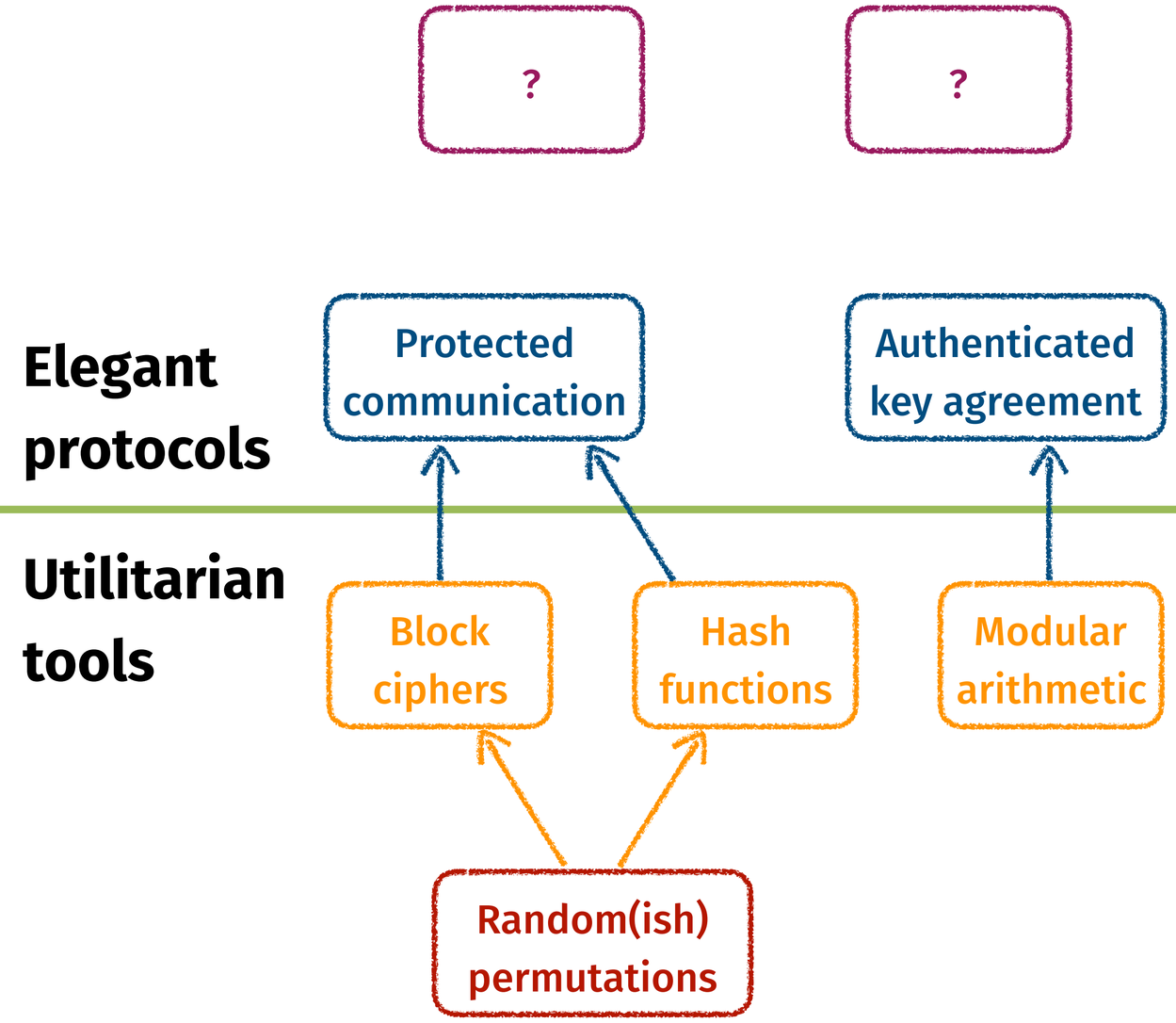
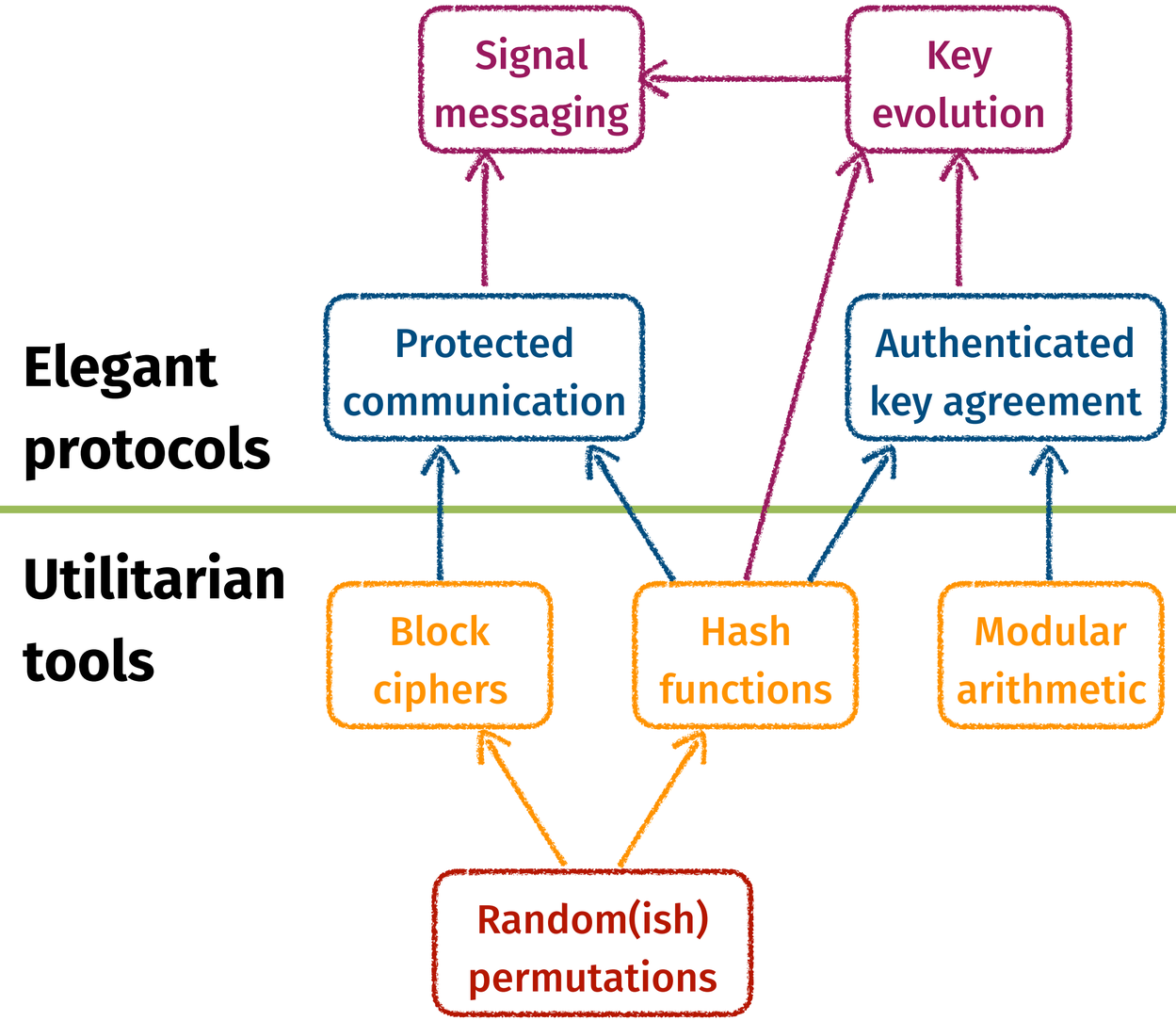
Lecture 16: Key evolution, aka ratcheting

- Lab 8 posted on Piazza (not Gradescope!), due Monday 4/8 at 11pm
- Guest lectures next week on (1) Encryption and the Fifth Amendment and (2) data privacy laws in the US and European Union
- Read next week's reading before Tuesday's lecture

