CS459/659 Privacy, Cryptography, Network and Data Security

Secure Messaging

Fall 2026

Secure Messaging Goals

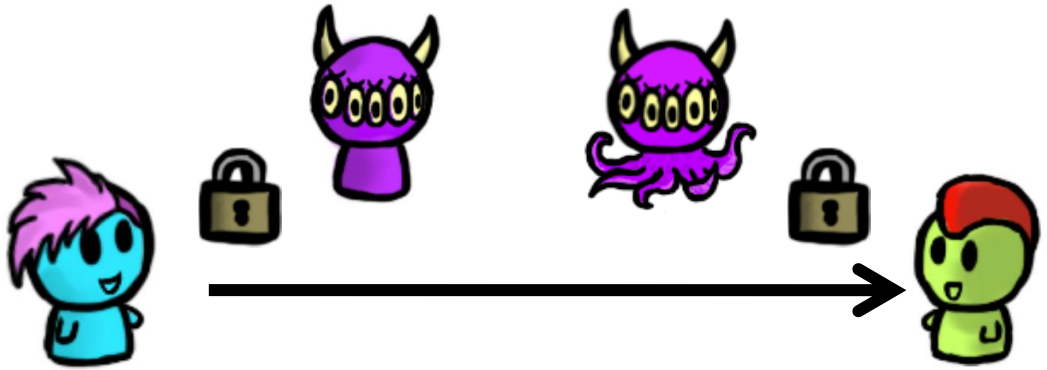
Secure Messaging Goals

- **Confidentiality:** Only Alice and Bob can read the message
- **Integrity:** Bob knows Mallory has not tampered with the message (and that it has not been corrupted)
- **Authenticity:** Bob knows Alice wrote the message
  - Non-repudiation?



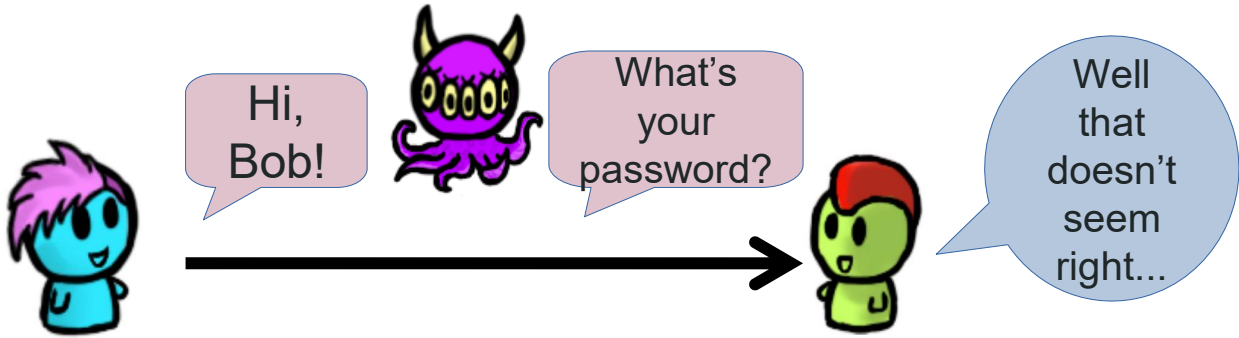
Secure Messaging Goals

- **Confidentiality:** Only Alice and Bob can read the message
- **Integrity:** Bob knows Mallory has not tampered with the message (and that it has not been corrupted)
- **Authenticity:** Bob knows Alice wrote the message
  - Non-repudiation?



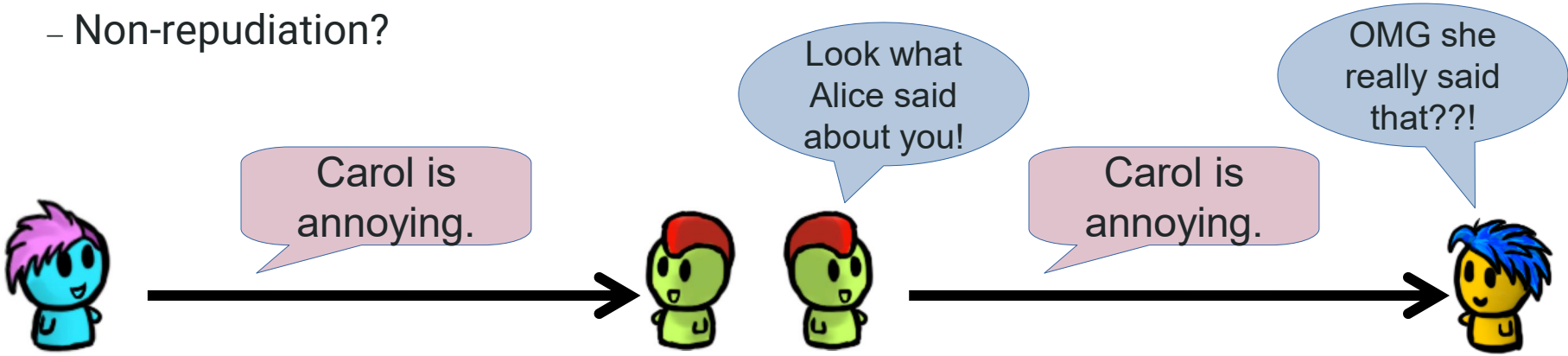
Secure Messaging Goals

- **Confidentiality:** Only Alice and Bob can read the message
- **Integrity:** Bob knows Mallory has not tampered with the message (and that it has not been corrupted)
- **Authenticity:** Bob knows Alice wrote the message
  - Non-repudiation?



Secure Messaging Goals

- **Confidentiality:** Only Alice and Bob can read the message
- **Integrity:** Bob knows Mallory has not tampered with the message (and that it has not been corrupted)
- **Authenticity:** Bob knows Alice wrote the message
  - Non-repudiation?



Pretty Good Privacy

A bit of history on PGP

- Hybrid encryption tool used for email (and other uses)
- Created by Phil Zimmermann in 1991
  - In 1993, Zimmermann was investigated for violating US export regulations, as PGP encryption exceeded 40-bit key size – **PGP was classified as a munition**.
  - In 1995, Zimmermann published PGP's code in a **book**, using First Amendment protections for printed materials.
  - Courts later ruled that cryptographic software source code is protected speech under the First Amendment.
  - US export controls on cryptography were eased in the late 1990s and, since 2000 PGP can be exported
    - (with some restrictions – certain countries/groups are barred)

– https://www.philzimmermann.com/EN/essays/WhyIWrotePGP.html

What do you mean by “PGP”?

