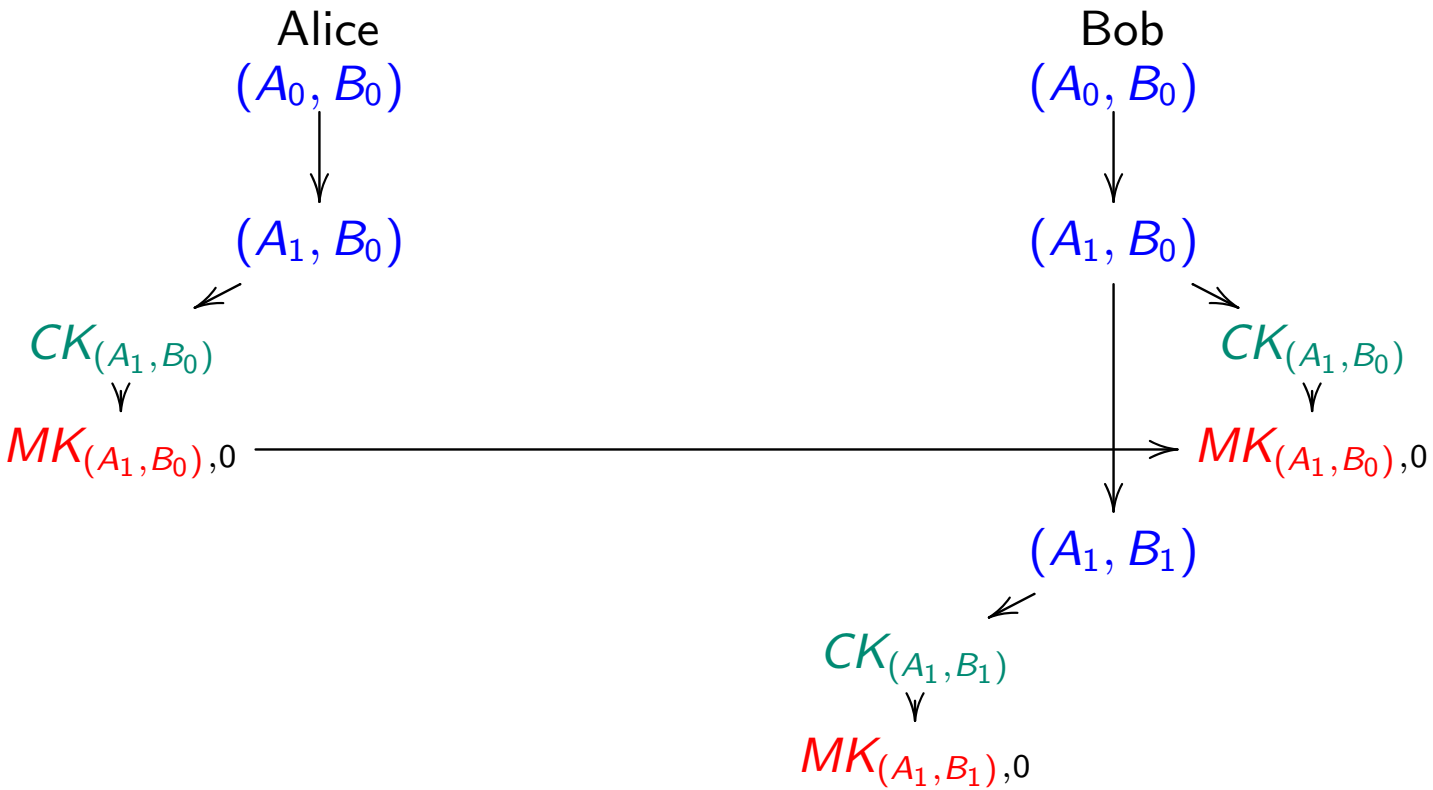
DH Ratchet: flow

DH Ratchet: flow

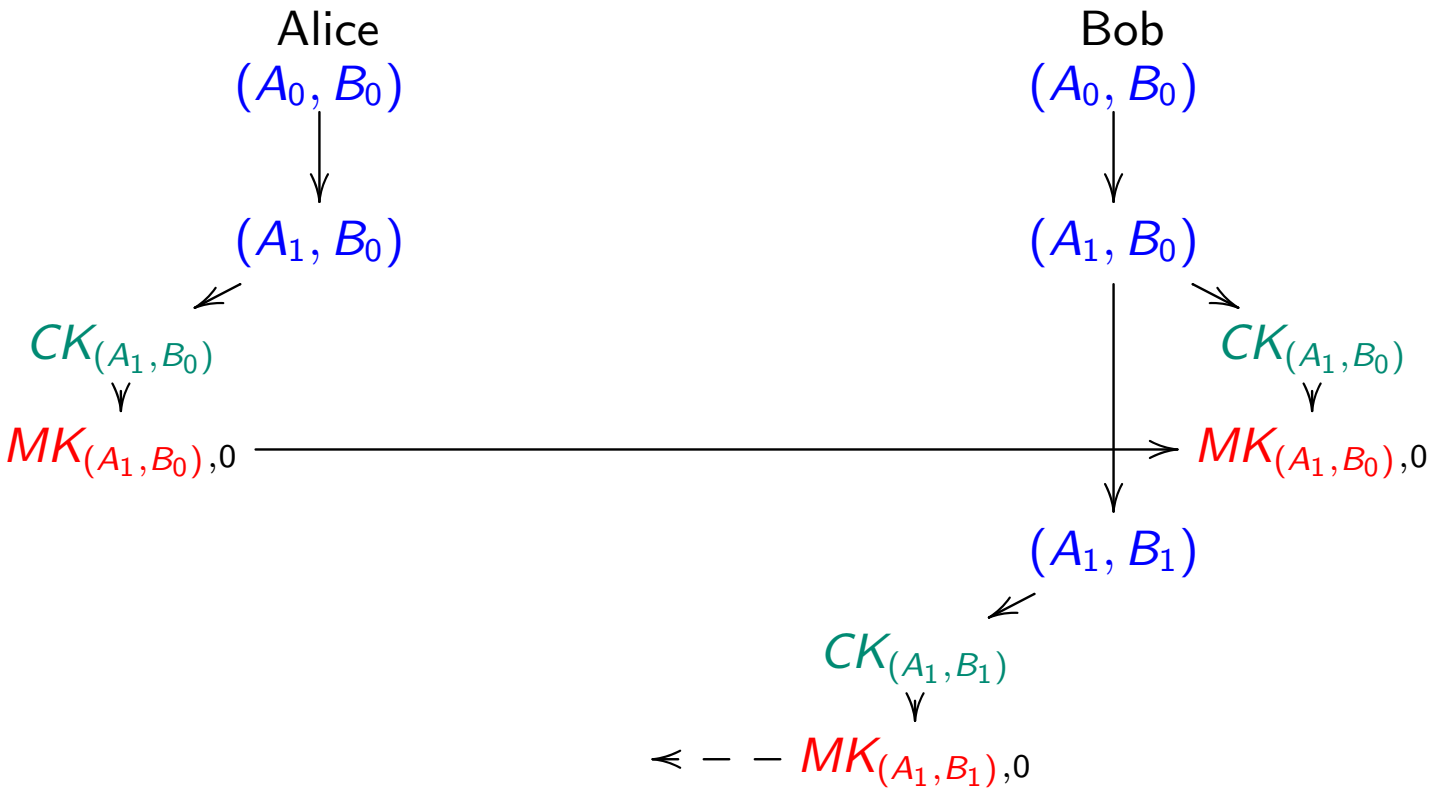
DH Ratchet: flow



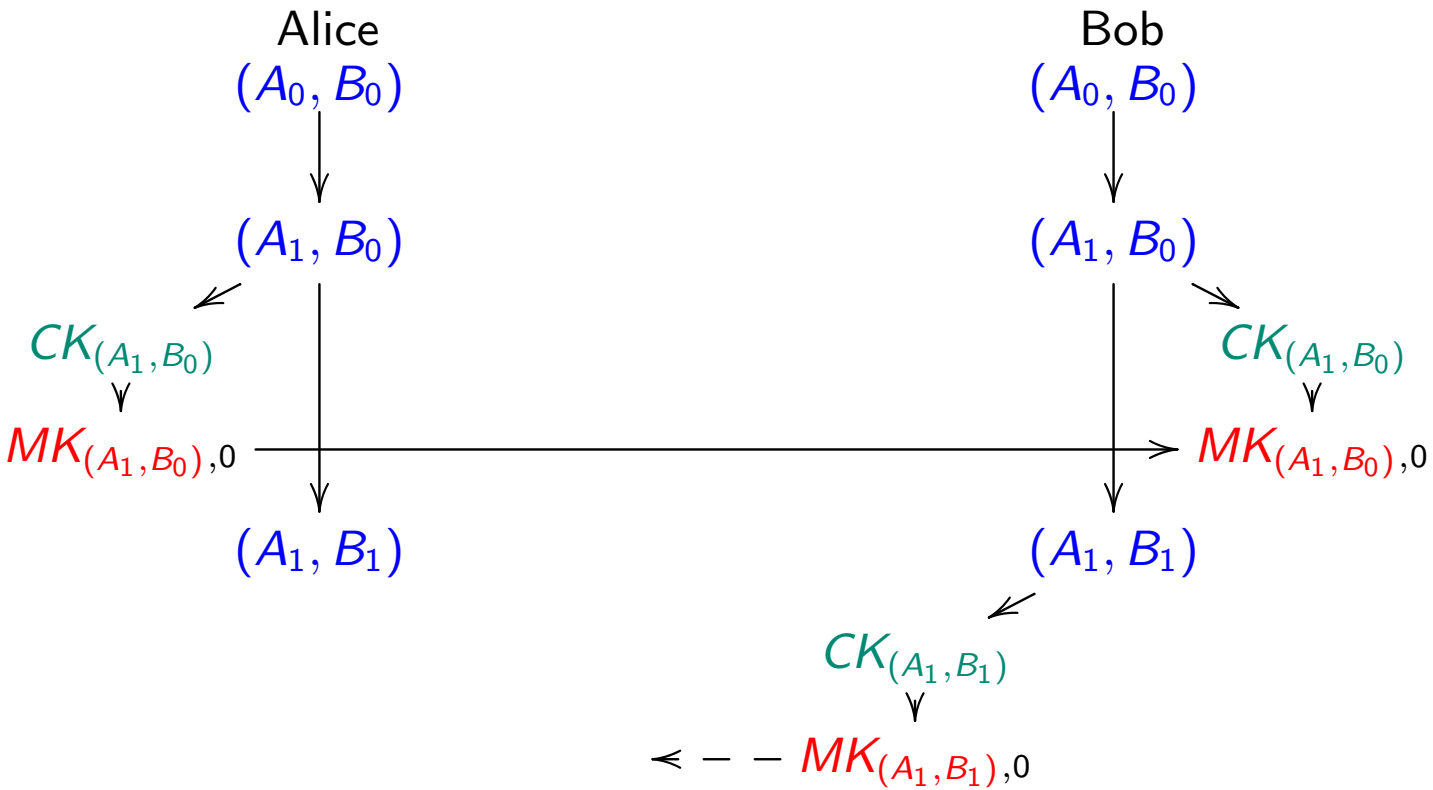
DH Ratchet: flow



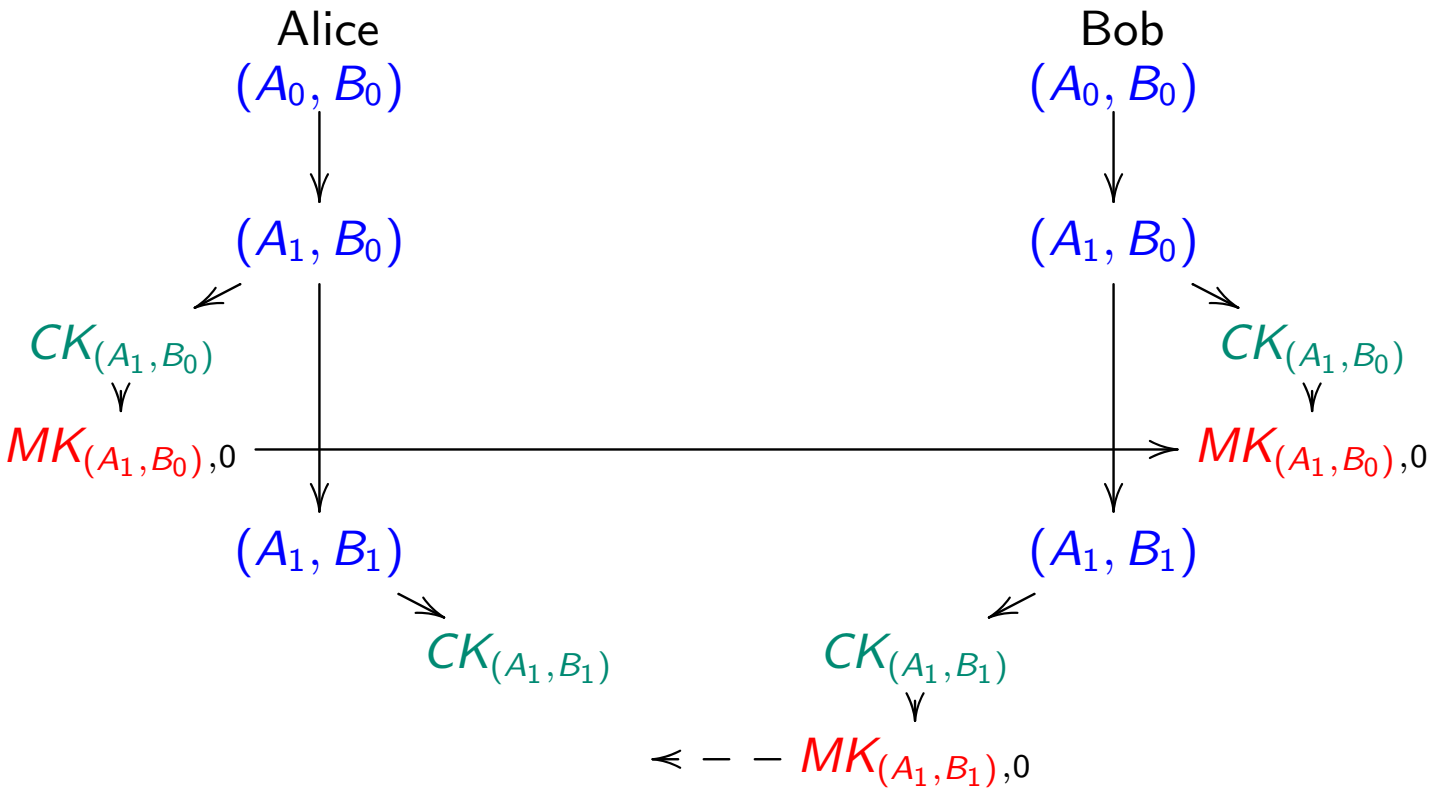
DH Ratchet: flow



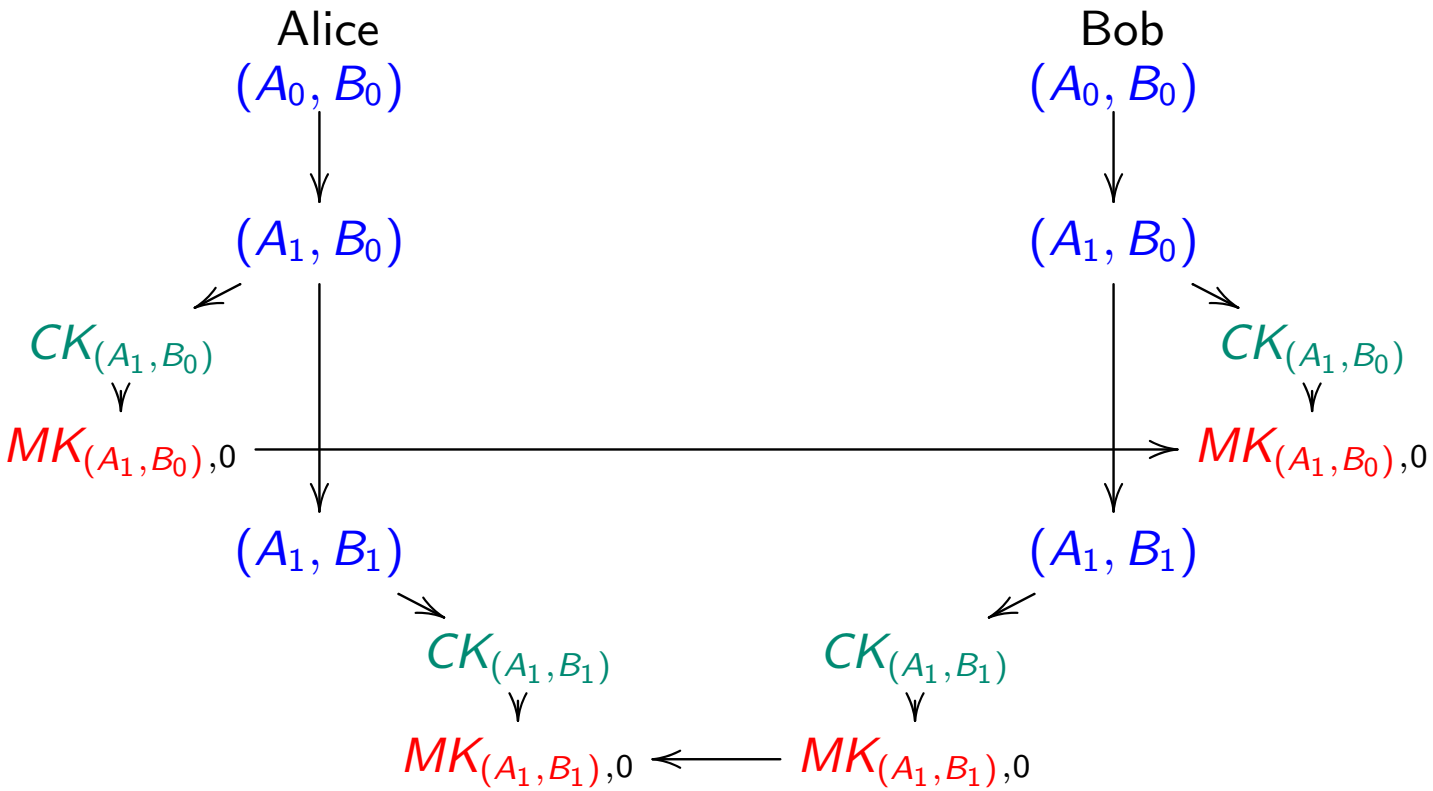
DH Ratchet: flow



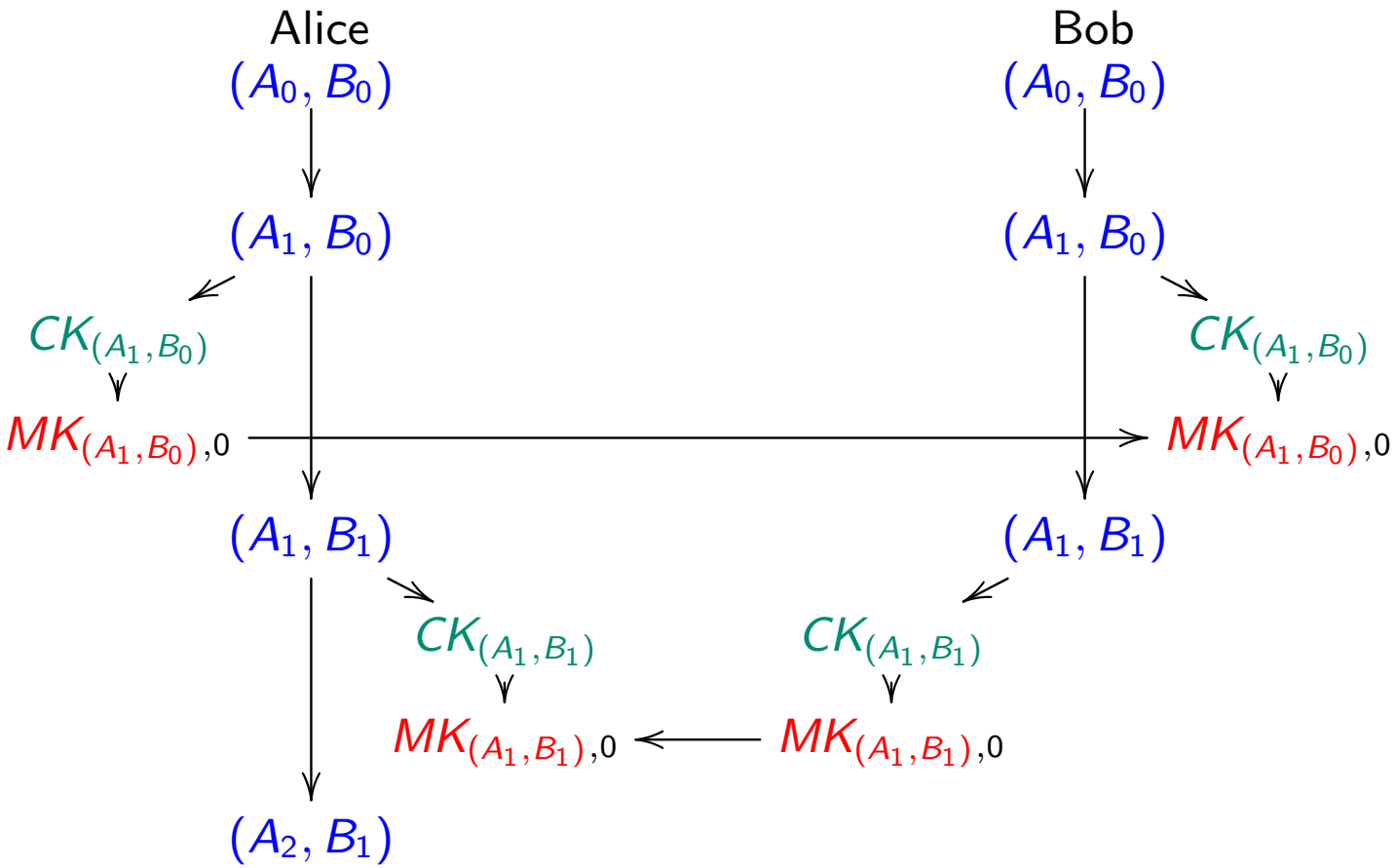
DH Ratchet: flow



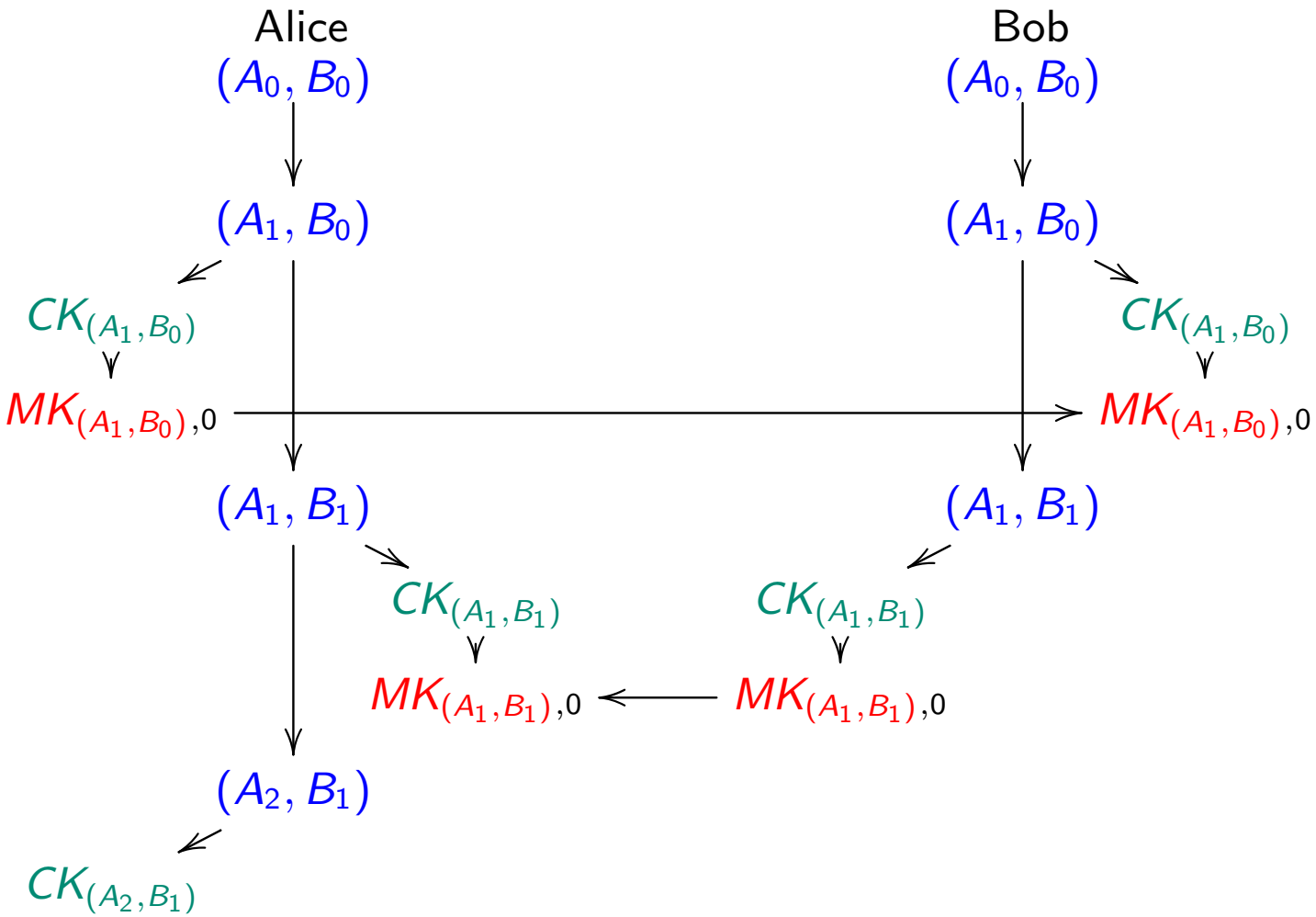
DH Ratchet: flow



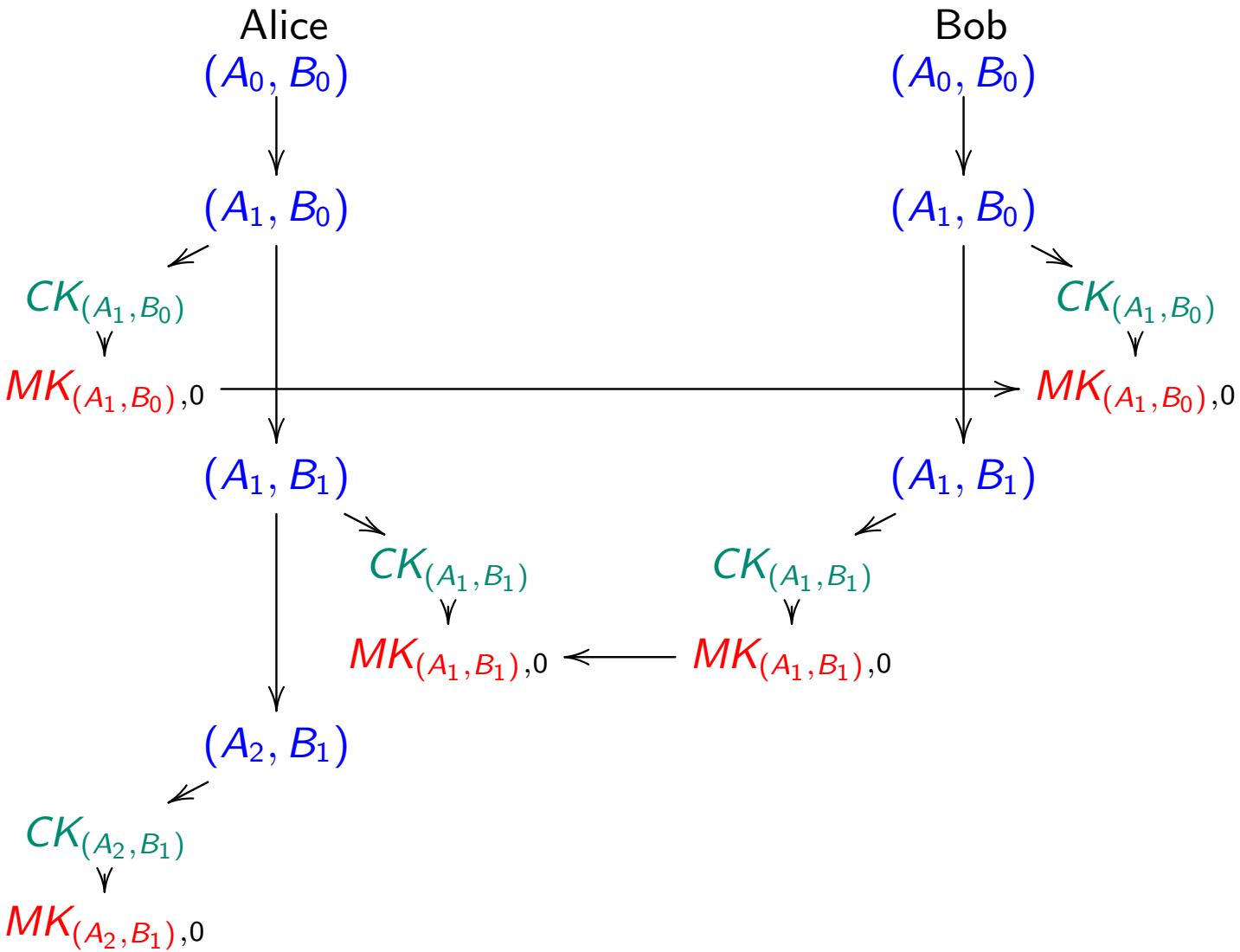
DH Ratchet: flow



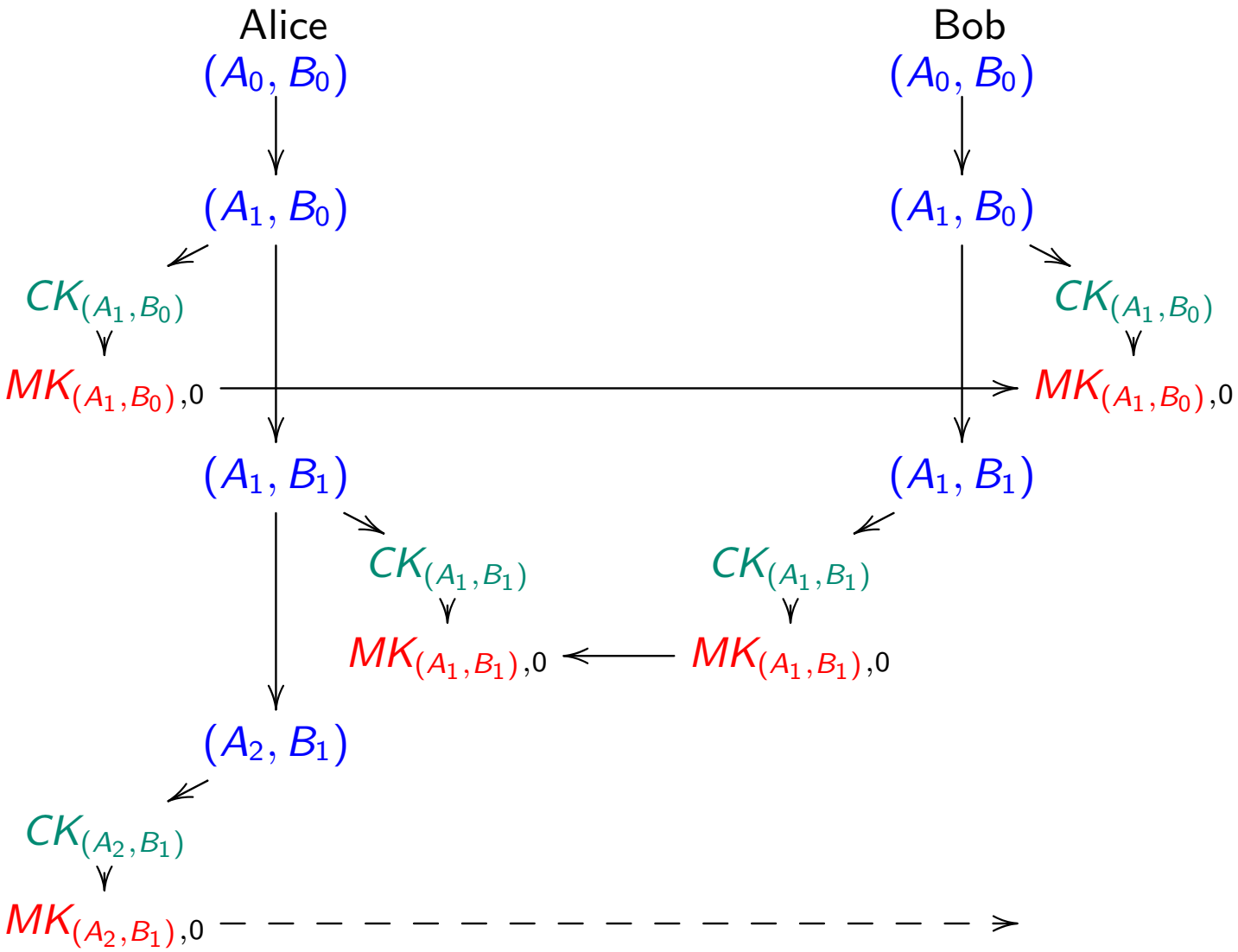
DH Ratchet: flow



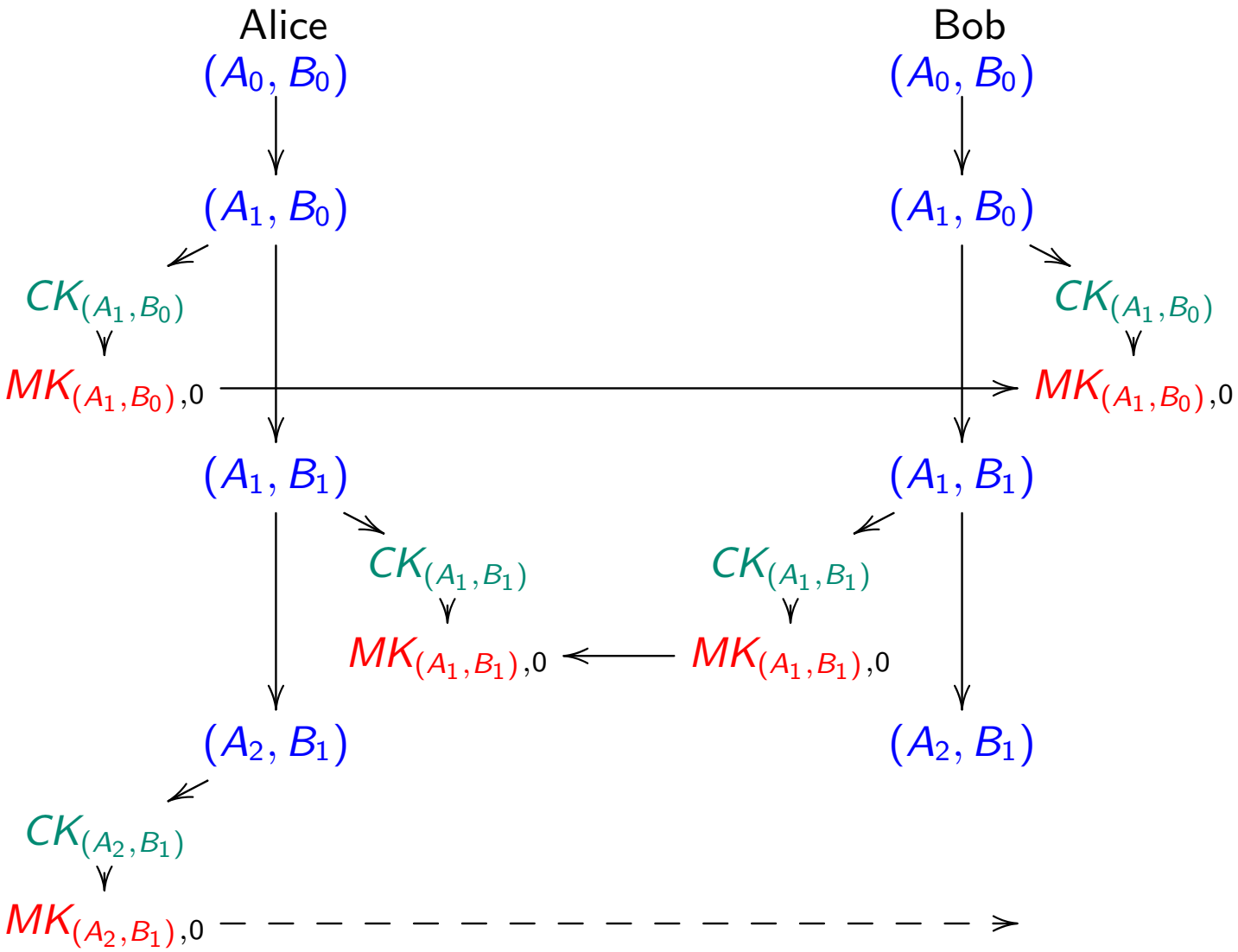
DH Ratchet: flow