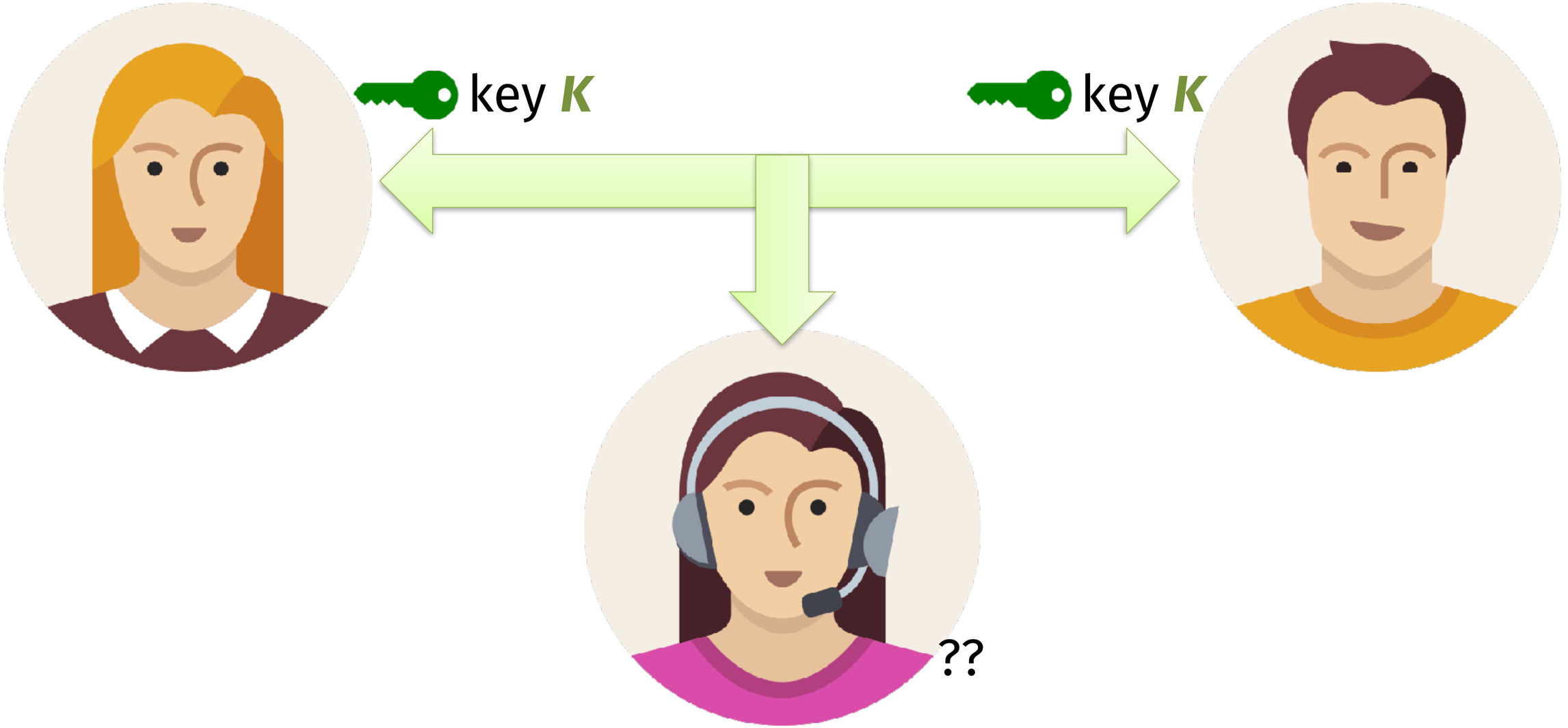
Course roadmap



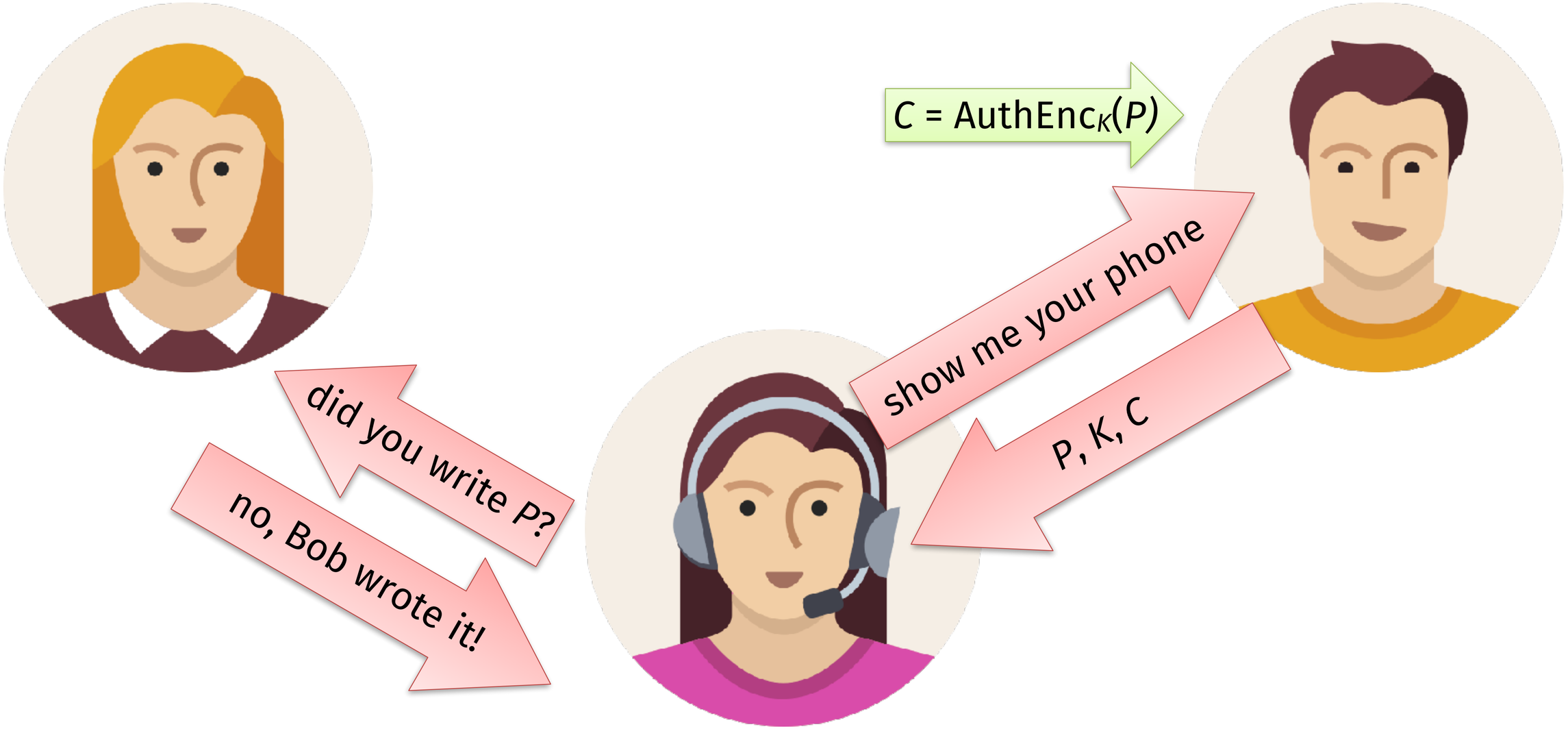
Course roadmap



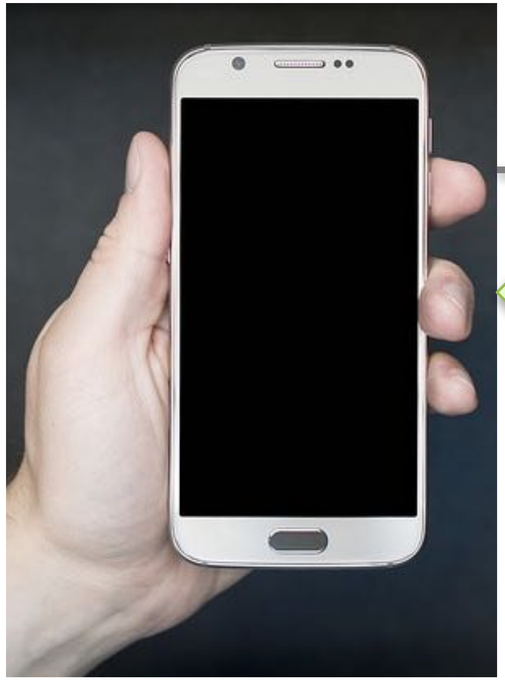
Part 3: Generate, exchange, evolve, and delete keys

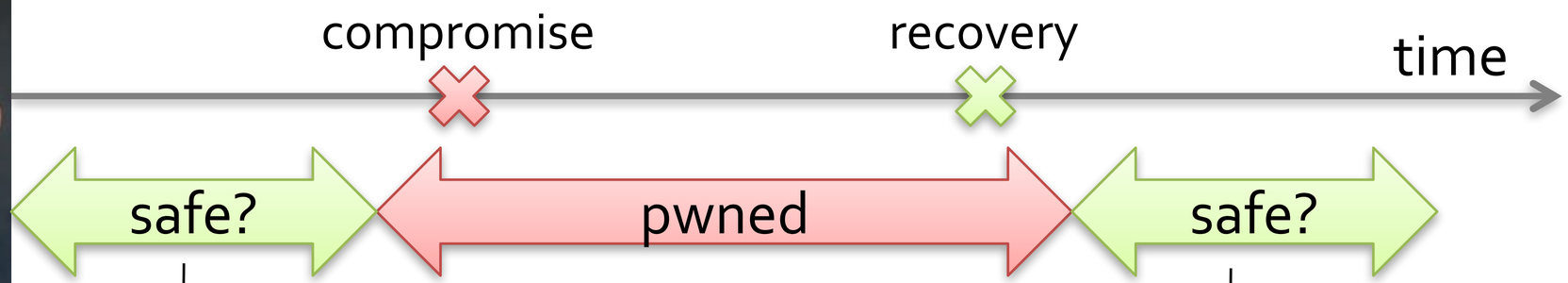


Security objective 1: Partial sender deniability



Security objective 2: Forward and backward secrecy





Forward (aka pre-compromise) secrecy: past messages remain confidential even if secret state is exposed later

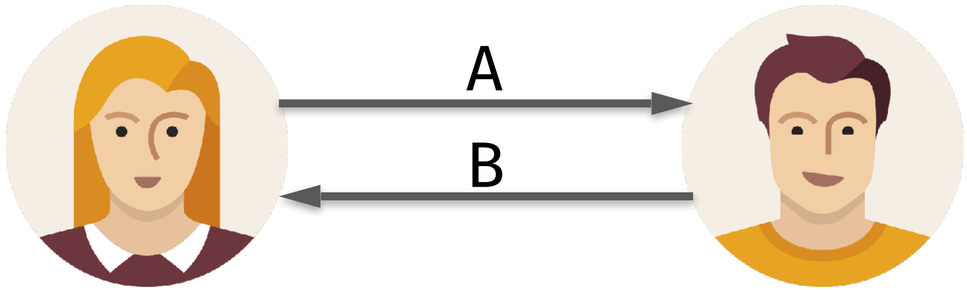
Backward (aka post-recovery) secrecy: future messages remain confidential even if secret state is compromised and then the device recovers

Last time: Diffie-Hellman key agreement

Protocol (for a publicly known g)