- **PGP:** Pretty Good Privacy (original program)
- **OpenPGP:** Open standard (RFC 4880)
- **GPG/GnuPG:** GNU Privacy Guard (a popular OpenPGP program)
- Today, many programs implement OpenPGP
  - Thunderbird, Evolution, Mailvelope, OpenKeychain, Delta Chat, Proton Mail, ...



PGP is a hybrid crypto scheme!

- To send a message to Bob, Alice will:
  - ❑ Write a message
  - ❑ Sign a hash of the message with her own signature key
  - ❑ Encrypt both the message and the signature with a symmetric key (C1)
  - ❑ Encrypt the symmetric key with Bob's public encryption key (C2)
- Bob receives the ciphertext and:
  - ❑ Decrypts C2 using his private decryption key to yield the symmetric key
  - ❑ Decrypts C1 using the symmetric key to yield the message and the signature
  - ❑ Uses Alice's verification key to check the signature

PGP is a hybrid crypto scheme!



Msg



PGP is a hybrid crypto scheme!





Msg

1. Sign (hash(Msg)) = sig

PGP is a hybrid crypto scheme!





Msg

1. Sign (hash(Msg)) = sig



2. Enc (sig Msg) = C1



PGP is a hybrid crypto scheme!





Msg

1. Sign (hash(Msg)) = sig



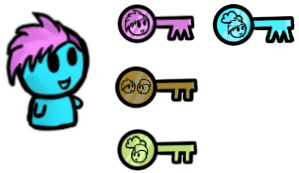
2. Enc (sig Msg) = C1



3. Enc () = C2



PGP is a hybrid crypto scheme!





Msg

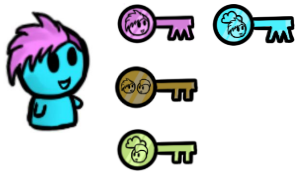
1. Sign (hash(Msg)) = sig

2. Enc (sig Msg) = C1

3. Enc ( ) = C2

C1 C2

PGP is a hybrid crypto scheme!





Msg

1. Sign (hash(Msg)) = sig

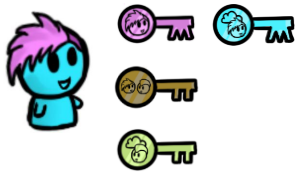
2. Enc (sig Msg) = C1

3. Enc ( ) = C2

4. Dec (C2) =

C1 C2

PGP is a hybrid crypto scheme!





Msg

1. Sign (hash(Msg)) = sig

2. Enc (sig Msg) = C1

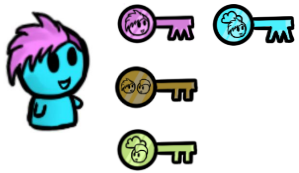
3. Enc ( ) = C2

C1 C2

4. Dec (C2) =

5. Dec (C1) = sig Msg

PGP is a hybrid crypto scheme!





Msg

1. Sign (hash(Msg)) = sig

2. Enc (sig Msg) = C1

3. Enc ( ) = C2

C1 C2

4. Dec (C2) =

5. Dec (C1) = sig Msg

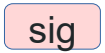
6. Ver (sig , Msg) =

Encrypted Messaging Goals and PGP

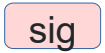
- Confidentiality



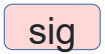
- Integrity



- Authenticity



  - Non-repudiation

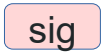


Encrypted Messaging Goals and PGP

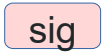
- Confidentiality



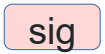
- Integrity



- Authenticity



  - Non-repudiation

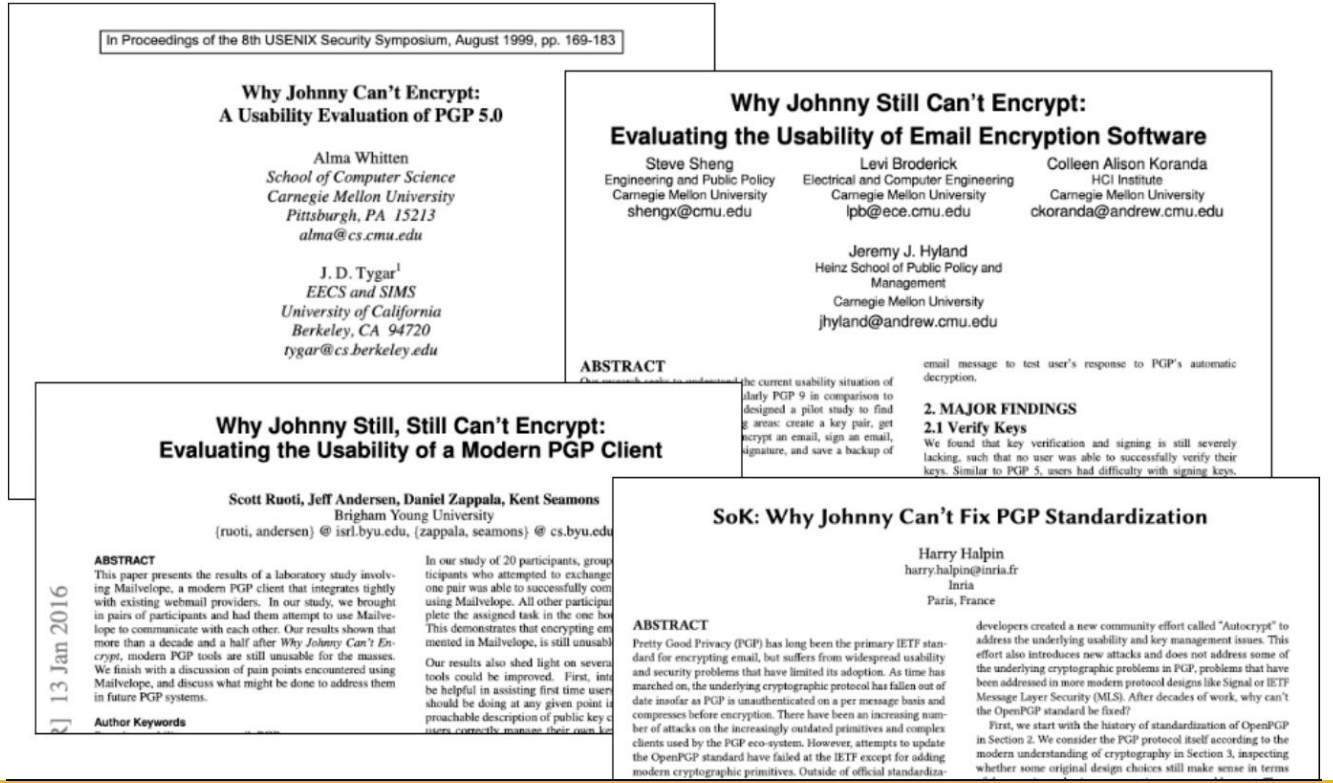




Problems with PGP

Problem #1: Usability

- Hard to use
- Low adoption



In Proceedings of the 8th USENIX Security Symposium, August 1999, pp. 169-183

Why Johnny Can't Encrypt:
A Usability Evaluation of PGP 5.0

Alma Whitten
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
alma@cs.cmu.edu