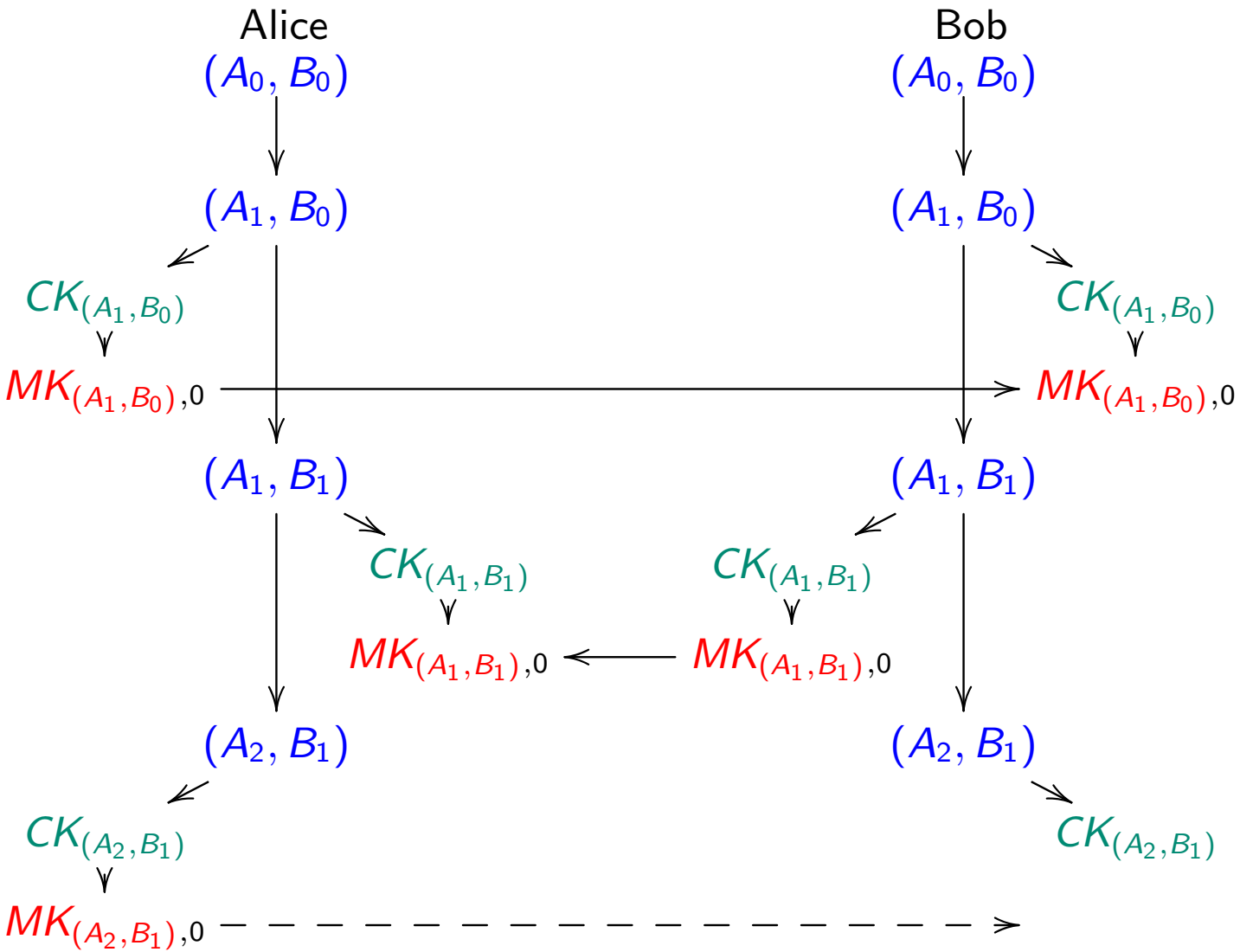
DH Ratchet: flow



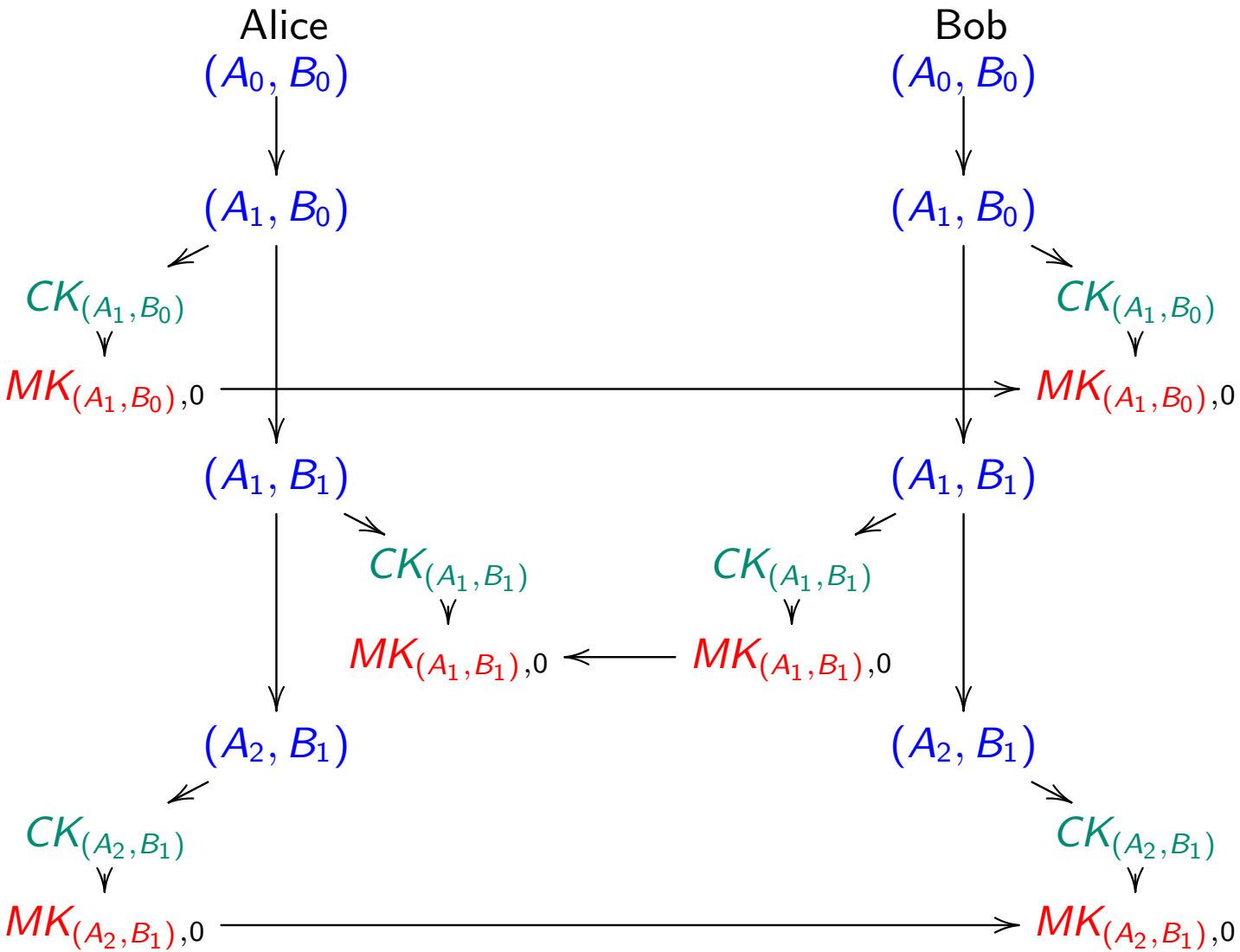
DH Ratchet: flow



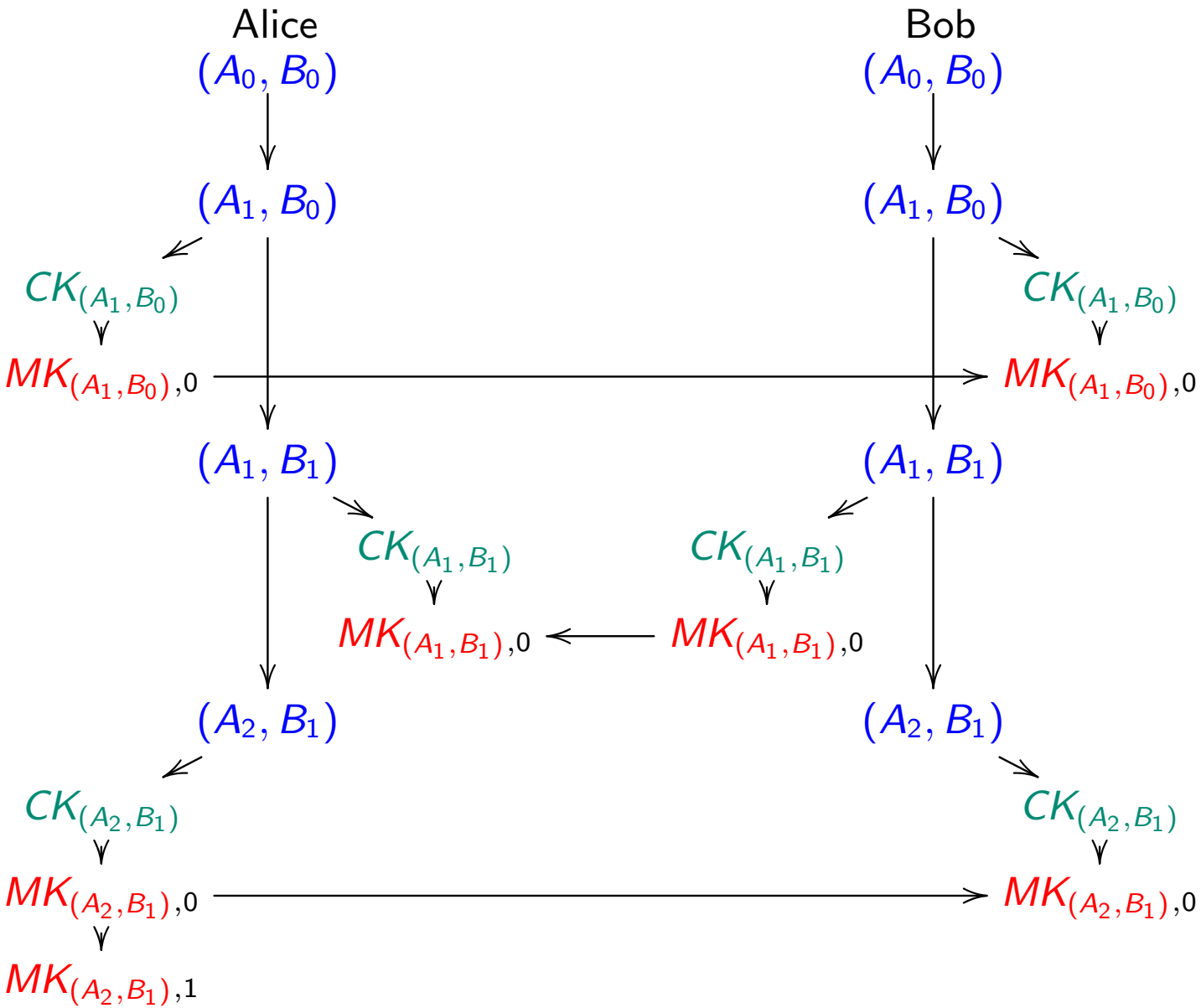
DH Ratchet: flow



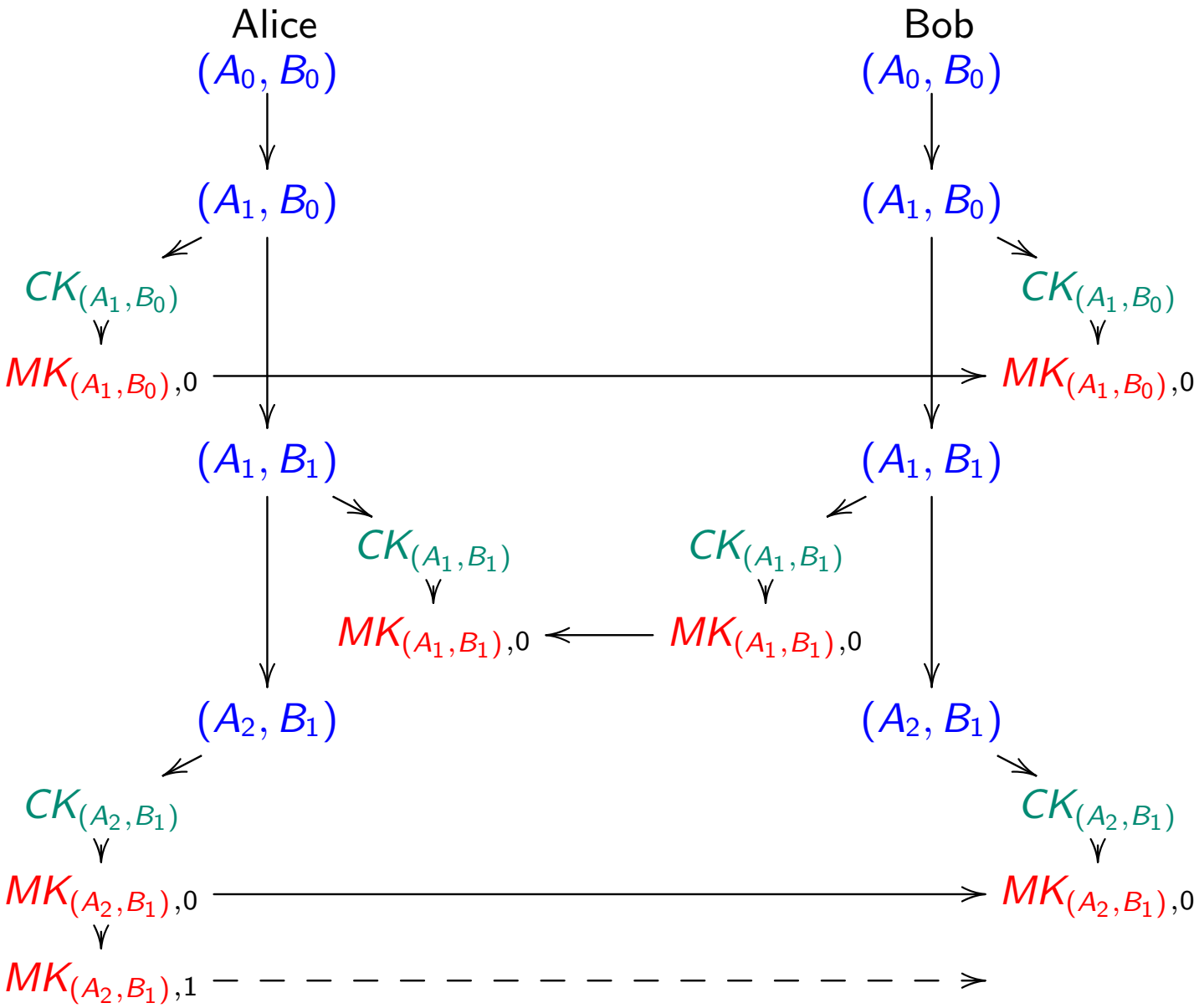
DH Ratchet: flow



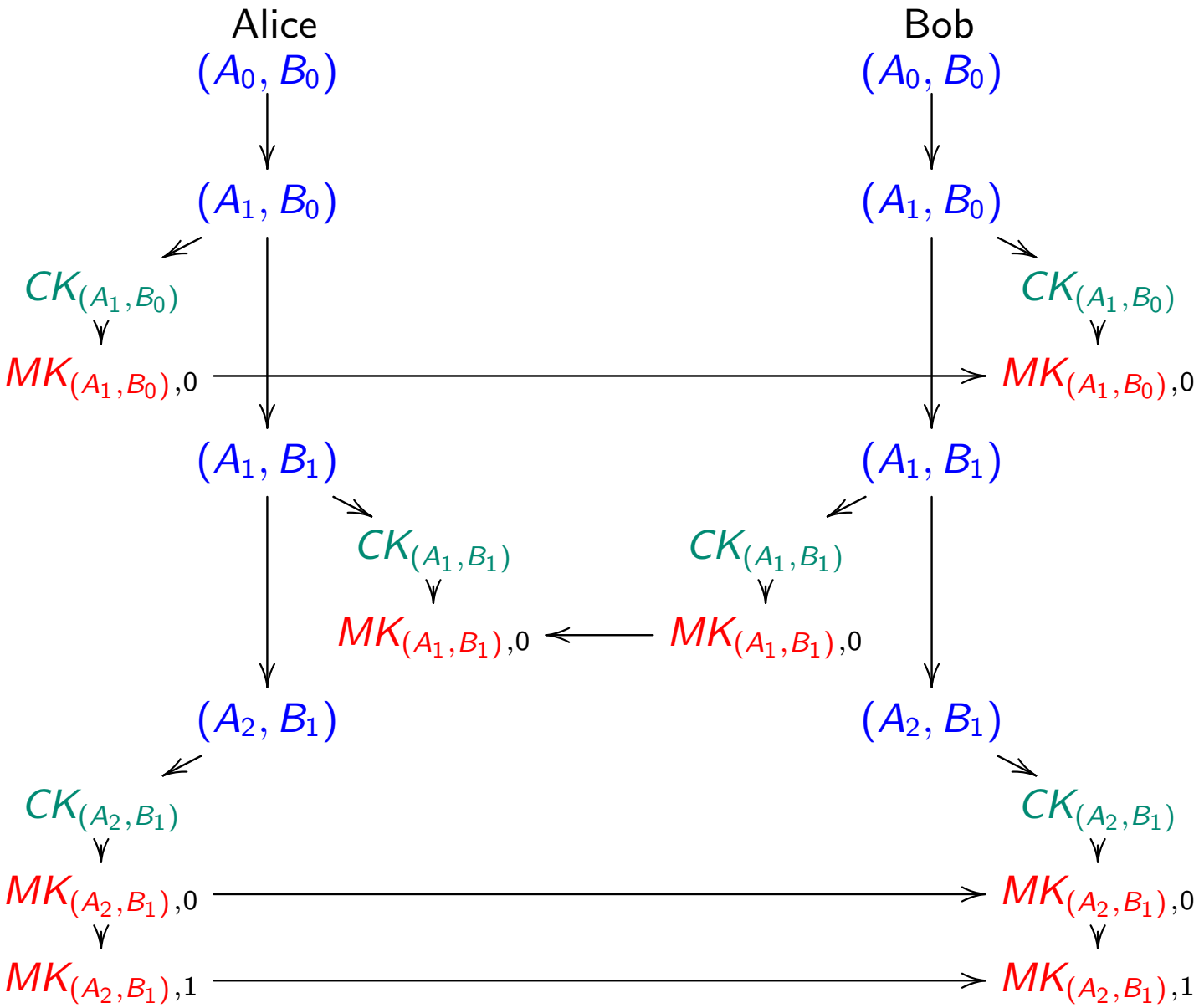
DH Ratchet: flow



DH Ratchet: flow



DH Ratchet: flow



Network Diagram

Use Tor
at all time

Server1

User1

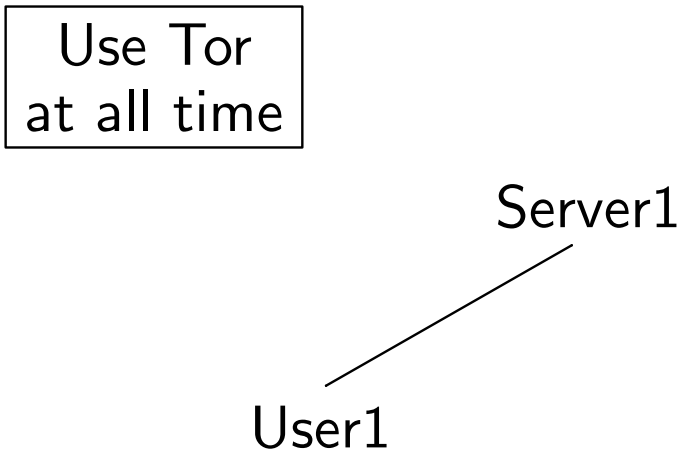
$G_{i\ pub}$: Group i public key

$G_{i\ sk[j]}$: Group i secret key of member j

$S_{i\ pub}$: Server i public key

$U_{i\ serv}$: User i home server

Network Diagram



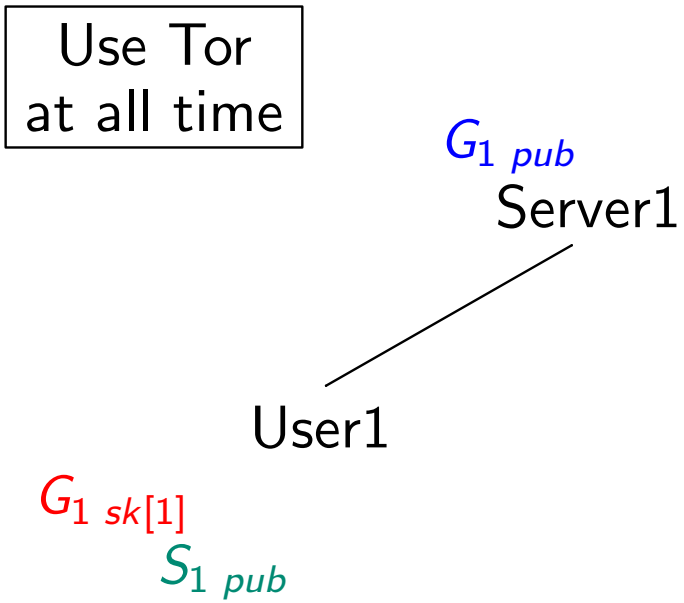