Choose *a* randomly
Compute $A = g^a$

Choose *b* randomly
Compute $B = g^b$

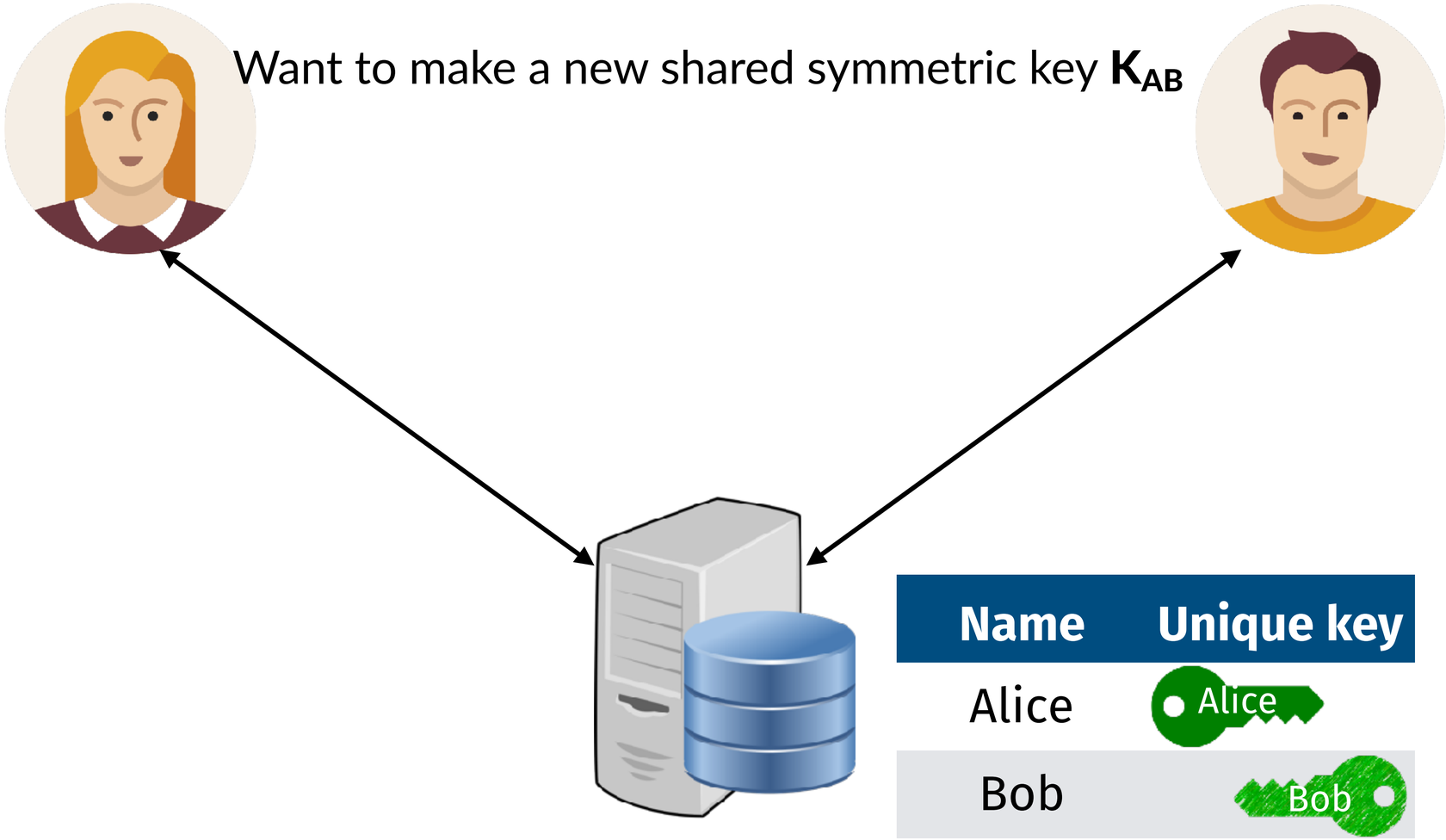


Output B^a

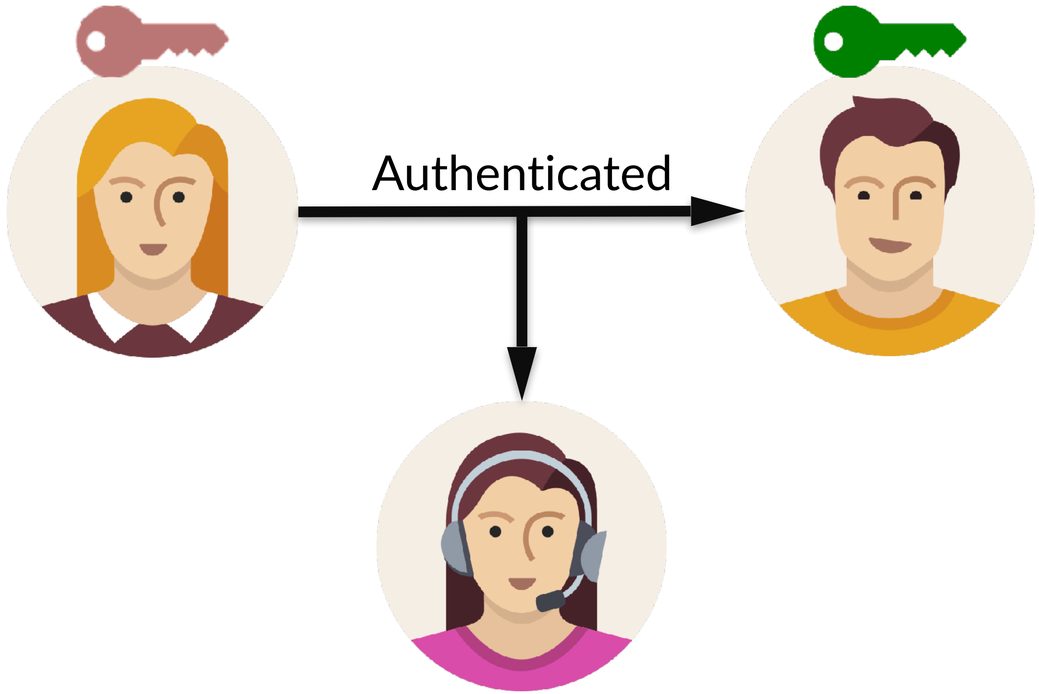
Output A^b

Shared secret = g^{ab}

Last time: PKI for initial key exchange

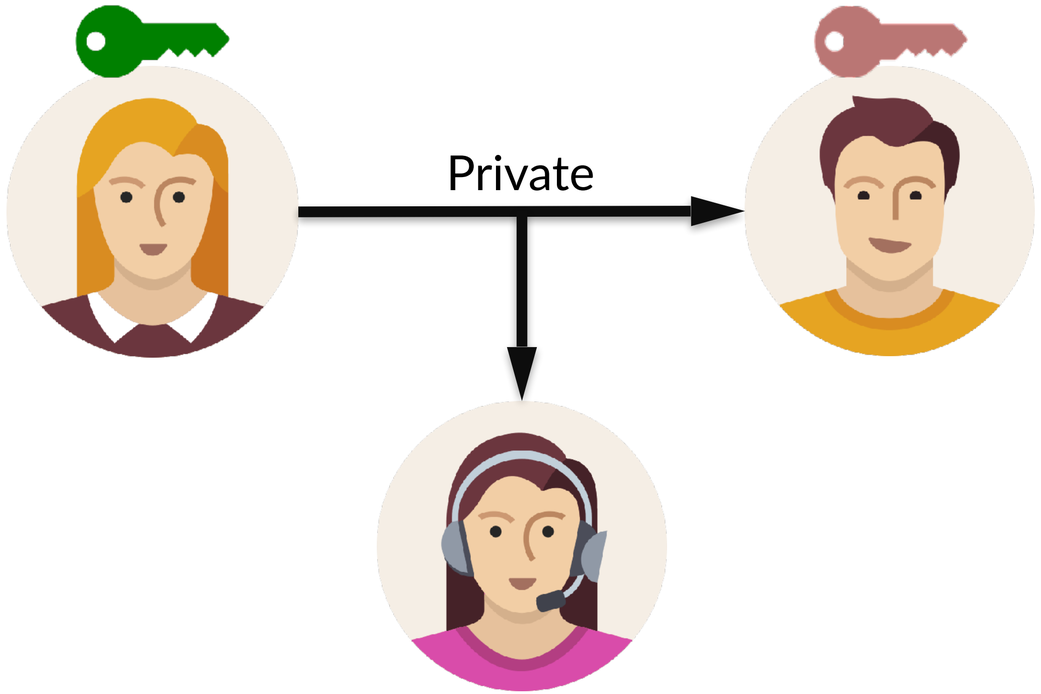


Public key signatures



- Only Alice can generate signatures
- Anybody can verify
- Security guarantee: EU-CMA

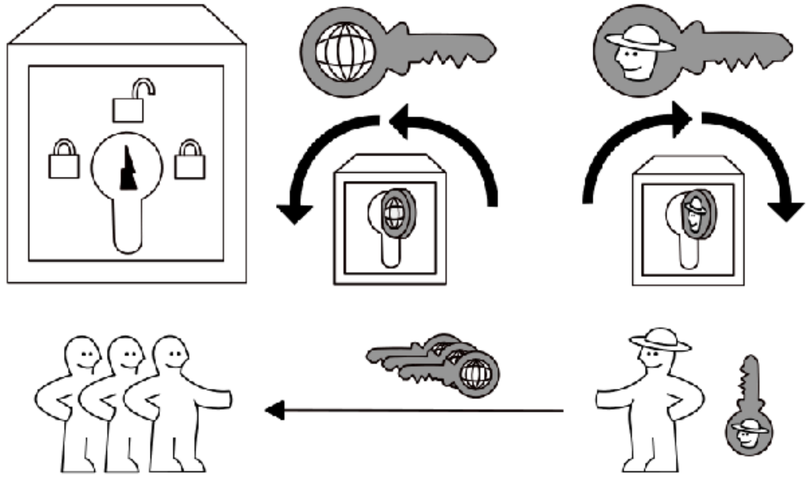
Public key encryption

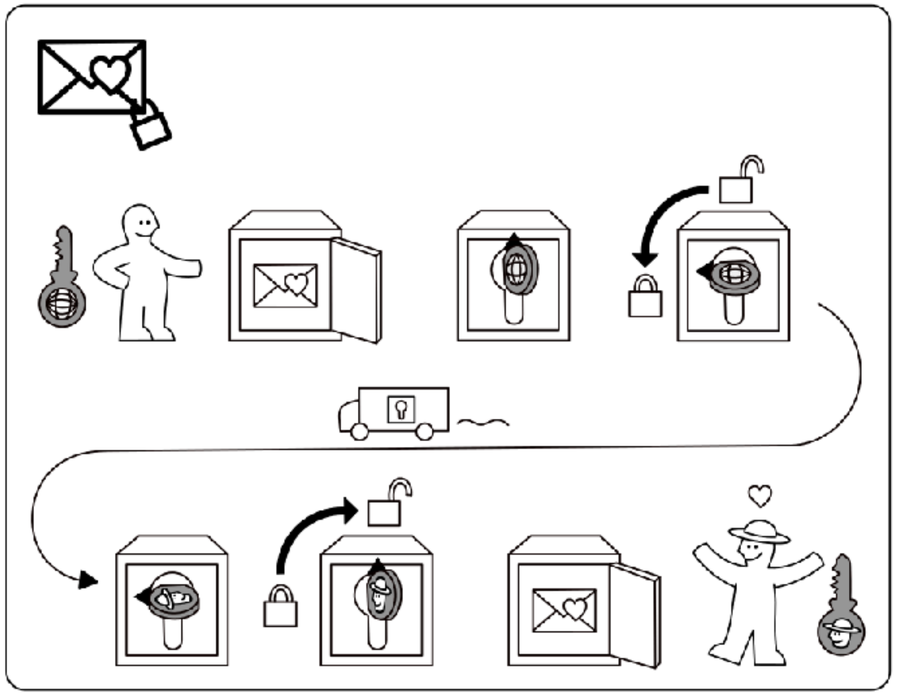


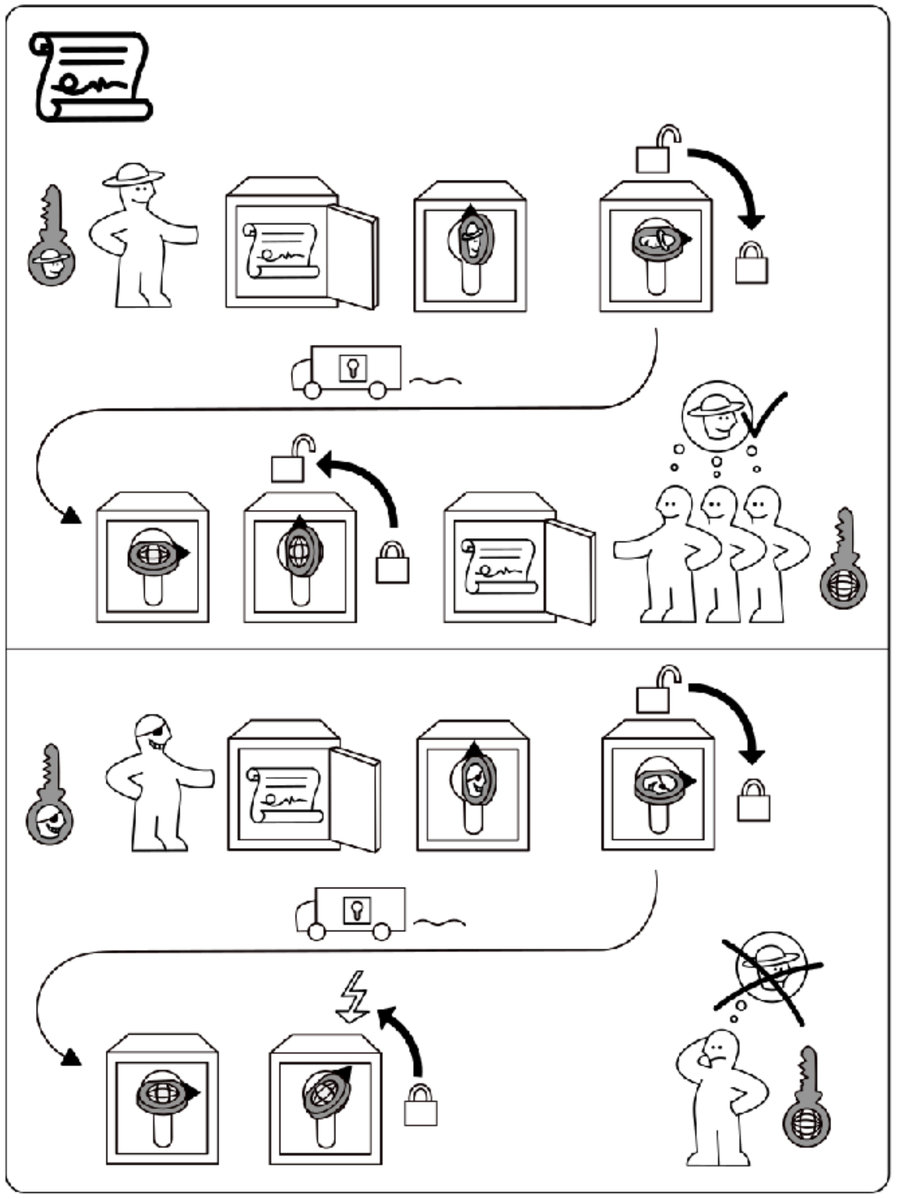
- Anybody can send ciphertexts
- Only Bob can decrypt + read
- Security guarantee: CPA or CCA

PUBLIC KEY KRÜPTO

idea-instructions.com/public-key/
v1.0, CC by-nc-sa 4.0
IDEA







SignCryption = public key version of Auth Enc

Construction

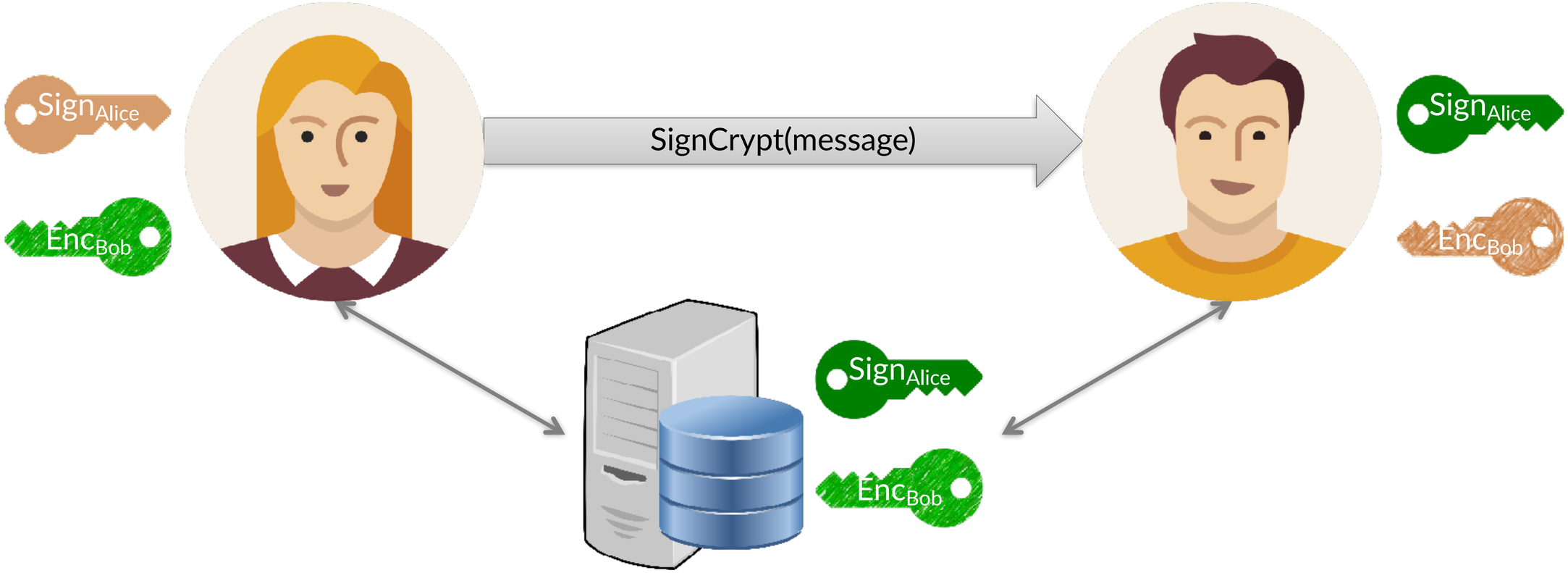
- Integrity via pk sign
- Confidentiality via pk encrypt