J. D. Tygar[1]
EECS and SIMS
University of California
Berkeley, CA 94720
tygar@cs.berkeley.edu

Why Johnny Still Can't Encrypt:
Evaluating the Usability of Email Encryption Software

Steve Sheng
Engineering and Public Policy
Carnegie Mellon University
shengx@cmu.edu

Levi Broderick
Electrical and Computer Engineering
Carnegie Mellon University
lpb@ece.cmu.edu

Colleen Alison Koranda
HCI Institute
Carnegie Mellon University
ckoranda@andrew.cmu.edu

Jeremy J. Hyland
Heinz School of Public Policy and Management
Carnegie Mellon University
jhyland@andrew.cmu.edu

ABSTRACT

email message to test user's response to PGP's automatic decryption.

2. MAJOR FINDINGS

2.1 Verify Keys

We found that key verification and signing is still severely lacking, such that no user was able to successfully verify their keys. Similar to PGP 5, users had difficulty with signing keys.

Why Johnny Still, Still Can't Encrypt:
Evaluating the Usability of a Modern PGP Client

Scott Ruoti, Jeff Andersen, Daniel Zappala, Kent Seamons
Brigham Young University
{ruoti, andersen} @ isrl.byu.edu, {zappala, seamons} @ cs.byu.edu

ABSTRACT

This paper presents the results of a laboratory study involving Mailvelope, a modern PGP client that integrates tightly with existing webmail providers. In our study, we brought in pairs of participants and had them attempt to use Mailvelope to communicate with each other. Our results shown that more than a decade and a half after *Why Johnny Can't Encrypt*, modern PGP tools are still unusable for the masses. We finish with a discussion of pain points encountered using Mailvelope, and discuss what might be done to address them in future PGP systems.

Author Keywords

13 Jan 2016

SoK: Why Johnny Can't Fix PGP Standardization

Harry Halpin
harry.halpin@inria.fr
Inria
Paris, France

ABSTRACT

Pretty Good Privacy (PGP) has long been the primary IETF standard for encrypting email, but suffers from widespread usability and security problems that have limited its adoption. As time has marched on, the underlying cryptographic protocol has fallen out of date insofar as PGP is unauthenticated on a per message basis and compresses before encryption. There have been an increasing number of attacks on the increasingly outdated primitives and complex clients used by the PGP eco-system. However, attempts to update the OpenPGP standard have failed at the IETF except for adding modern cryptographic primitives. Outside of official standardiza-

developers created a new community effort called "Autocrypt" to address the underlying usability and key management issues. This effort also introduces new attacks and does not address some of the underlying cryptographic problems in PGP, problems that have been addressed in more modern protocol designs like Signal or IETF Message Layer Security (MLS). After decades of work, why can't the OpenPGP standard be fixed?

First, we start with the history of standardization of OpenPGP in Section 2. We consider the PGP protocol itself according to the modern understanding of cryptography in Section 3, inspecting whether some original design choices still make sense in terms

Problem #1: Usability

- https://moxie.org/2015/02/24/gpg-and-me.html

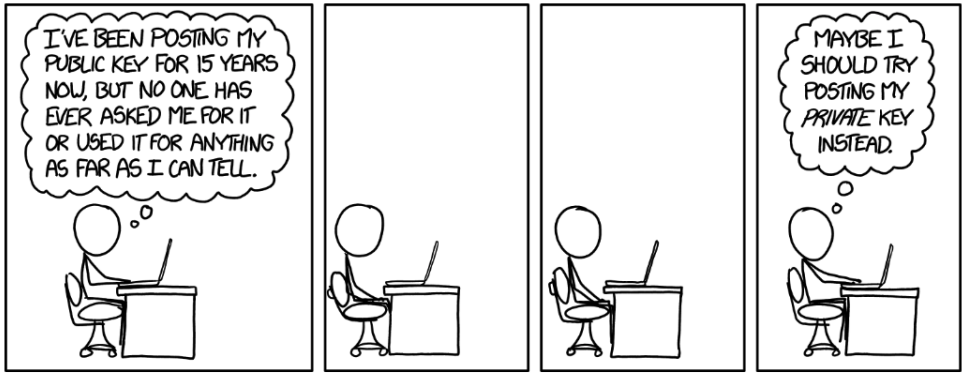
–"When I receive a *GPG encrypted* email from a stranger, though, I immediately get the feeling that I don't want to read it. [...] Eventually I realized that when I receive a GPG encrypted email, it simply means that the email was written by *someone who would voluntarily use GPG.*"



https://xkcd.com/1181/

Problem #1: Usability

- Usability is a security parameter
  - If it's hard to use, people will not use it
  - If it's hard to use **properly**, people will use it, but in **insecure** ways



(Public Key)

Problem #2: Lack of Forward Secrecy

- Alice sends many encrypted messages to Bob
  - Possibly over the course of months, years
- Suppose Eve saves all of them
  - Not so unreasonable if Eve runs the email server
- What if Eve steals Bob's private key?
  - She can decrypt all messages sent to him. **Past, present, and future...**

Problem #3: Non-repudiation

- Why non-repudiation?
- Good for contracts, not private emails
- Casual conversations are "off-the-record"
  - Alice and Bob talk in private
  - No one else can hear
  - No one else knows what they say
  - No one can prove what was said
    - Not even Alice or Bob



Off-The-Record (OTR) Messaging

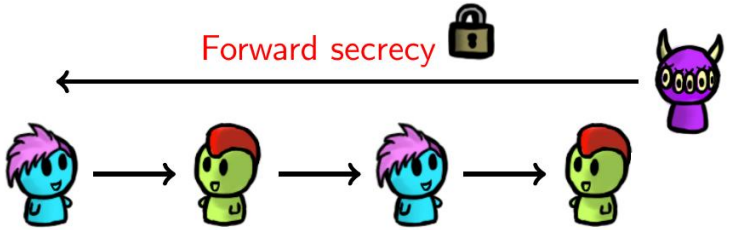
OTR

- Messaging extension for encryption with:
  - Forward secrecy
  - Post-compromise security
  - Repudiation (deniability)