$G_{i\ pub}$: Group i public key

$G_{i\ sk[j]}$: Group i secret key of member j

$S_{i\ pub}$: Server i public key

$U_{i\ serv}$: User i home server

Network Diagram



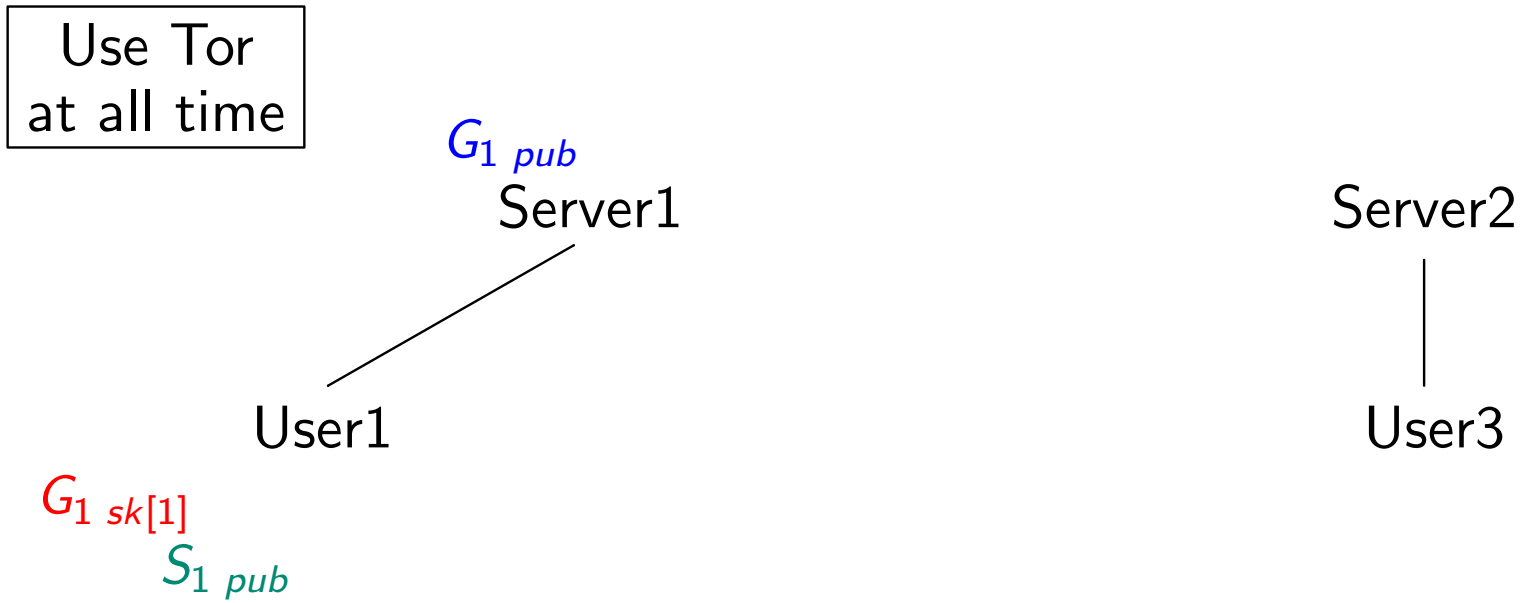
$G_{i\ pub}$: Group i public key

$G_{i\ sk[j]}$: Group i secret key of member j

$S_{i\ pub}$: Server i public key

$U_{i\ serv}$: User i home server

Network Diagram



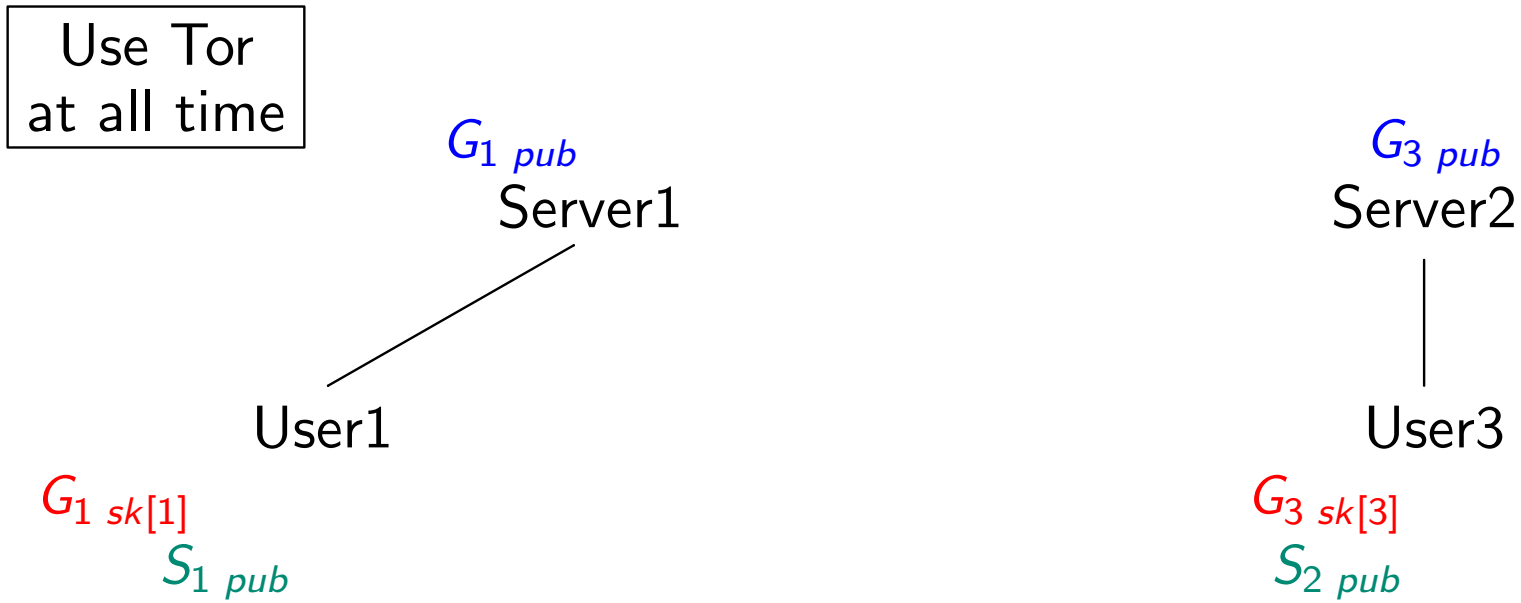
$G_{i\ pub}$: Group i public key

$G_{i\ sk[j]}$: Group i secret key of member j

$S_{i\ pub}$: Server i public key

$U_{i\ serv}$: User i home server

Network Diagram



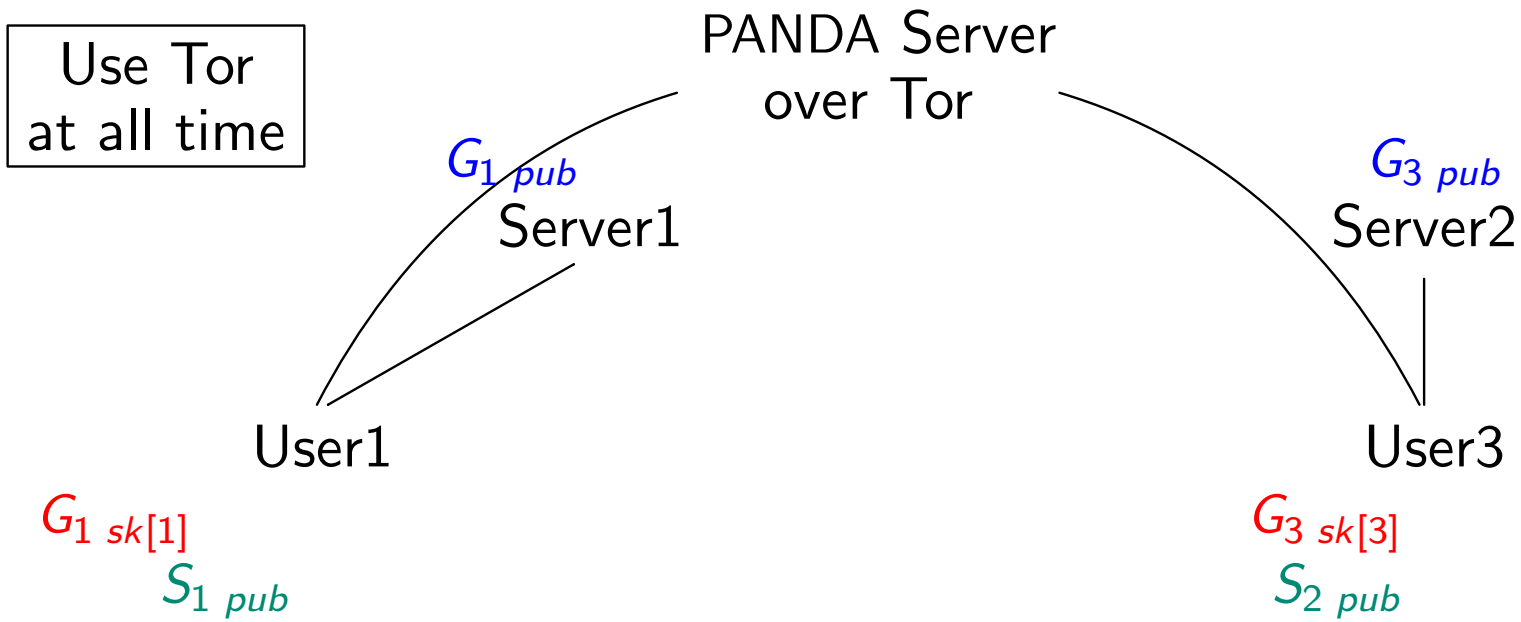