But any issue persists

- Non-repudiable
- Risk of device compromise

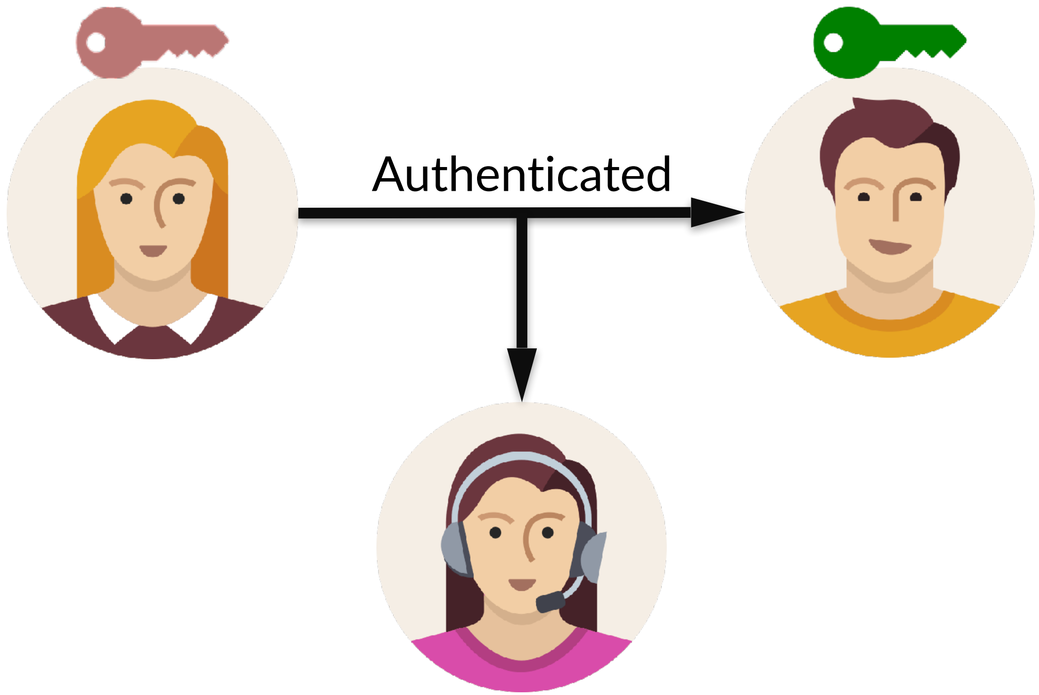
What we want

- Deniability
- Forward secrecy



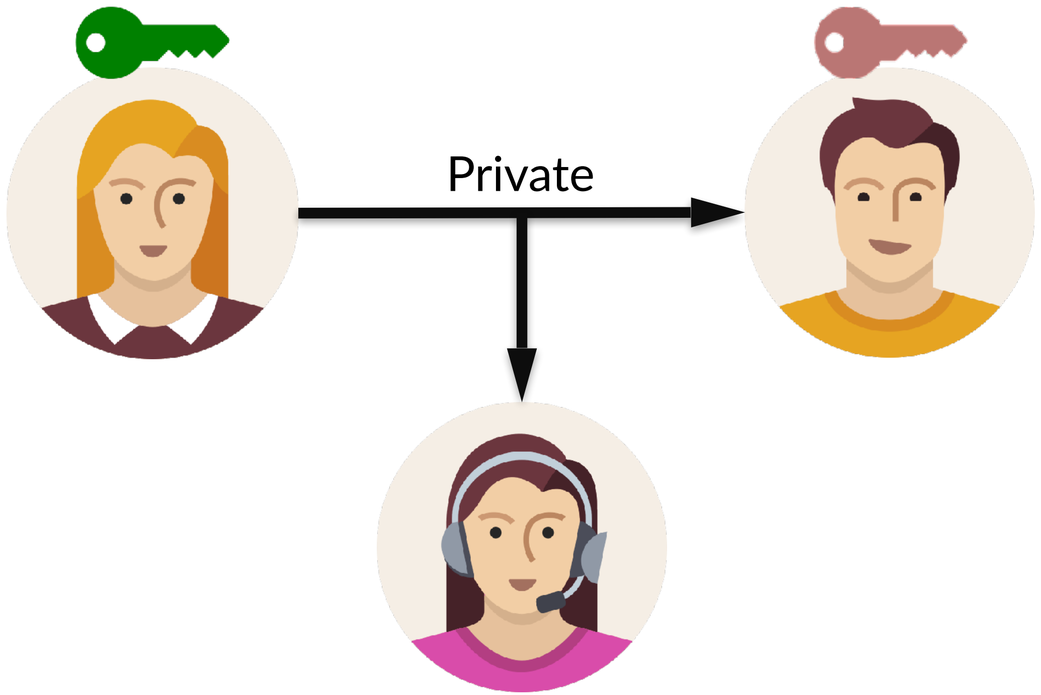
Consequences of losing

Public key signatures



- Problem: Eve forges msgs in *future*
- Response: Alice can *revoke* her key

Public key encryption

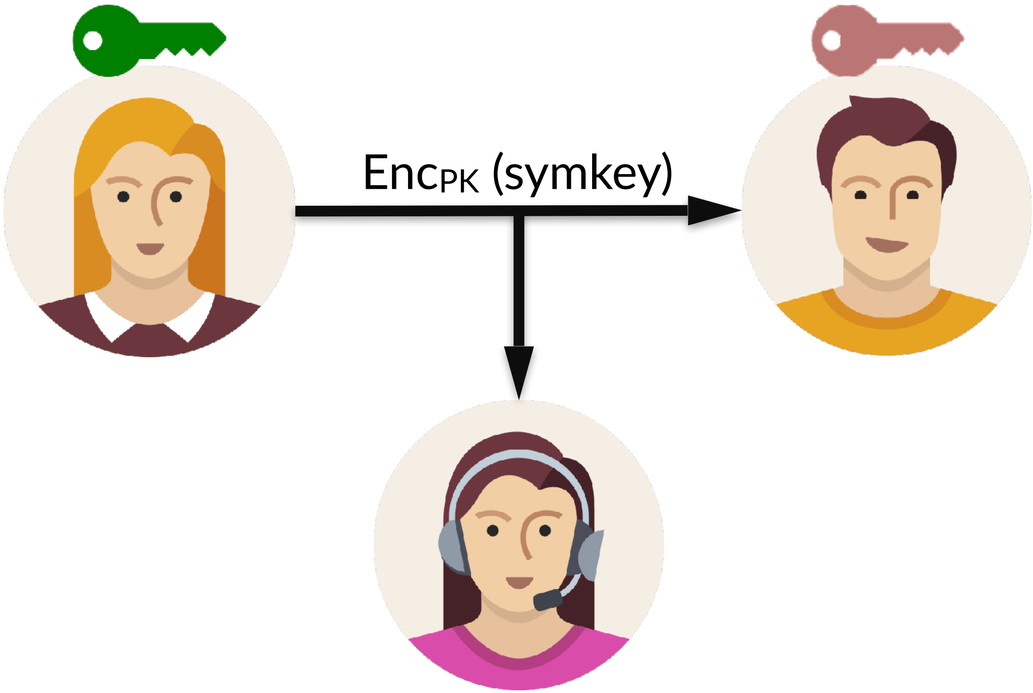


- Problem: Eve reads msgs from *past*
- Response: ???

Key encapsulation → Hybrid encryption

If you must use public key enc…

- Only use it once in order to encrypt (or encapsulate) a symmetric key that you will use from now onward
- This object is formally called a key encapsulation mechanism (KEM)
- This overall procedure is called hybrid encryption



Today: Key evolution

Server trust?	Crypto used	Method
Full (learns keys)	Symmetric	Needham-Schroeder ⟹ Kerberos system
Partial (need PKI)	Asymmetric	(Authenticated) Diffie-Hellman key exchange
None	Symmetric	Key evolution, starting from an initial shared symmetric key

Key evolution

Question: Once Alice and Bob negotiate a shared symmetric key K_{AB} for authenticated encryption, must they re-execute another (expensive) key negotiation protocol each time they want to update the key?

Basically, seek Authenticated Encryption with a new updating mechanism

- KeyGen: randomly choose key K of length λ, e.g. uniform in $\{0,1\}^\lambda$
- AuthEnc_K (private P, authenticated A, nonce N) → ciphertext C
- AuthDec_K (C, A, N) → P or $\perp$
- KeyUpdate (K) → K' where Alice + Bob agree to use K' from now onward, and cannot compute K from K'

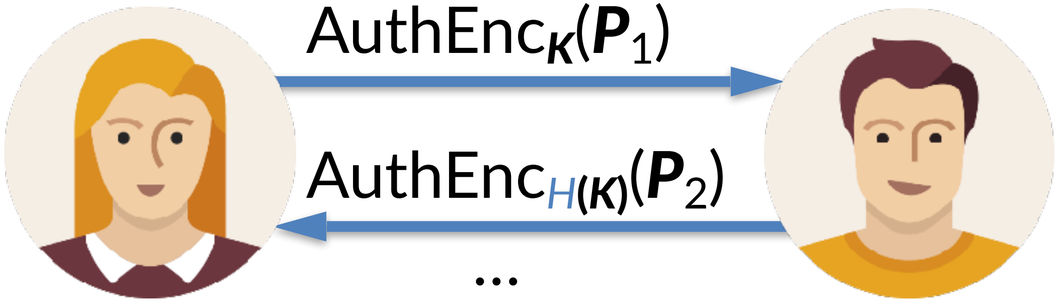
Key evolution via hash functions

Idea: Once we have a single shared key K_{AB}, expand using a chain of hash functions

$$K_{AB} \rightarrow H(K_{AB}) \rightarrow H(H(K_{AB})) \rightarrow H(H(H(K_{AB}))) \rightarrow \ldots$$

Algorithm:

- Alice + Bob agree on key K_{AB} to use for auth enc
- After some time has passed, they can *evolve* their key by updating $K \leftarrow H(K)$
  - Here, “time” can denote actual wall-clock time or a message counter
  - Alice + Bob must stay in sync, or else the chain breaks & they need to redo key agreement
- Crucially, they ensure that old values of K are deleted from their system!
  Evolution relies on the fact that Mallory cannot steal something that isn’t around to be stolen

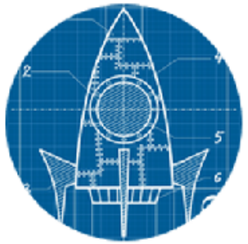


Axolotl protocol, aka Signal protocol, aka double ratchet

Used in a messaging system near you!