Goals of Off-The-Record Messaging

- **Forward secrecy:** a key compromise does not reveal past communication



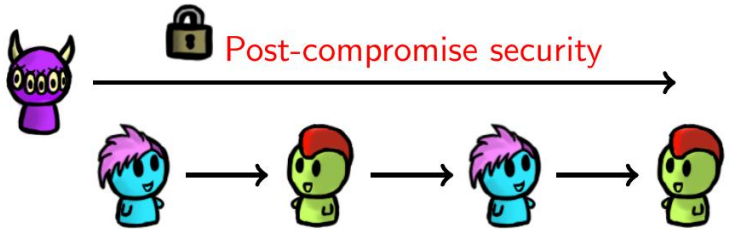
AKA
- Perfect Forward Secrecy (PFS)

Goals of Off-The-Record Messaging

- **Forward secrecy:** a key compromise does not reveal past communication

- **Post-compromise security:** a key compromise does not reveal future communication

AKA
- Backward secrecy
- Future secrecy
- Self-healing
- Break-in recovery

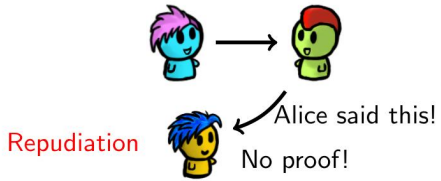


Goals of Off-The-Record Messaging

- **Forward secrecy:** a key compromise does not reveal past communication

- **Post-compromise security:** a key compromise does not reveal future communication

- **Repudiation:** a participant cannot prove *to a third party* that another participant said something
  - We want repudiation AND authentication: **deniable authentication**

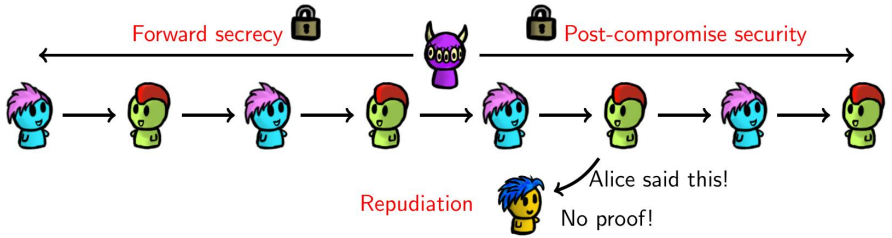


Goals of Off-The-Record Messaging

- **Forward secrecy:** a key compromise does not reveal past communication

- **Post-compromise security:** a key compromise does not reveal future communication

- **Repudiation:** a participant cannot prove *to a third party* that another participant said something



Forward Secrecy

- **Key compromise does not reveal past messages**

  Q: How can we accomplish that?

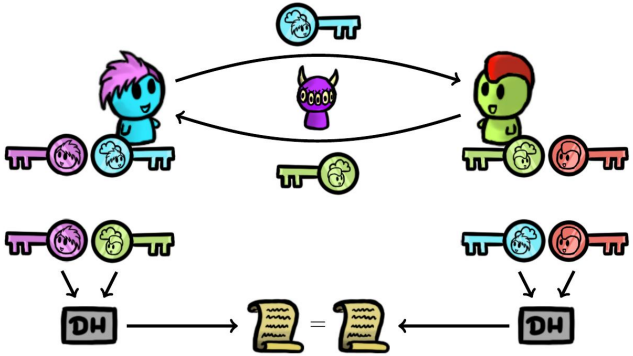
  Change the key!

  Old keys must be securely deleted



Forward Secrecy (one approach)

- Recall Diffie-Hellman...

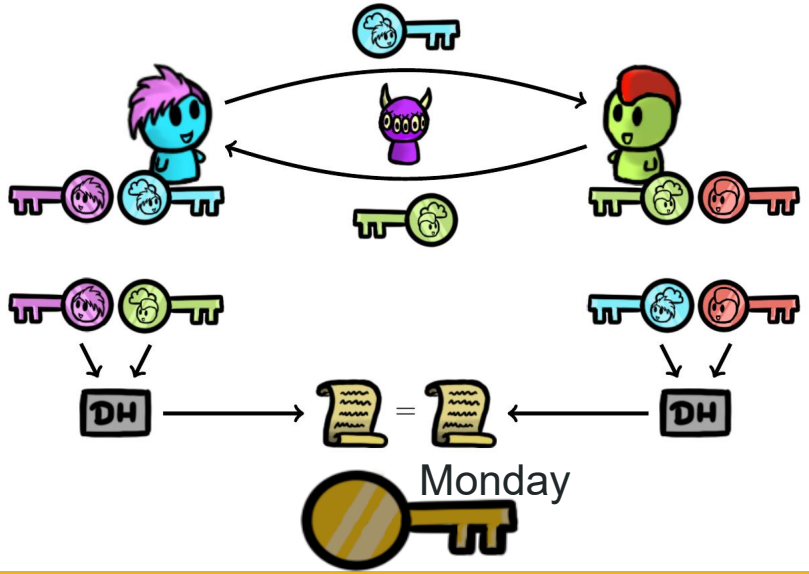


- Alice and Bob find a shared secret used to create a symmetric key

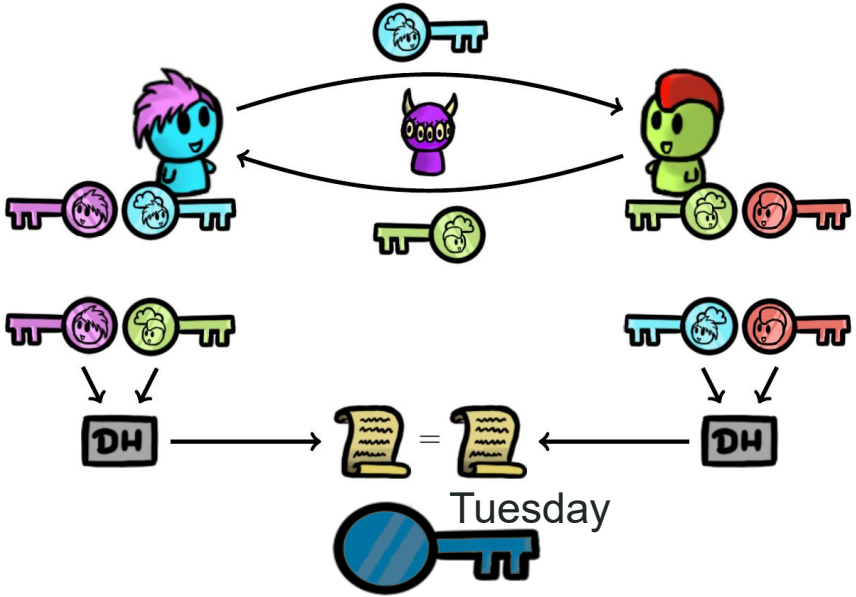
- This key can be used for ephemeral (temporary) communication “session”

Forward Secrecy (one approach)

- Alice and Bob talk on Monday...