$G_{i\ pub}$: Group i public key

$G_{i\ sk[j]}$: Group i secret key of member j

$S_{i\ pub}$: Server i public key

$U_{i\ serv}$: User i home server

Network Diagram



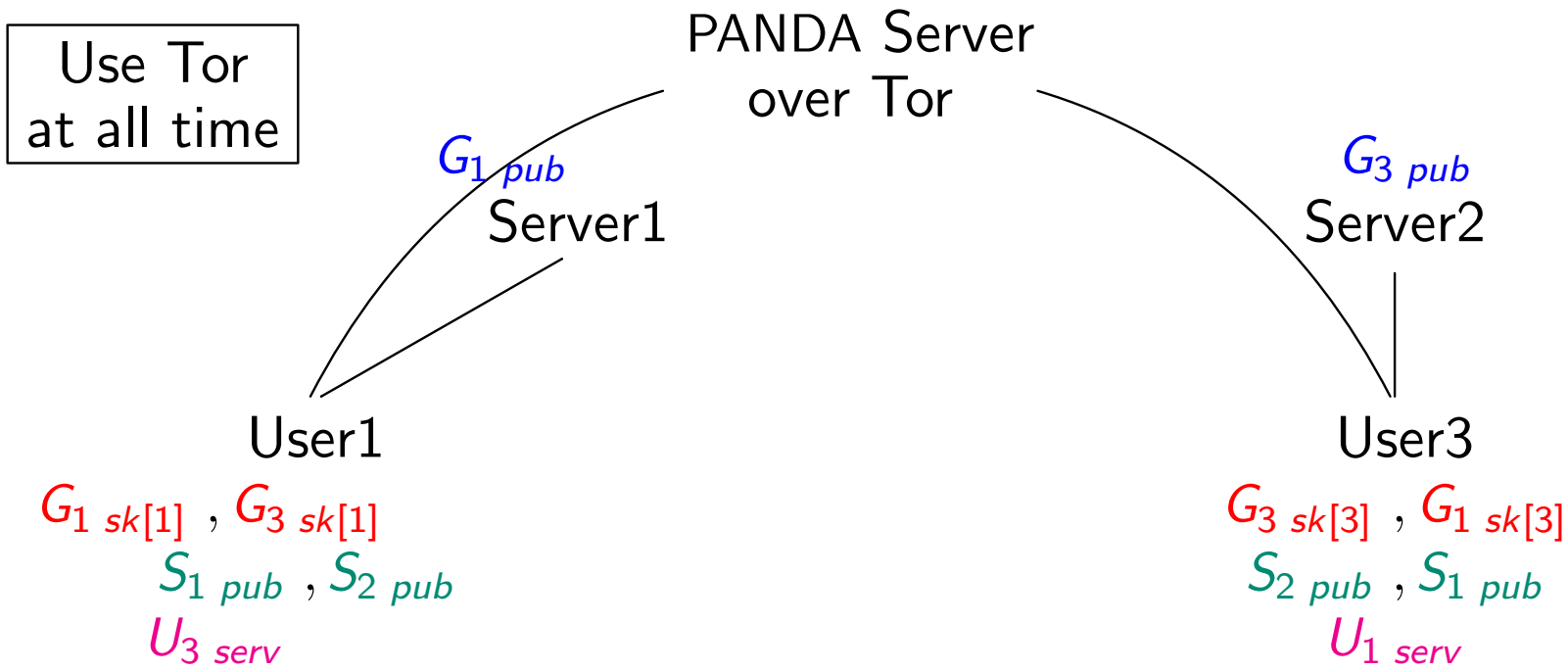
$G_{i\ pub}$: Group i public key

$G_{i\ sk[j]}$: Group i secret key of member j

$S_{i\ pub}$: Server i public key

$U_{i\ serv}$: User i home server

Network Diagram



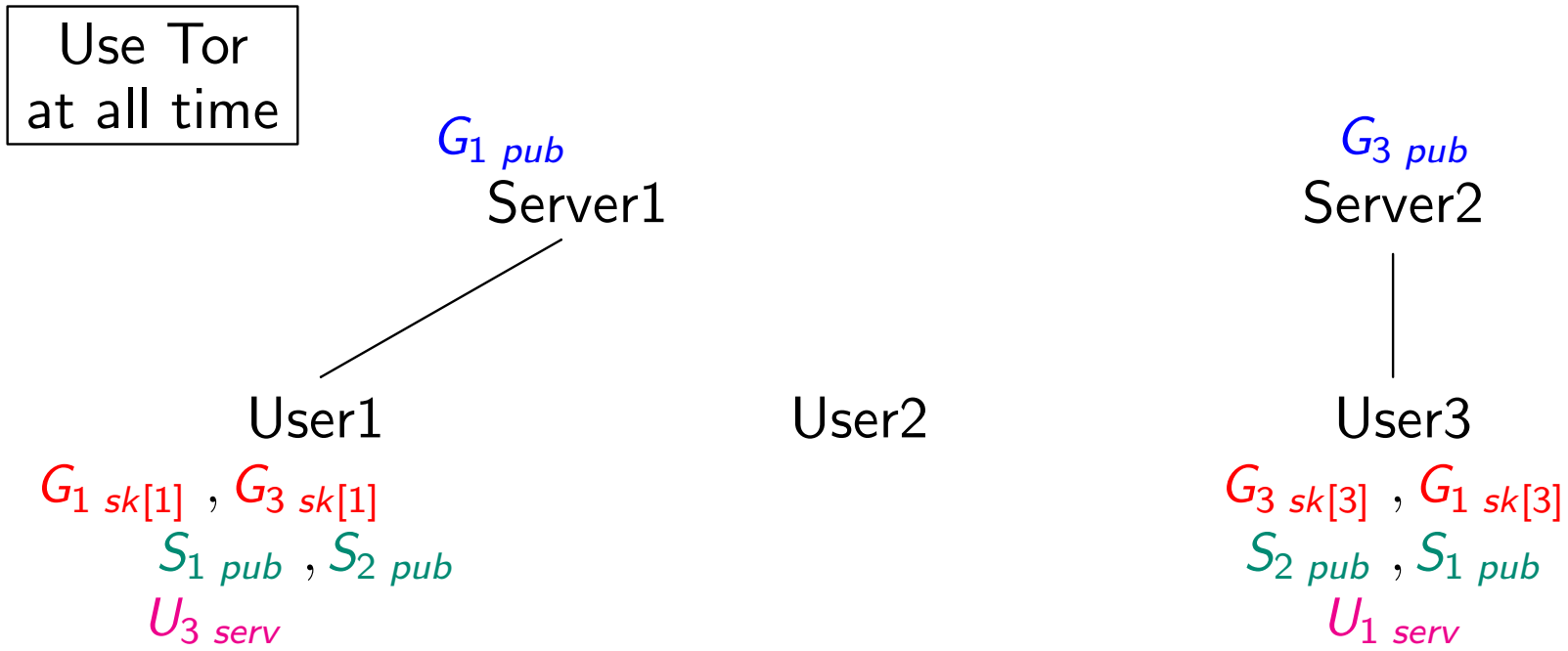
$G_{i\ pub}$: Group i public key

$G_{i\ sk[j]}$: Group i secret key of member j

$S_{i\ pub}$: Server i public key

$U_{i\ serv}$: User i home server

Network Diagram



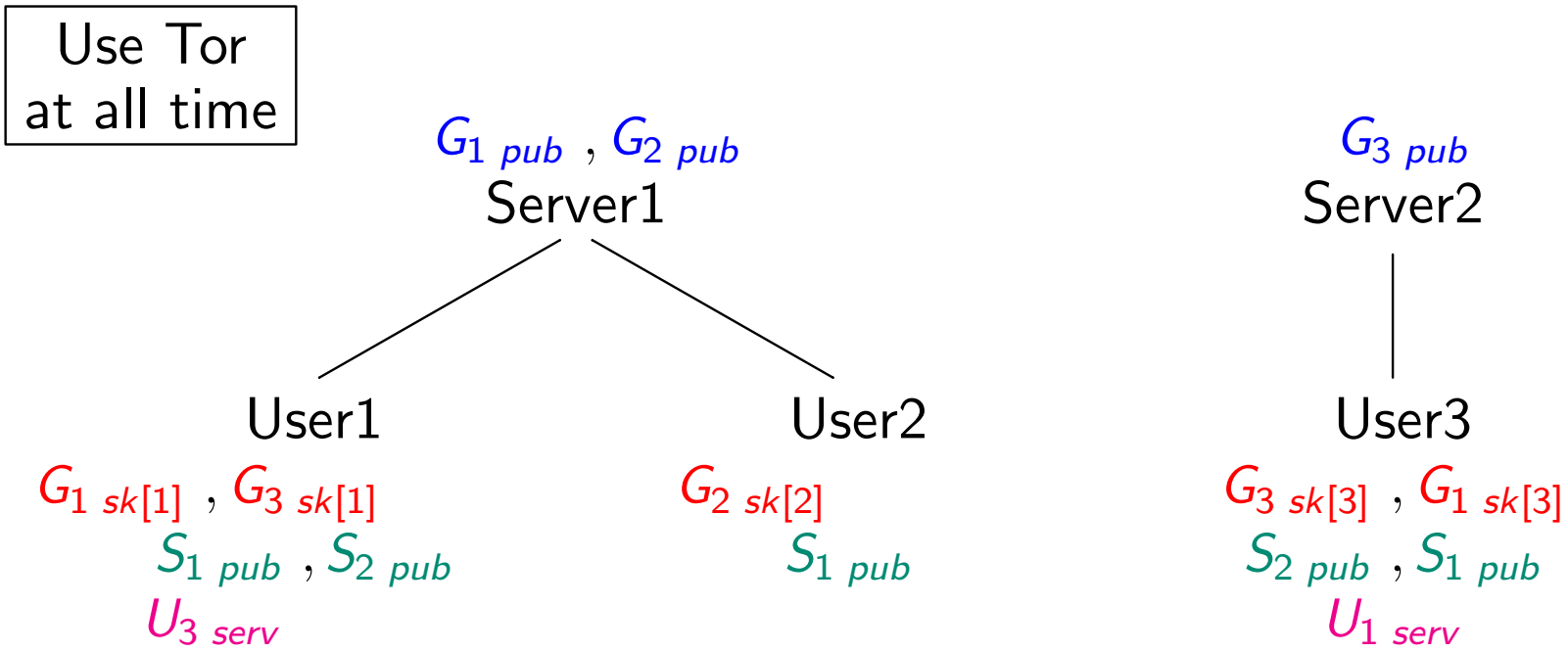