- Signal
- WhatsApp
- Facebook Messenger
- Google Allo
- Skype

Open Whisper Systems



The Double Ratchet Algorithm

Revision 1, 2016-11-20 [PDF]

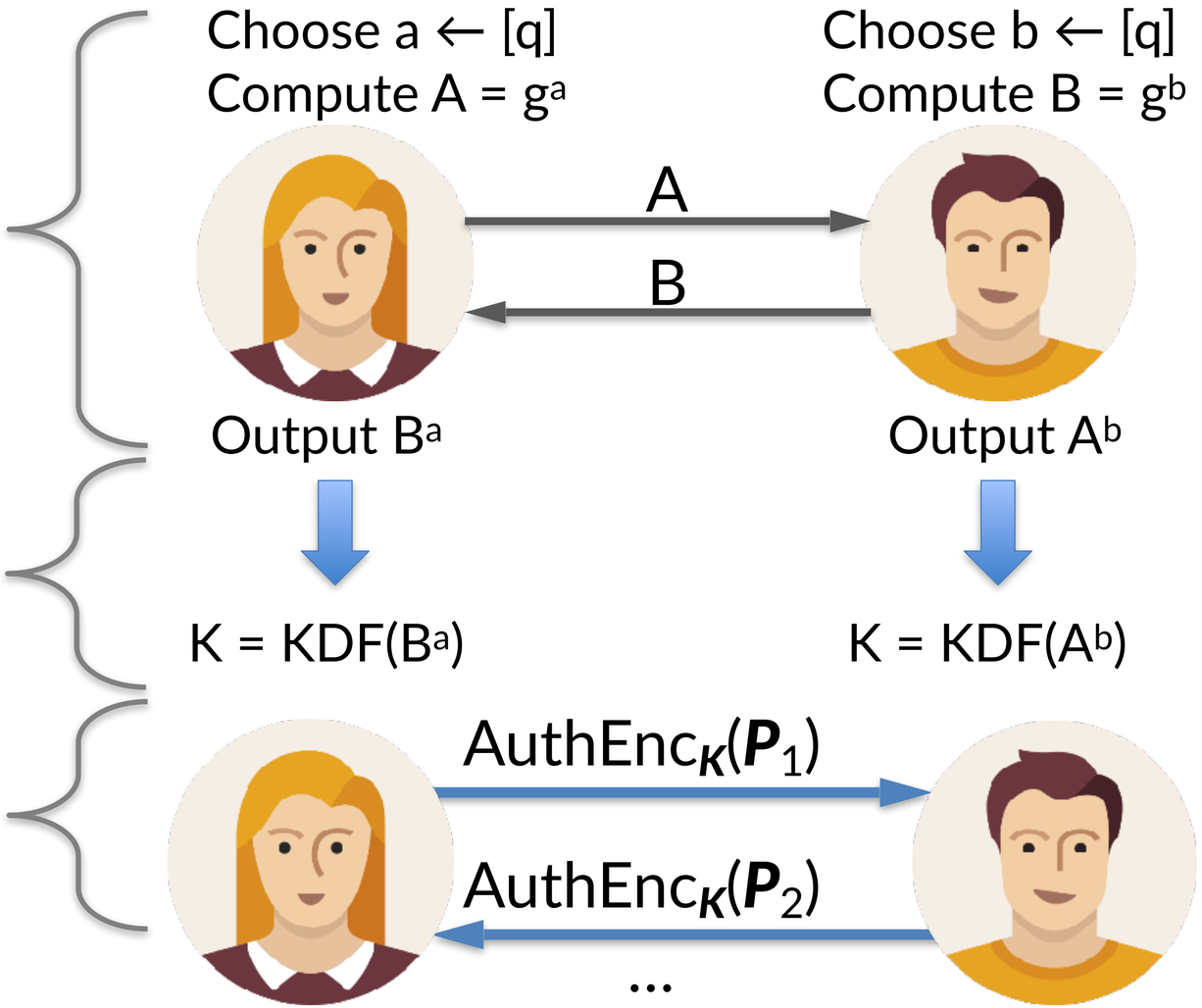
Trevor Perrin (editor), Moxie Marlinspike

Signal combines key exchange and evolution

Server trust?	Crypto used	Method
Full (learns keys)	Symmetric	Needham-Schroeder ⟹ Kerberos system
Partial (need PKI)	Asymmetric	(Authenticated) Diffie-Hellman key exchange
None	Symmetric	Key evolution, starting from an initial shared symmetric key

Last time: Public key exchange + AE ⇒ Deniability

1. Diffie-Hellman key exchange
   (* But use the authenticated version)
2. Derive a symmetric key
3. Use authenticated encryption

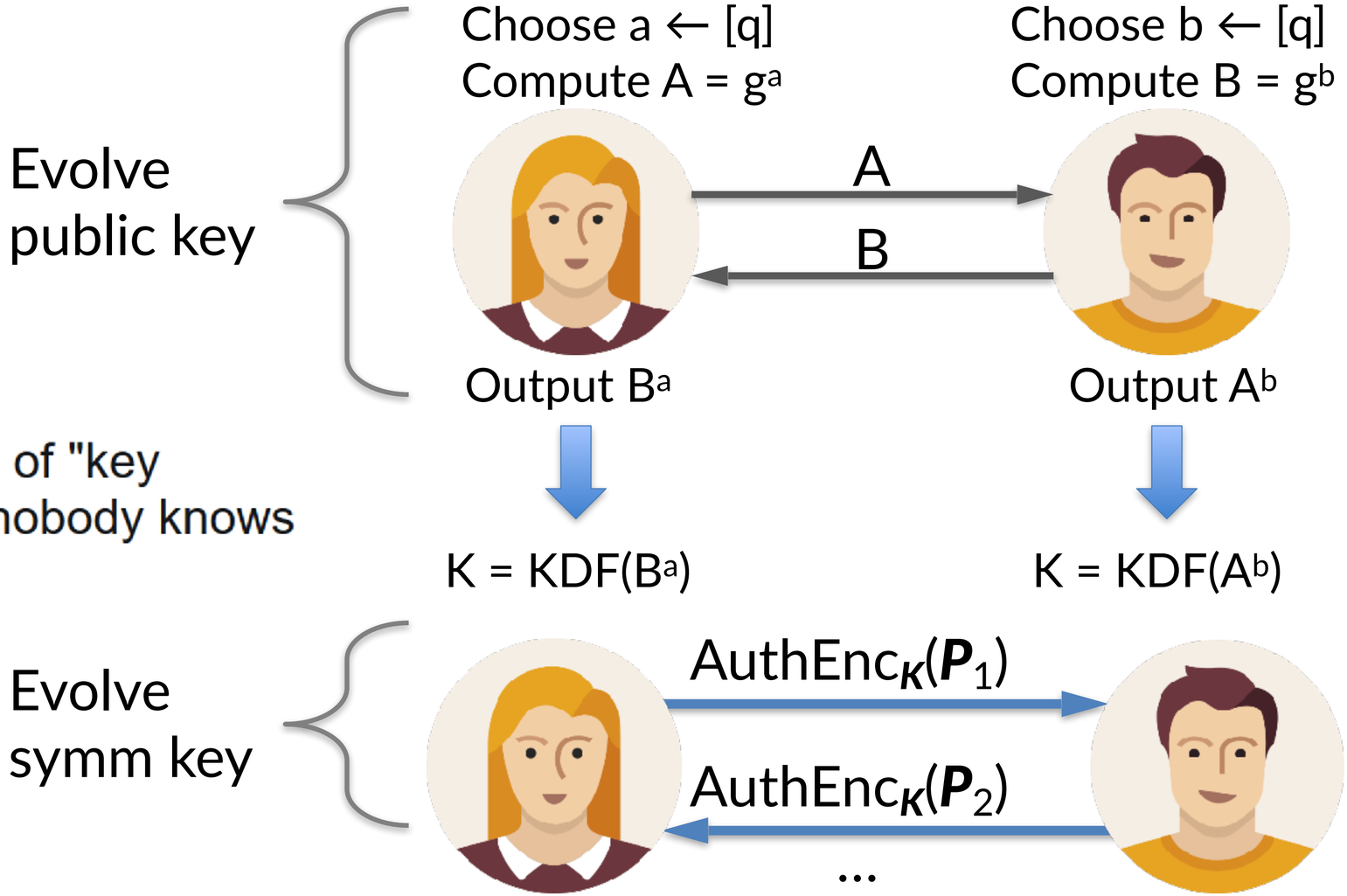


But how can we get forward secrecy too?

Today: Key evolution ⇒ forward secrecy!

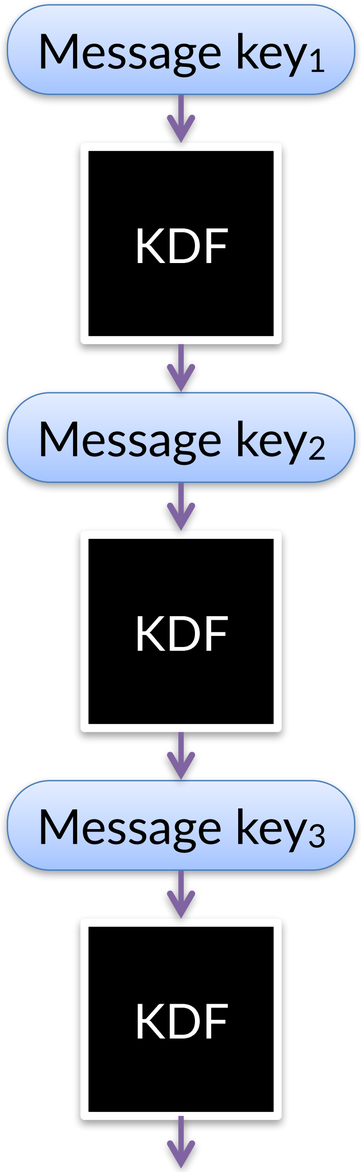
JP Aumasson
@veorq

want to sound cool? don't talk of "key agreement", but of "ratchet" (nobody knows the difference anyway)



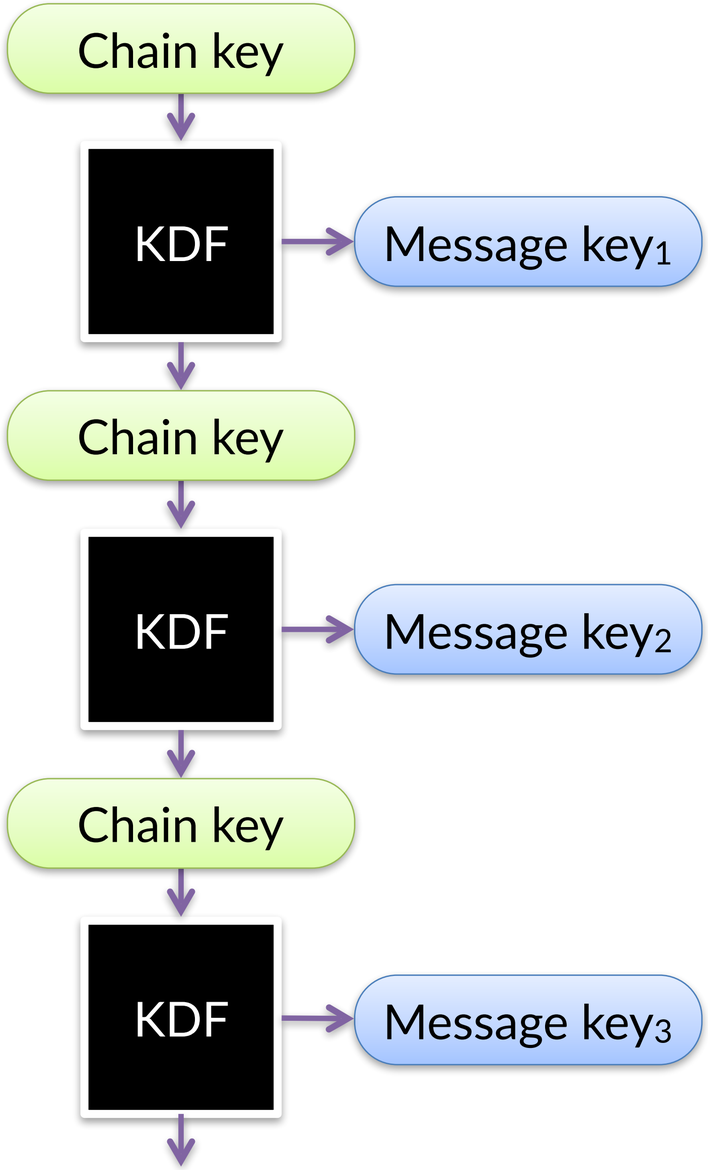
Signal messaging protocol (simplified)

1. Key evolution
   - Use each key to encrypt only 1 message
   - Concern: If initial **K** is lost, whole chain insecure