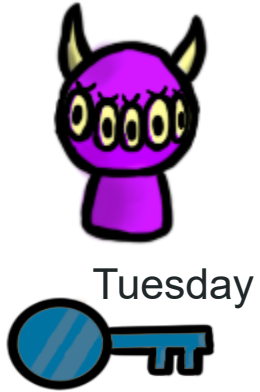
- Alice and Bob talk on Tuesday...



Forward Secrecy (one approach)

- Eve can compromise a session but not all past communication

- Problems?
  - Alice can't start a session unless Bob is online – **DH is interactive**
  - Eve can still compromise a whole session (which might last long…)



Forward Secrecy in OTR

- **Idea 1:** What if we make the sessions as short as possible?

- **Idea 2:** What if new session keys don’t have to be negotiated interactively?

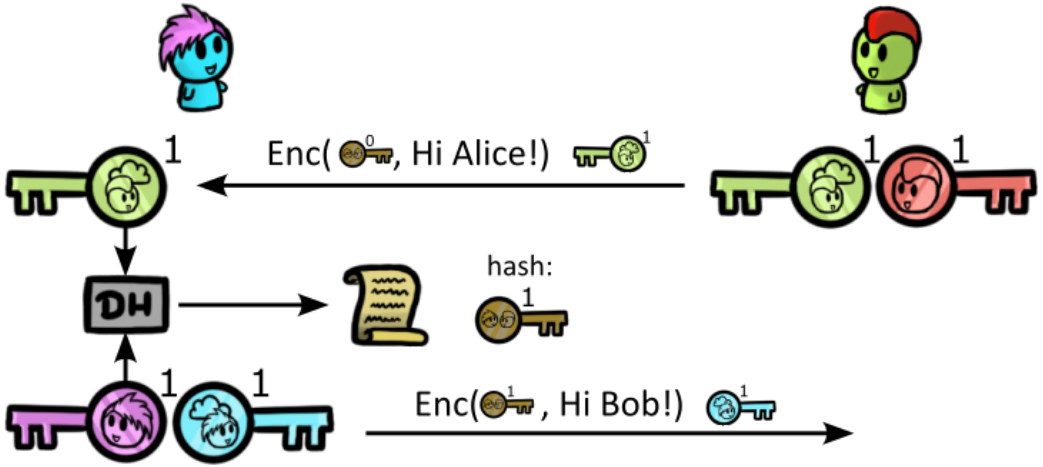
OTR's DH Ratchet (incomplete)

- Assume  is produced via DH between  and 
- In these slides, we use the notation Enc(Key, Message) for Encryption+MAC with a key.
  - (The MAC key is the hash of the encryption key.)



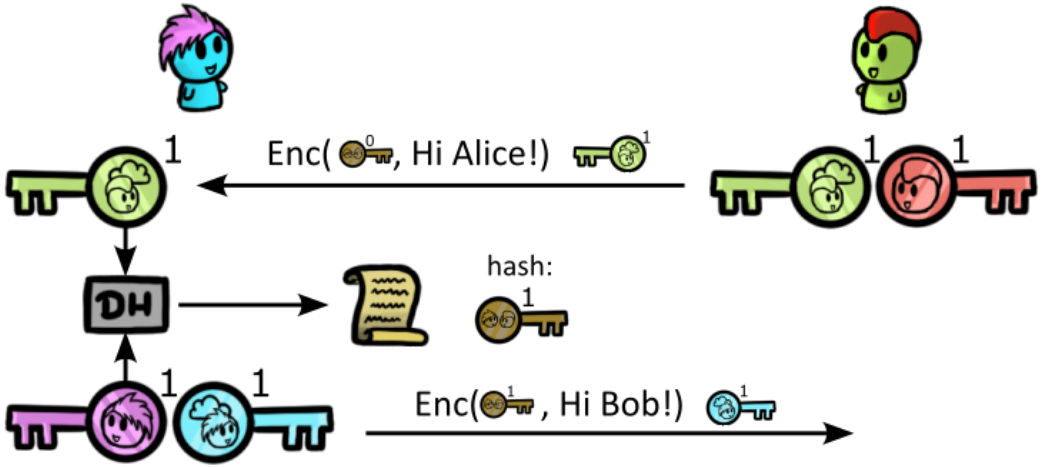
OTR's DH Ratchet (incomplete)

- Assume  is produced via DH between  and 
- In these slides, we use the notation Enc(Key, Message) for Encryption+MAC with a key.
  - (The MAC key is the hash of the encryption key.)



OTR's DH Ratchet (incomplete)

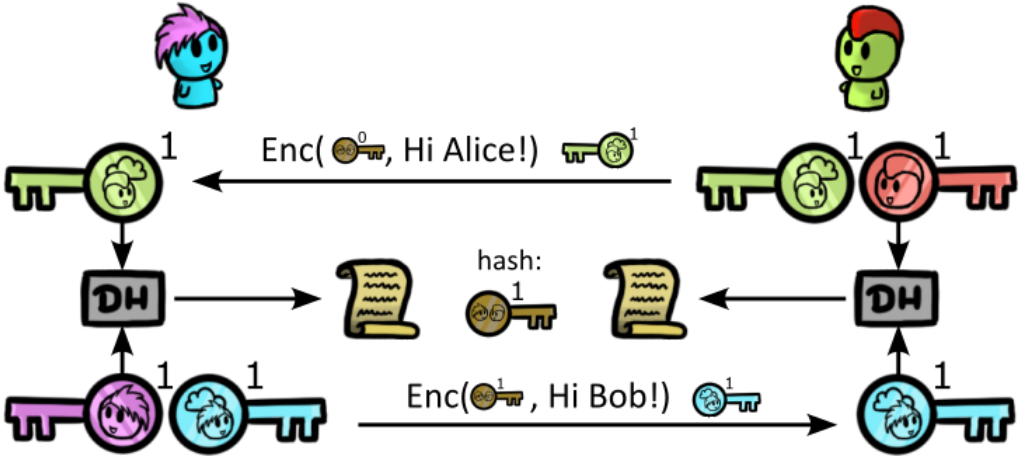
- Assume  is produced via DH between  and 
- In these slides, we use the notation Enc(Key, Message) for Encryption+MAC with a key.
  - (The MAC key is the hash of the encryption key.)