$G_{i\ pub}$: Group i public key

$G_{i\ sk[j]}$: Group i secret key of member j

$S_{i\ pub}$: Server i public key

$U_{i\ serv}$: User i home server

Network Diagram



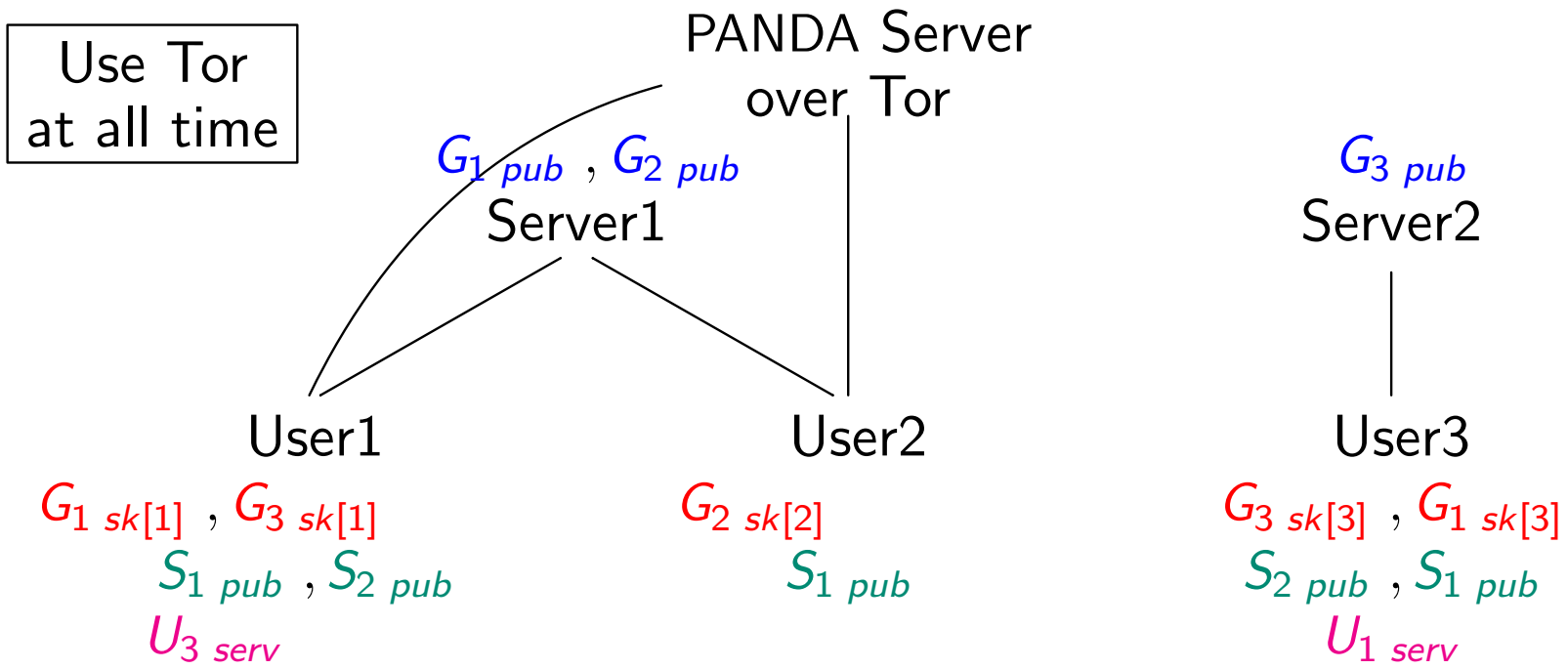
$G_{i\ pub}$: Group i public key

$G_{i\ sk[j]}$: Group i secret key of member j

$S_{i\ pub}$: Server i public key

$U_{i\ serv}$: User i home server

Network Diagram



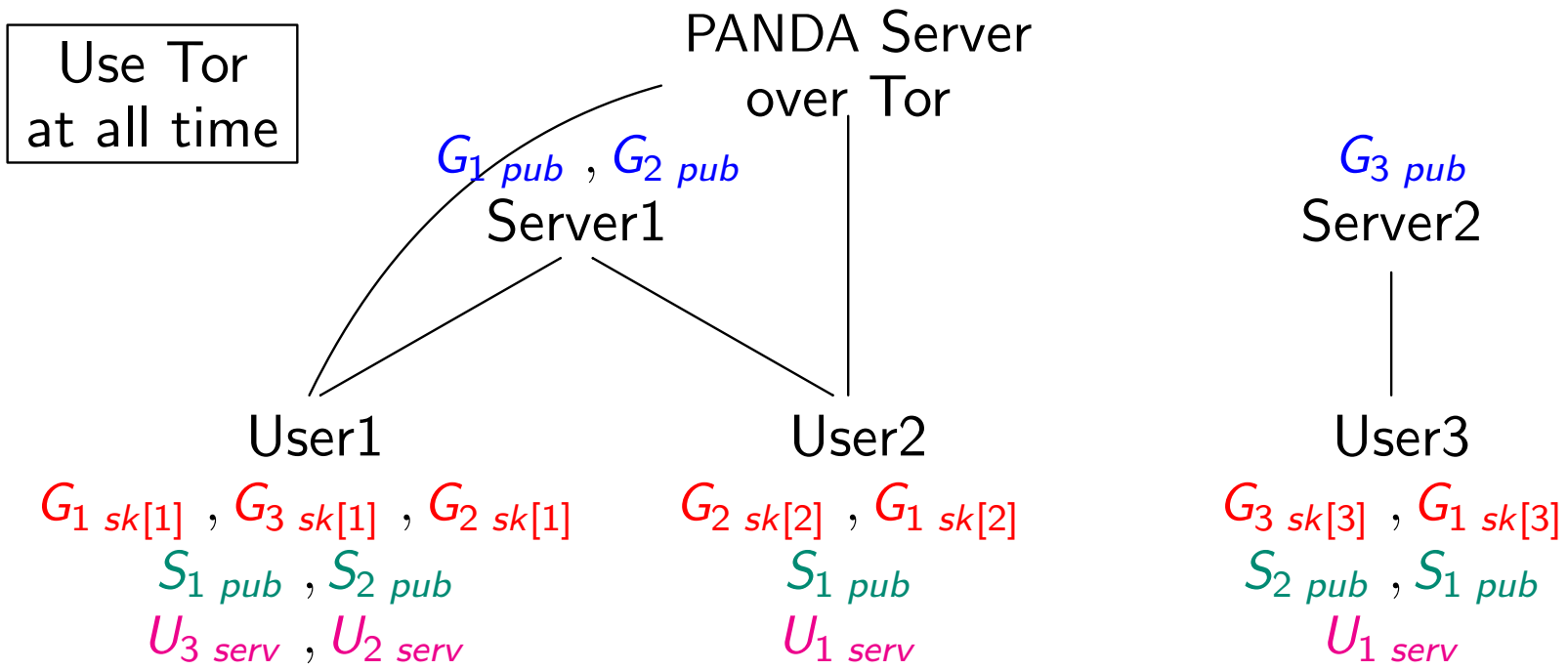
$G_{i\ pub}$: Group i public key

$G_{i\ sk[j]}$: Group i secret key of member j

$S_{i\ pub}$: Server i public key

$U_{i\ serv}$: User i home server

Network Diagram



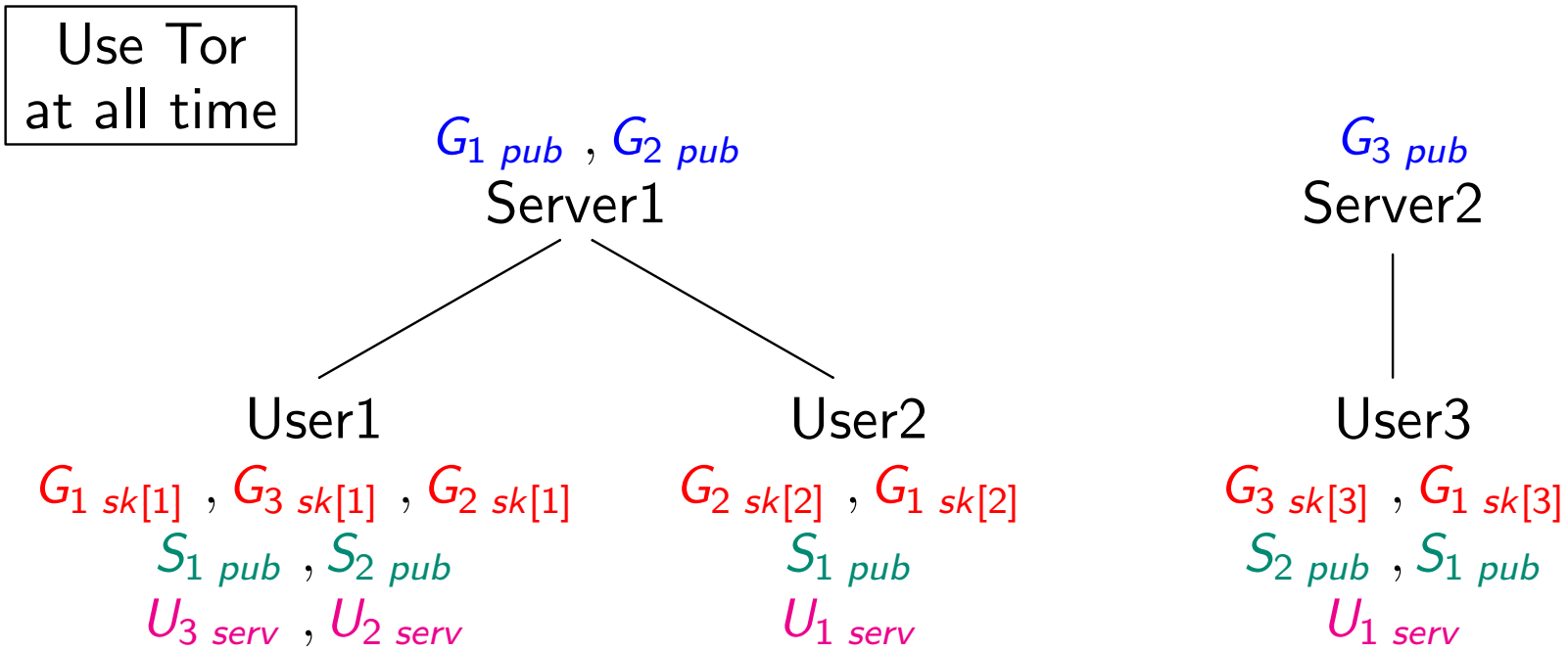