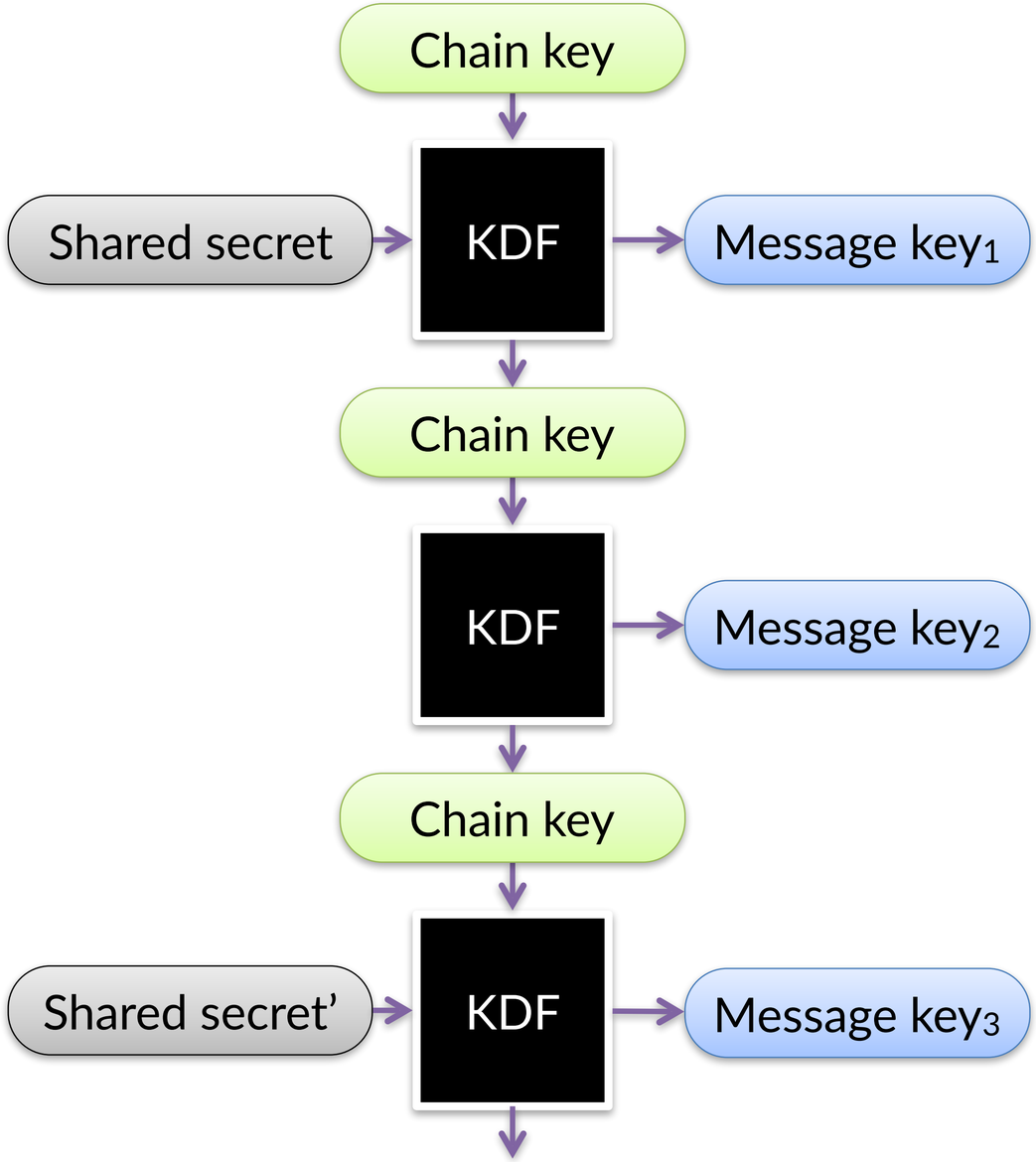
Signal messaging protocol (simplified)

1. Key evolution
   - Use each key to encrypt only 1 message
   - Concern: If initial **K** is lost, whole chain insecure
2. Key derivation
   - No direct link betweeen message keys
   - Engineering: Handle out of order messages?



Signal messaging protocol (simplified)

1. Key evolution
   - Use each key to encrypt only 1 message
   - Concern: If initial **K** is lost, whole chain insecure
2. Key derivation
   - No direct link betweeen message keys
   - Engineering: Handle out of order messages?
3. Key ratcheting
   - Can recover from chain key compromise
   - Opportunity: If ephemeral secret is unknown to adv, then ∃ potential for *post-compromise secrecy*



Double ratchet rules

1. When a message is sent or received, a *symmetric ratchet* KDF step is applied to the sending or receiving chain to derive a new message key

2. When a new ratchet public key is received, a *public ratchet* step is performed prior to the symmetric-key ratchet to replace the chain keys

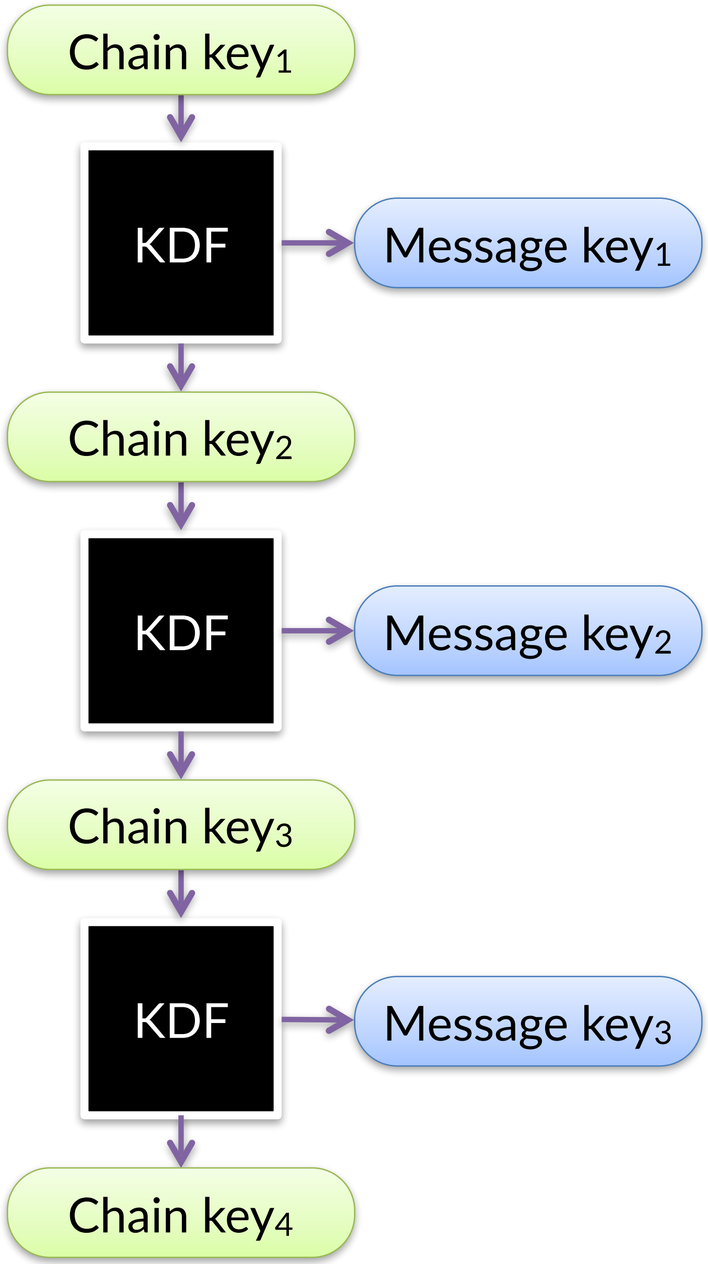
Let’s build each ratchet, then connect them to each other

Symmetric ratchet

- Modified version of the “chain of hash functions” idea K → K1 = *H*(K) → K2 = *H*(K1) → …
- Alice and Bob maintain *two* such ratchets, one for Alice → Bob messages and other for Bob → Alice
- Delete every chain + message key after using it

  “Because message keys aren't used to derive any other keys, message keys may be stored without affecting the security of other message keys. This is useful for handling lost or out-of-order messages”

Source: https://whispersystems.org/docs/specifications/doubleratchet/



Public ratchet

Use Diffie-Hellman key exchange to create the “shared secrets”

Choose a1 ← [q]

$B1 = g^{b1}$

$A1 = g^{a1}$

Shared secret $g^{a1 \cdot b1}$

Choose b1 ← [q]

$B2 = g^{b2}$

Choose b2 ← [q]

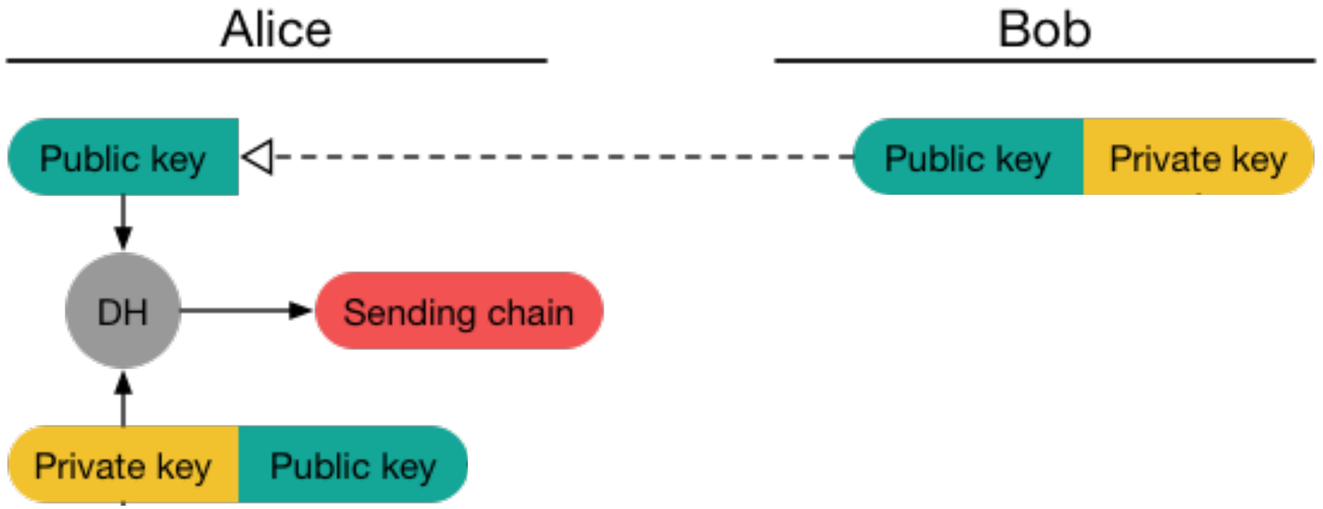
Shared secret $g^{a1 \cdot b2}$

Choose a2 ← [q]

$A2 = g^{a2}$

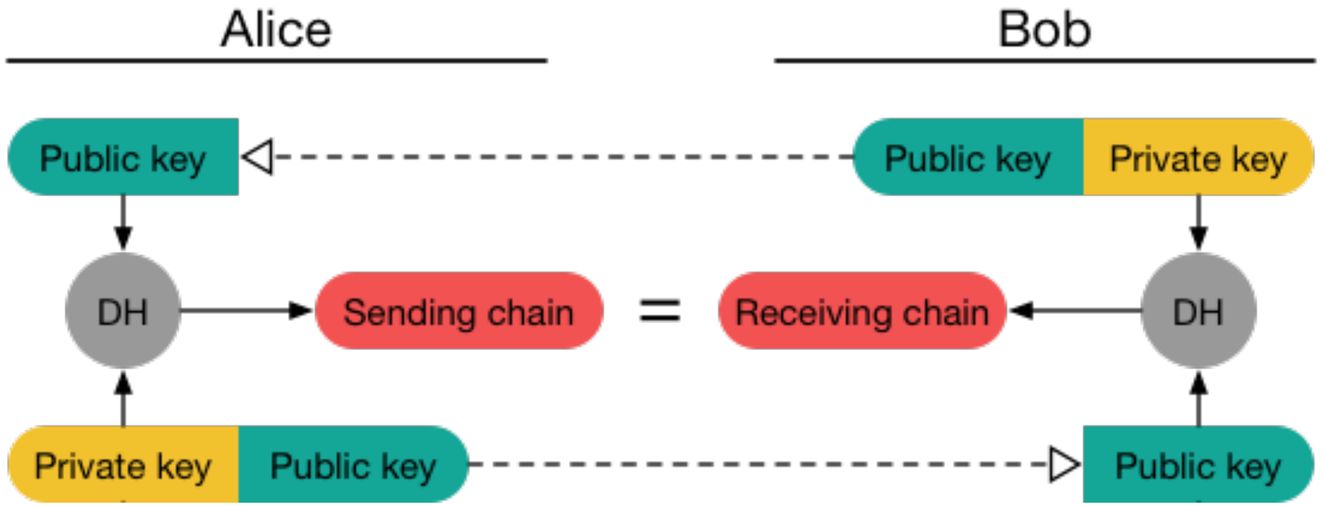
Shared secret $g^{a2 \cdot b2}$

Public ratchet seeds symmetric ratchets (one per direction)