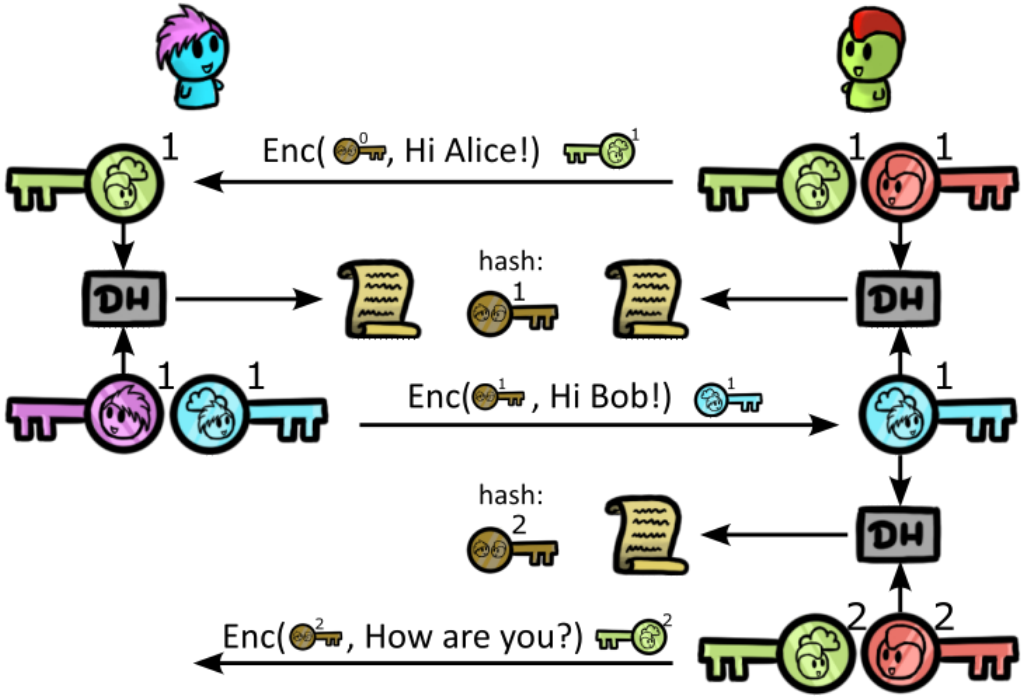
How does Bob recover the message?

OTR's DH Ratchet (incomplete)

- Assume  is produced via DH between  and 
- In these slides, we use the notation Enc(Key, Message) for Encryption+MAC with a key.
  - (The MAC key is the hash of the encryption key.)



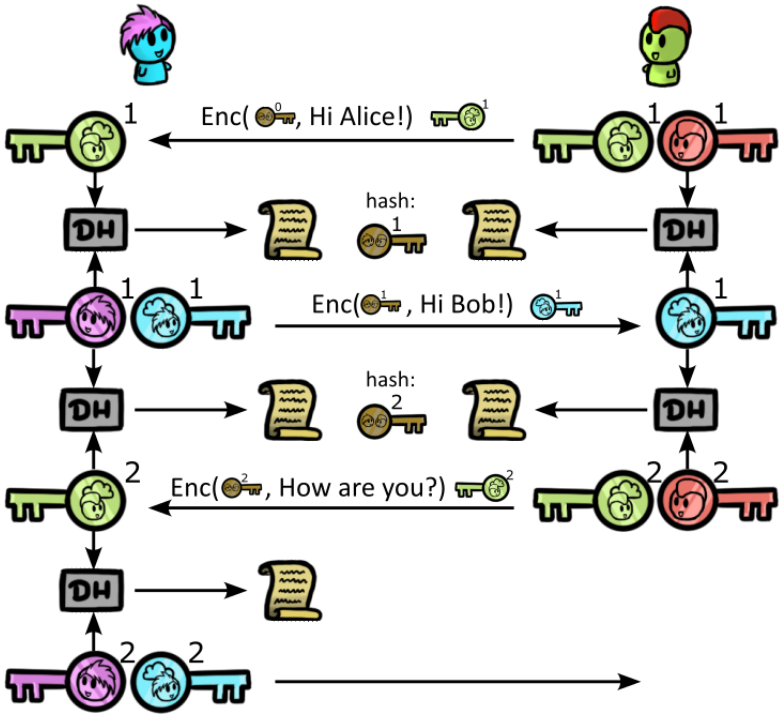
OTR's DH Ratchet (incomplete)



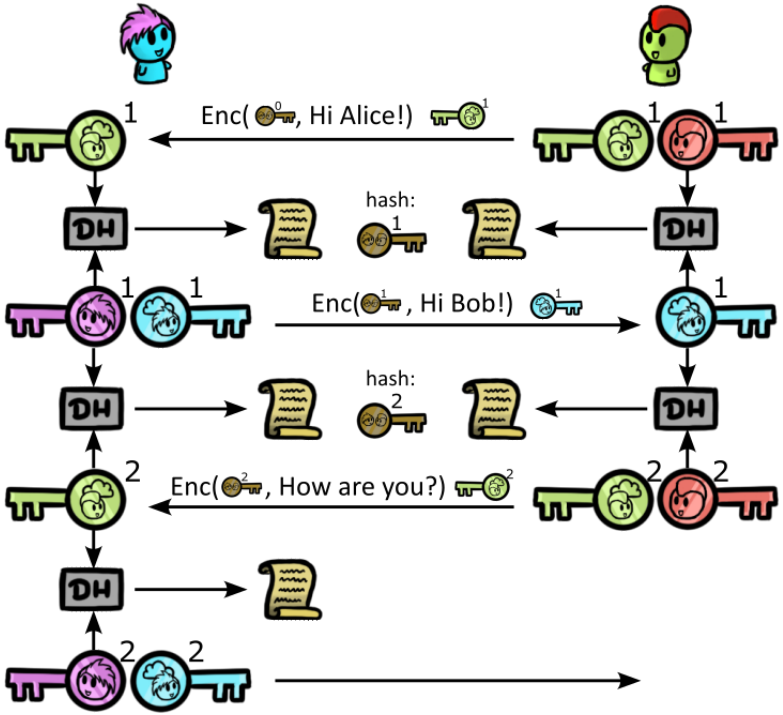
- Alice and Bob automatically create new sessions as they reply to each other
- Also provides post-compromise security
- Awesome! :)
- This is a "ratchet": You can't go backwards