$G_{i\ pub}$: Group i public key

$G_{i\ sk[j]}$: Group i secret key of member j

$S_{i\ pub}$: Server i public key

$U_{i\ serv}$: User i home server

Network Diagram: receive



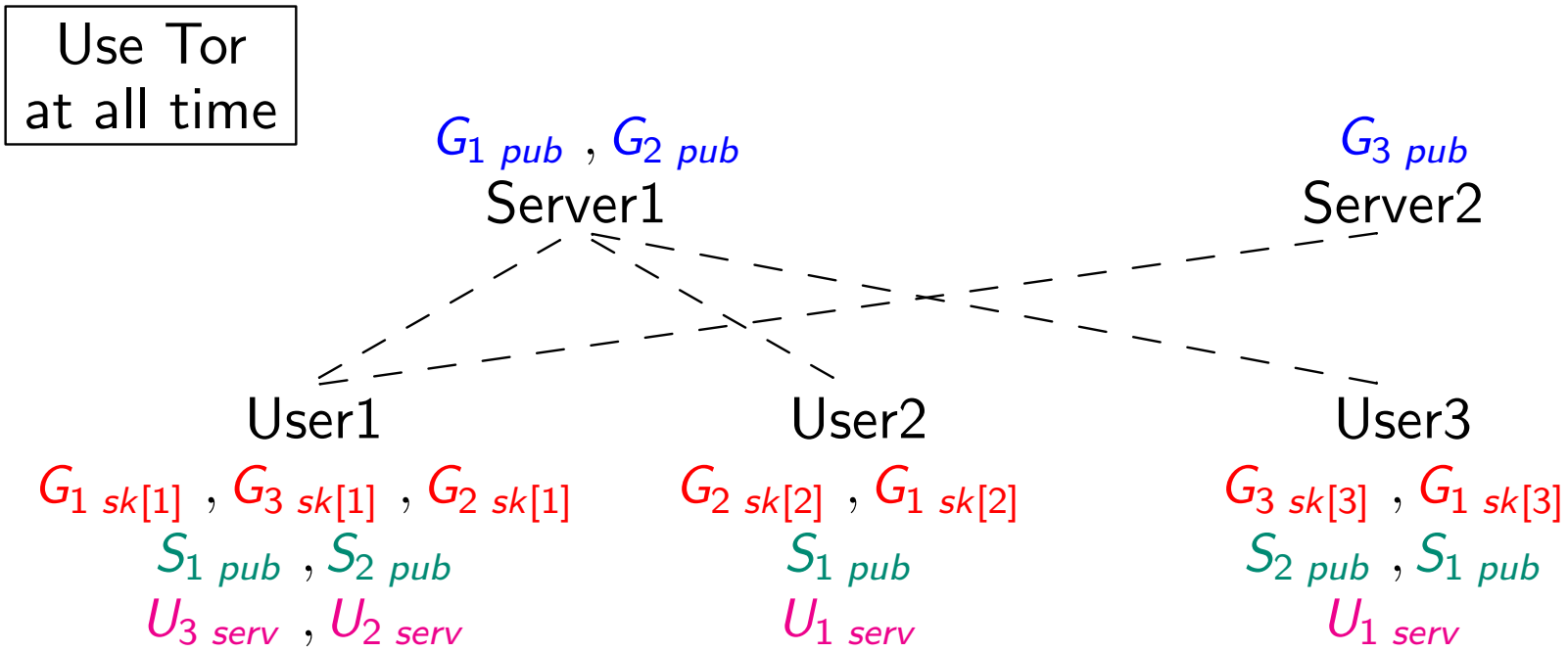
$G_{i\ pub}$: Group i public key

$G_{i\ sk[j]}$: Group i secret key of member j

$S_{i\ pub}$: Server i public key

$U_{i\ serv}$: User i home server

Network Diagram: send



$G_{i\ pub}$: Group i public key

$G_{i\ sk[j]}$: Group i secret key of member j

$S_{i\ pub}$: Server i public key

$U_{i\ serv}$: User i home server

BBS Group Signature

- Messages signed by a member in a group
  - Group_i: people authorized to send to i