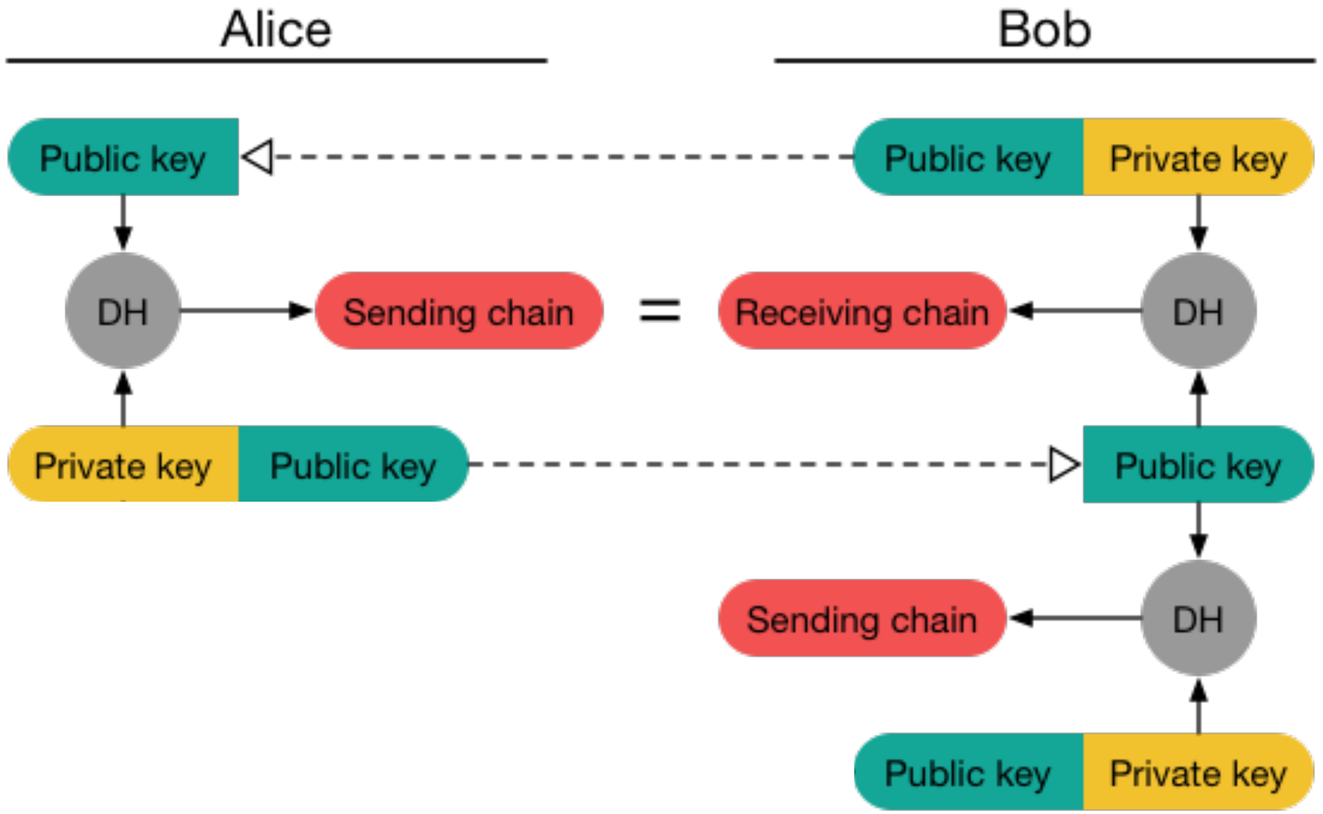
Source: https://whispersystems.org/docs/specifications/doubleratchet/

Public ratchet seeds symmetric ratchets (one per direction)



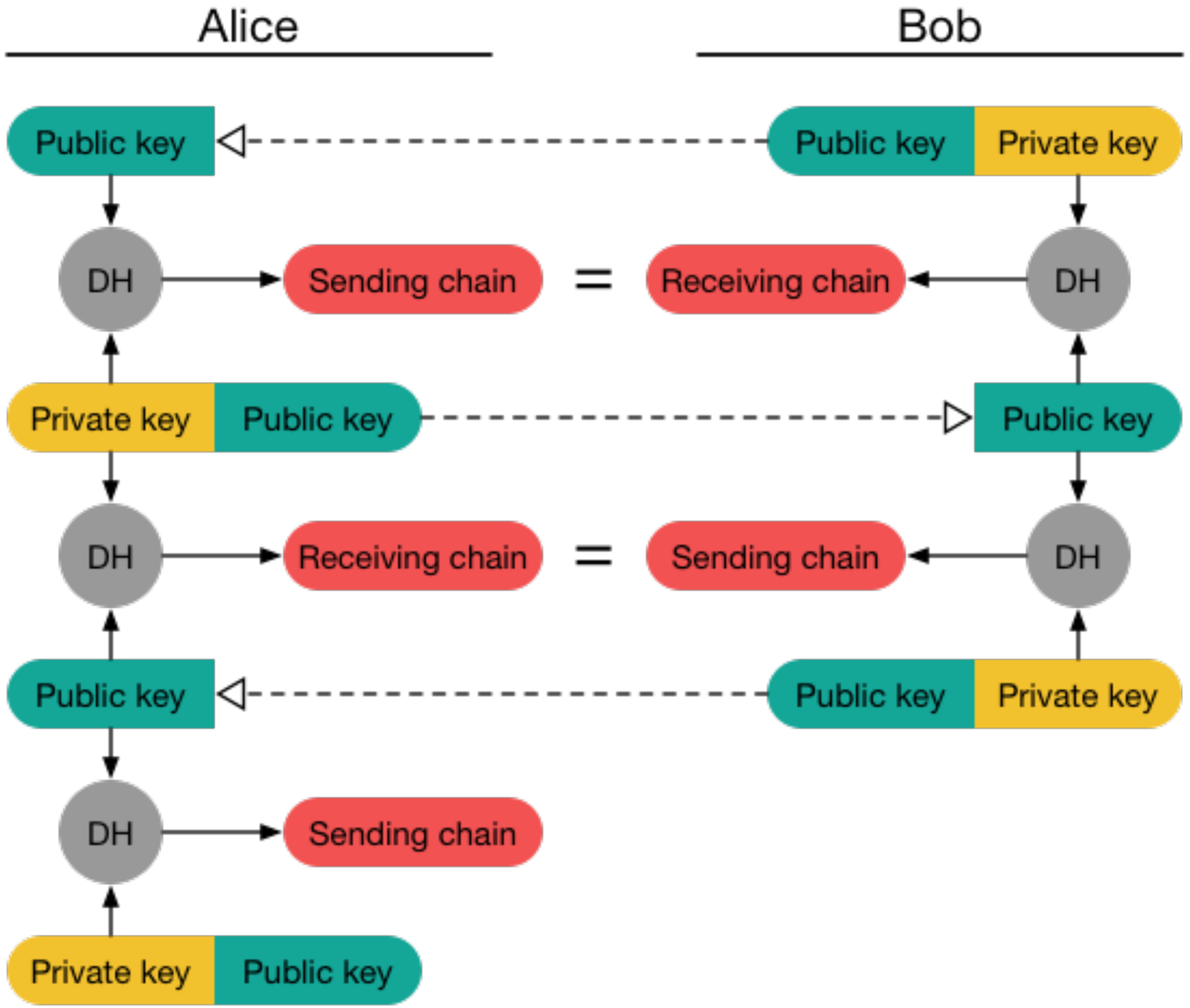
Source: https://whispersystems.org/docs/specifications/doubleratchet/

Public ratchet seeds symmetric ratchets (one per direction)



Source: https://whispersystems.org/docs/specifications/doubleratchet/

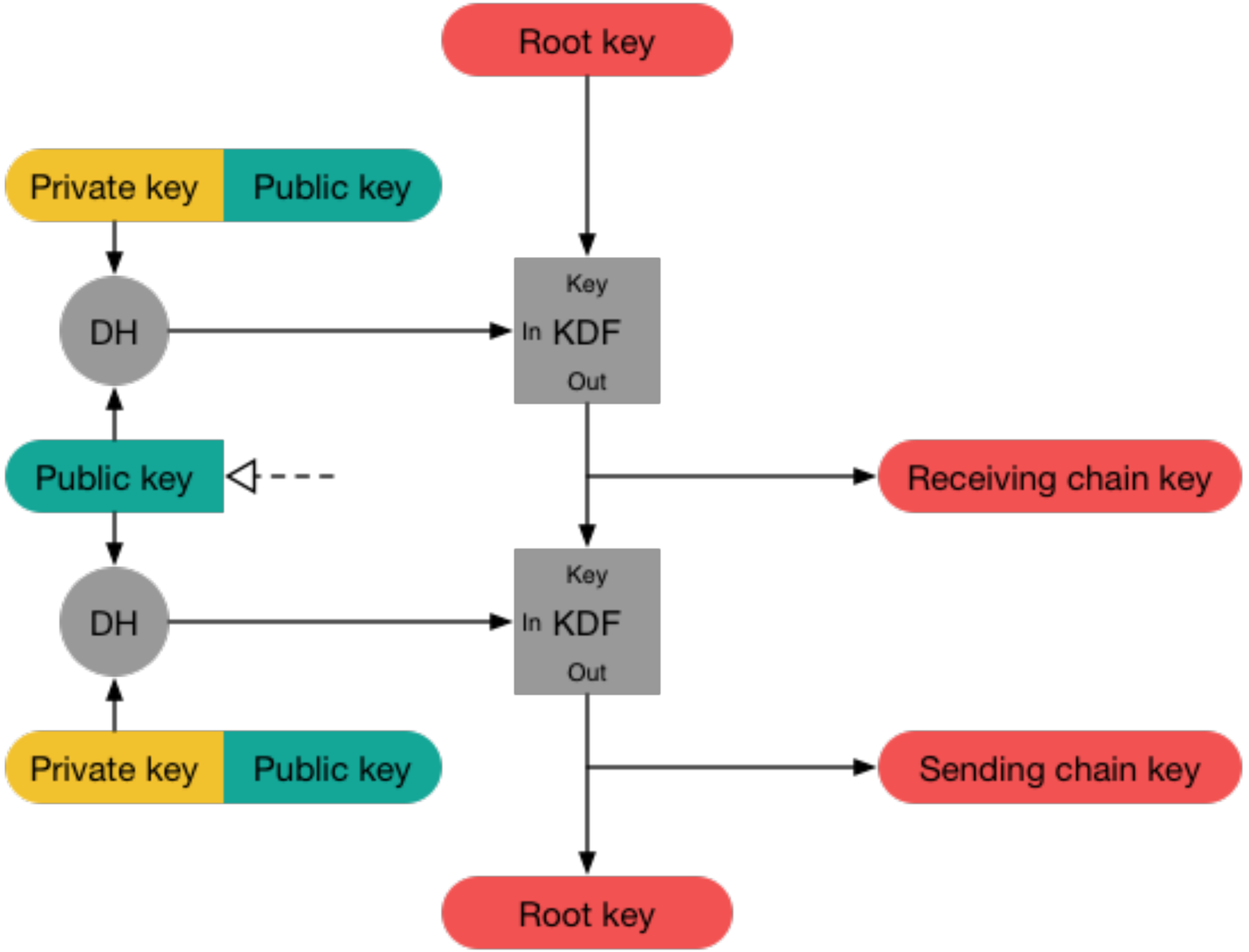
Public ratchet seeds symmetric ratchets (one per direction)



Source: https://whispersystems.org/docs/specifications/doubleratchet/

Add a third chain to improve post-compromise secrecy

How do we start this process? How does Alice learn Bob's first D-H message if Bob is offline?