OTR's DH Ratchet (incomplete)



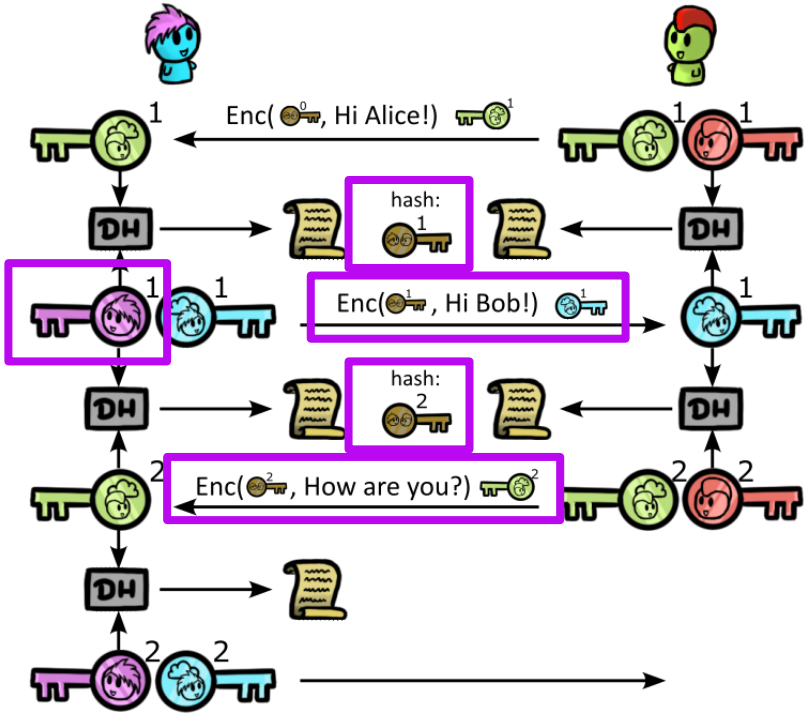
OTR's DH Ratchet (incomplete)



What happens if Eve learns a private key? E.g., Eve learns:



OTR's DH Ratchet (incomplete)

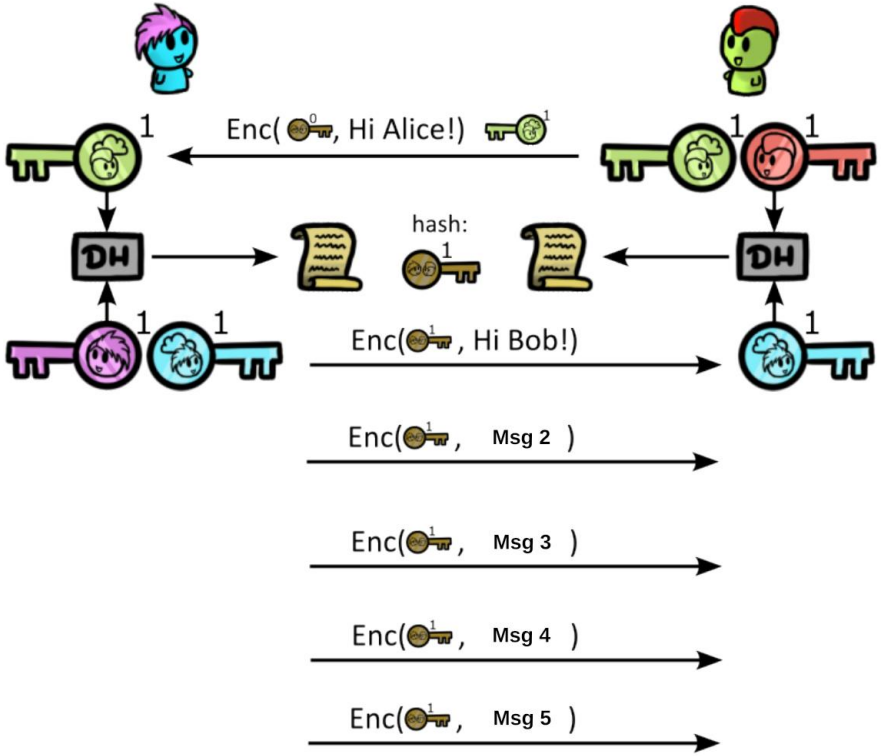


What happens if Eve learns a private key? E.g., Eve learns:



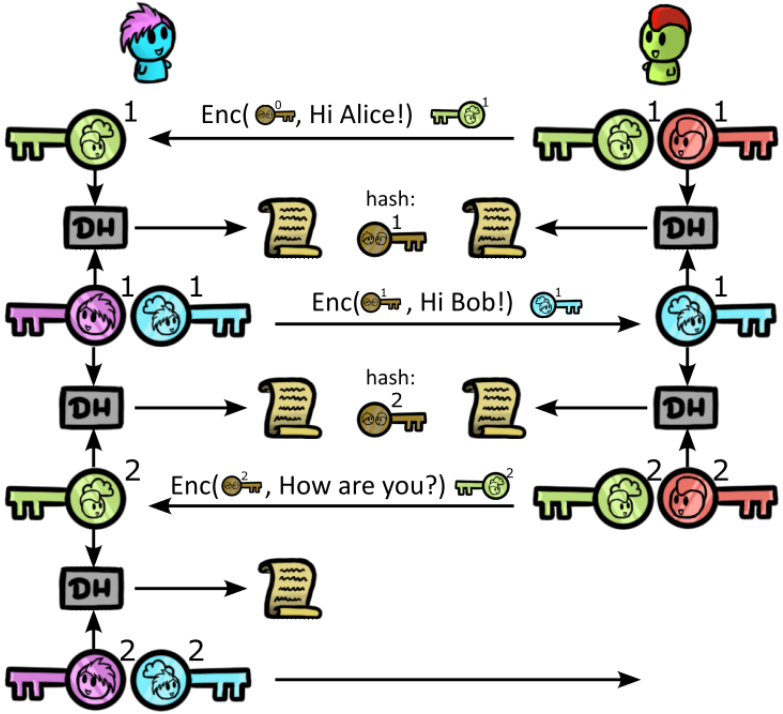
She can decrypt "Hi Bob!" and "How are you?"