- Server cannot learn which member of the group signed
- Revocation:
  - all previous signatures become invalid
  - each member has to update their private keys

BBS Group Signature: computational cost

- Precomputation:
  - 3 pairings (cached by both signers and verifiers)
  - 1 pairing (cached by signers)
- Sign:
  - 8 (multi-) exponentiations (7 in G_1, 1 in G_T)
  - 0 pairing
- Verify:
  - 6 multi-exponentiations (4 in G_1, 1 in G_2, 1 in G_T)
  - 1 pairing

Note: pairing $e : G_1 \times G_2 \rightarrow G_T$

BBS Group Signature: computational cost

- Precomputation:
  - 3 pairings (cached by both signers and verifiers)
  - 1 pairing (cached by signers)
- Sign:
  - 8 (multi-) exponentiations (7 in G_1, 1 in G_T)
  - 0 pairing
- Verify:
  - 6 multi-exponentiations (4 in G_1, 1 in G_2, 1 in G_T)
  - 1 pairing

Note: pairing $e : G_1 \times G_2 \rightarrow G_T$

Comment: This is not (yet) how Pond is implemented.

Future Research

- Open problems:
  - Formalize security assumptions
  - Prove protocol secure (or modify to make proof work)
- Desired feature:
  - Friends introduction
    A knows B and C; how can A introduces B to C?
  - Scalability of group signature
  - Decentralize PANDA
  - Multiple recipients
  - Group recipient
  - Post-quantum crypto