Source: https://whispersystems.org/docs/specifications/doubleratchet/

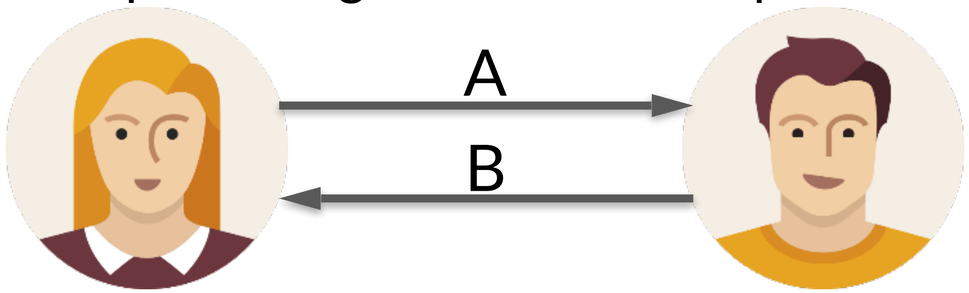
Putting everything together

Deniable

Fwd/back secure

Choose a ← [q]
Compute A = g^a

Choose b ← [q]
Compute B = g^b



A

B

Output B^a

Output A^b

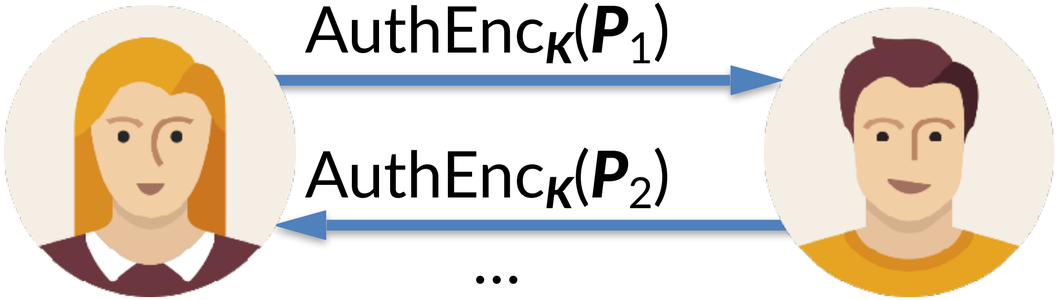
Ephemeral secret

Evolve public key

K = Hash(B^a)

K = Hash(A^b)

AuthEnc is deniable



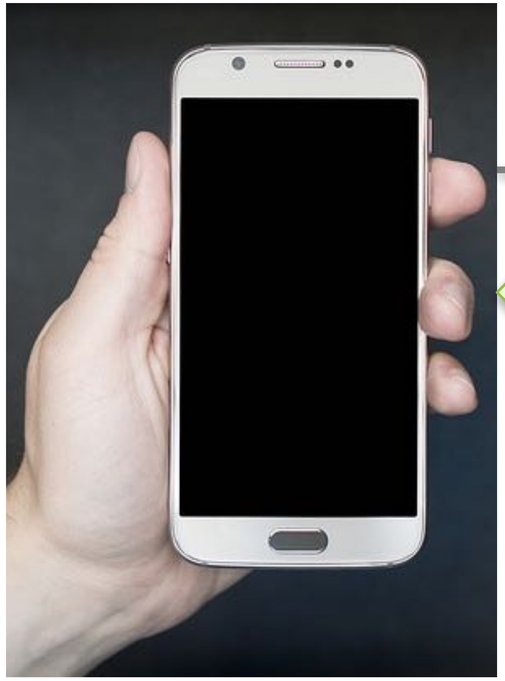
$\text{AuthEnc}_K(P_1)$

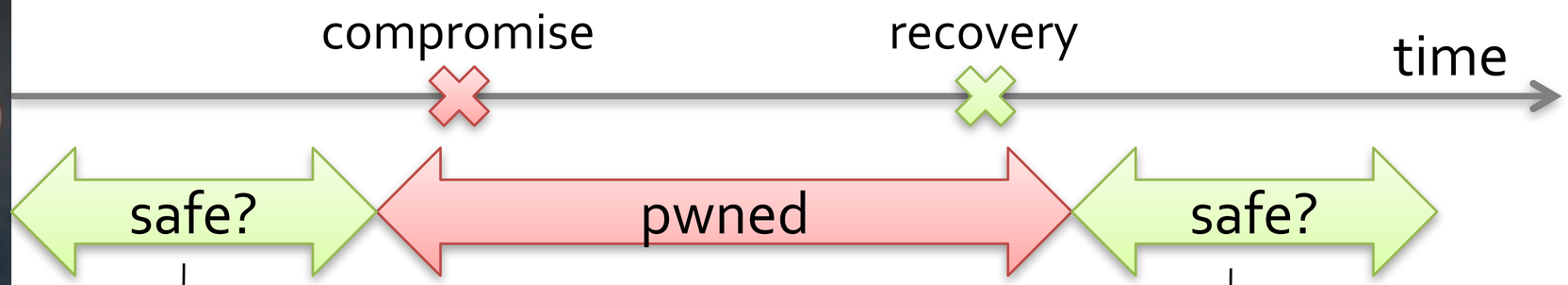
$\text{AuthEnc}_K(P_2)$

…

Evolve symm key

Why Signal provides forward and backward secrecy





"The parties derive new keys for every Double Ratchet message so that earlier keys cannot be calculated from later ones."

"The parties also send Diffie-Hellman public values attached to their messages. The results of Diffie-Hellman calculations are mixed into the derived keys so that later keys cannot be calculated from earlier ones."

Quotes from https://signal.org/docs/specifications/doubleratchet/

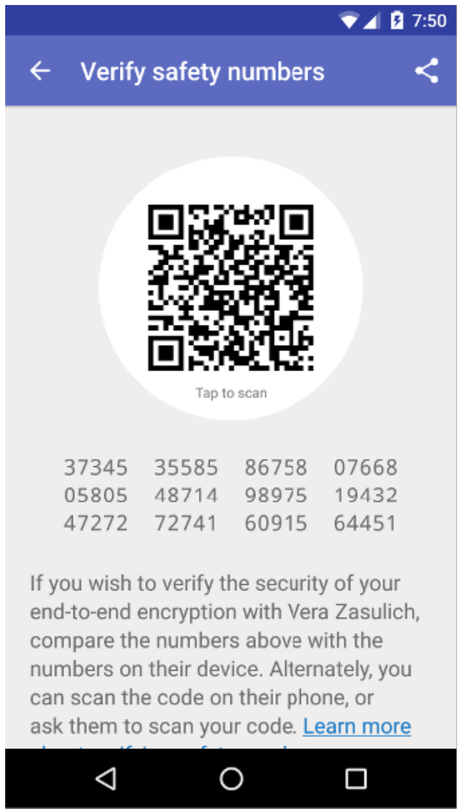
Ephemeral utopia

No long-term keys ⇒ great forward secrecy

- Message key used to AuthEnc a message is used once and tossed
- Chain key used to construct msg key is refreshed in each public ratchet
- Diffie-Hellman key pairs chosen ephemerally in each public ratchet

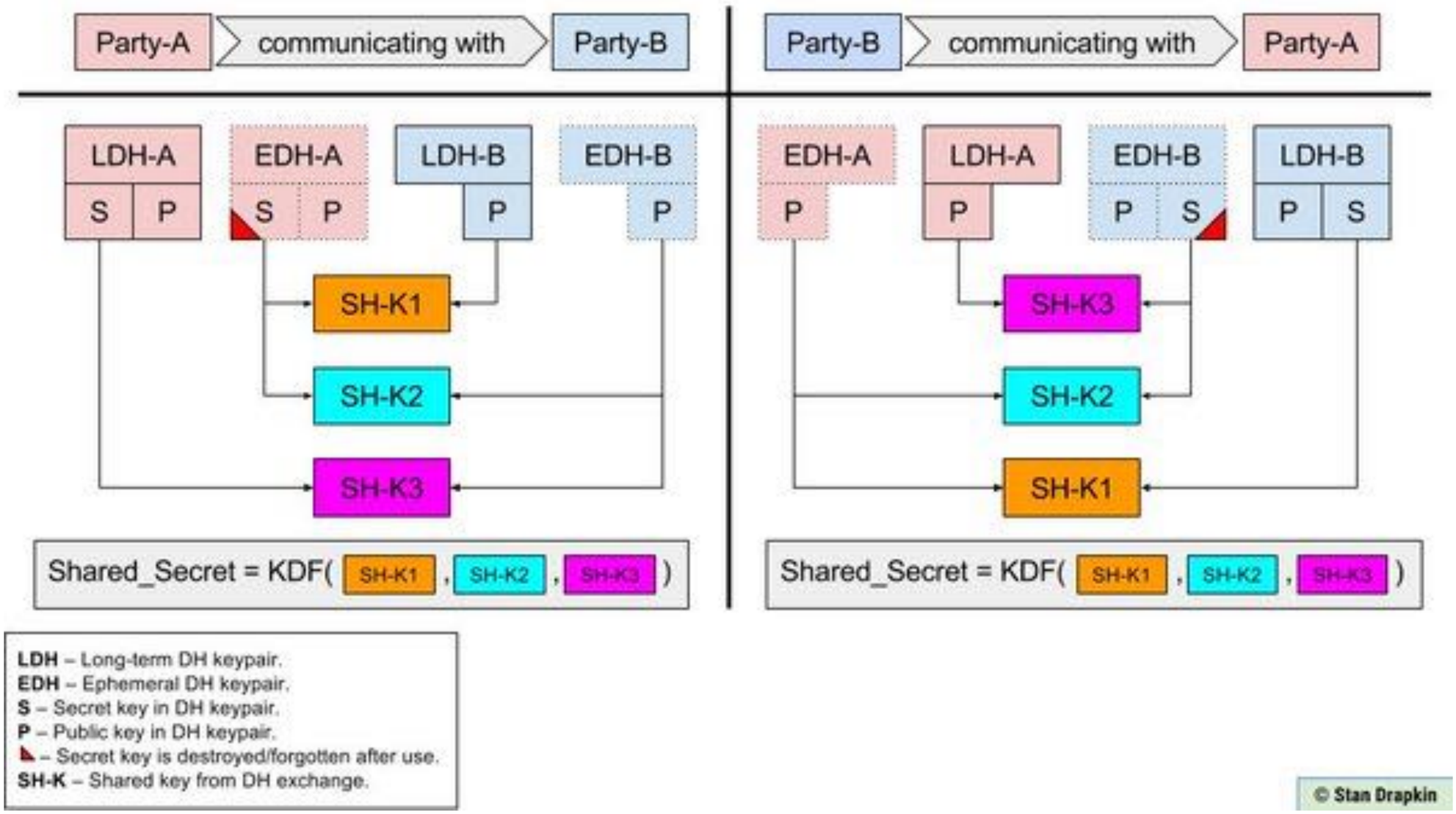
Wait… actually, is this a utopia or a dystopia?

- If you don't have *any* long-term state, then who are you?!
- Resolution: Also have a long-term key, Signal maintains a PKI



Source: https://whispersystems.org/blog/safety-number-updates/

Solution: a more involved *Triple-DH protocol*



LDH – Long-term DH keypair.
EDH – Ephemeral DH keypair.
S – Secret key in DH keypair.
P – Public key in DH keypair.
◣ – Secret key is destroyed/forgotten after use.
SH-K – Shared key from DH exchange.

SECURE MESSAGING APPS COMPARISON

BECAUSE PRIVACY MATTERS

App name	Allo	iMessage	Messenger	Signal	Skype	Telegram	Threema	Viber	Whatsapp	Wickr	Wire
TL;DR: Does the app secure my messages and attachments?	No	No	No	Yes	No	No	Yes	No	No	No	Yes

Source: https://www.securemessagingapps.com

Thinking About What You Need In A Secure Messenger

BY GENNIE GEBHART | MARCH 28, 2018

Source: https://www.eff.org/deeplinks/2018/03/thinking-about-what-you-need-secure-messenger