OTR's DH Ratchet (incomplete)



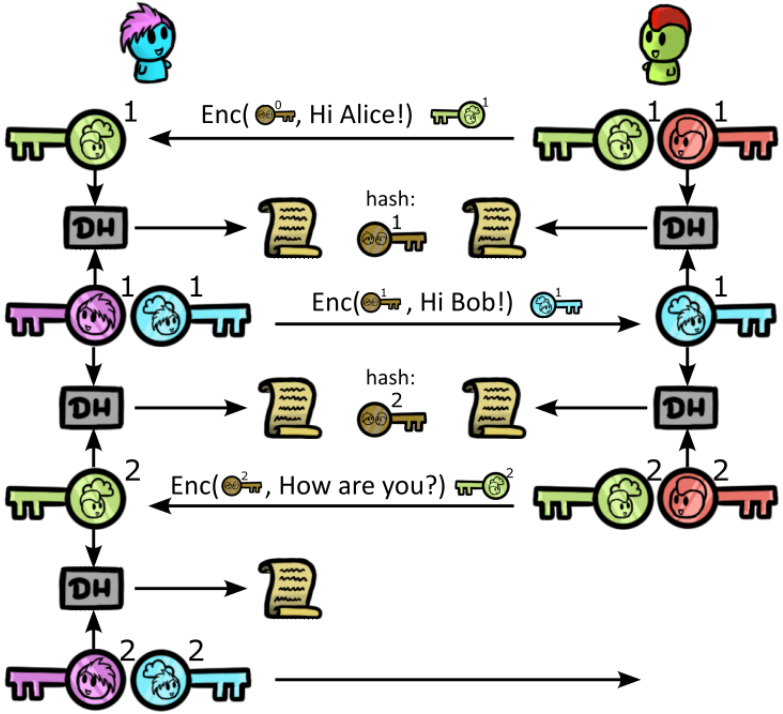
- Session keys only roll forward with interactive replies.
- If Alice sends multiple messages but Bob takes a long time to reply, multiple messages will get encrypted with the same key!
- OTR's forward secrecy is limited.

Deniable Authentication in OTR



Q: How can we get authentication without non-repudiation?

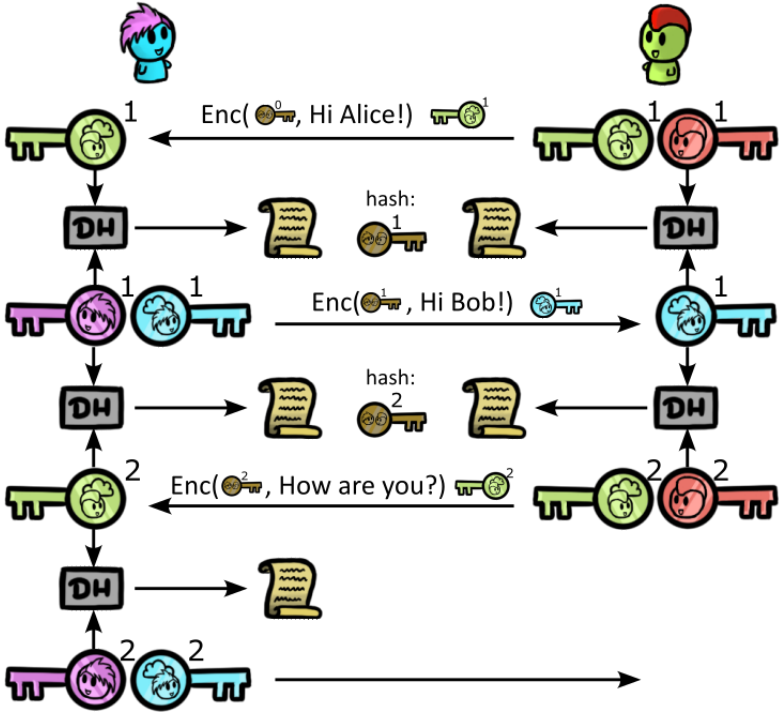
Deniable Authentication in OTR



Q: How can we get authentication without non-repudiation?
A: With a MAC!

Q: Why are MACs deniable?

Deniable Authentication in OTR



Q: How can we get authentication without non-repudiation?

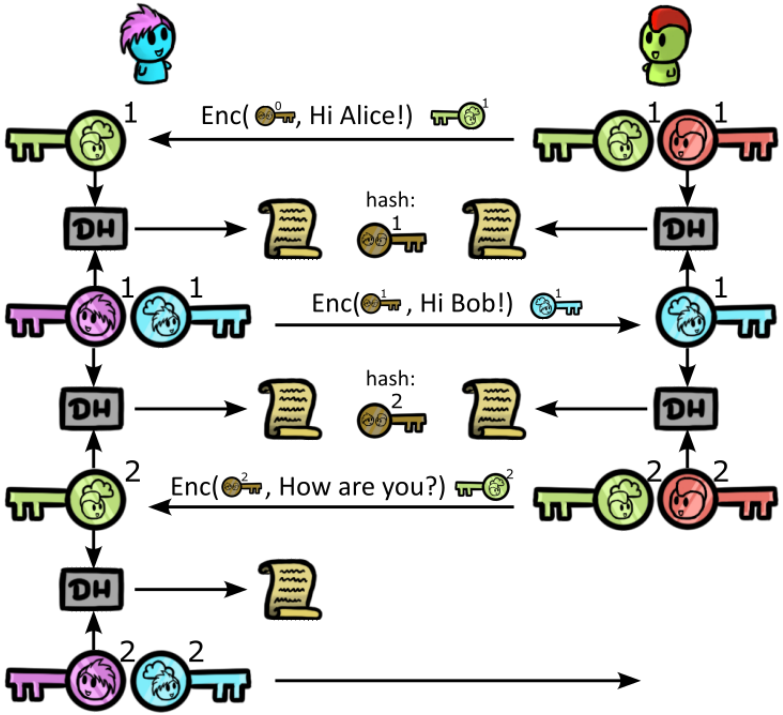
A: With a MAC!

Q: Why are MACs deniable?

A: Only Alice and Bob know K

- Alice sends Bob a message MACed with K
- Bob knows it was Alice because he did not produce the MAC himself

OTR's DH Ratchet (incomplete)

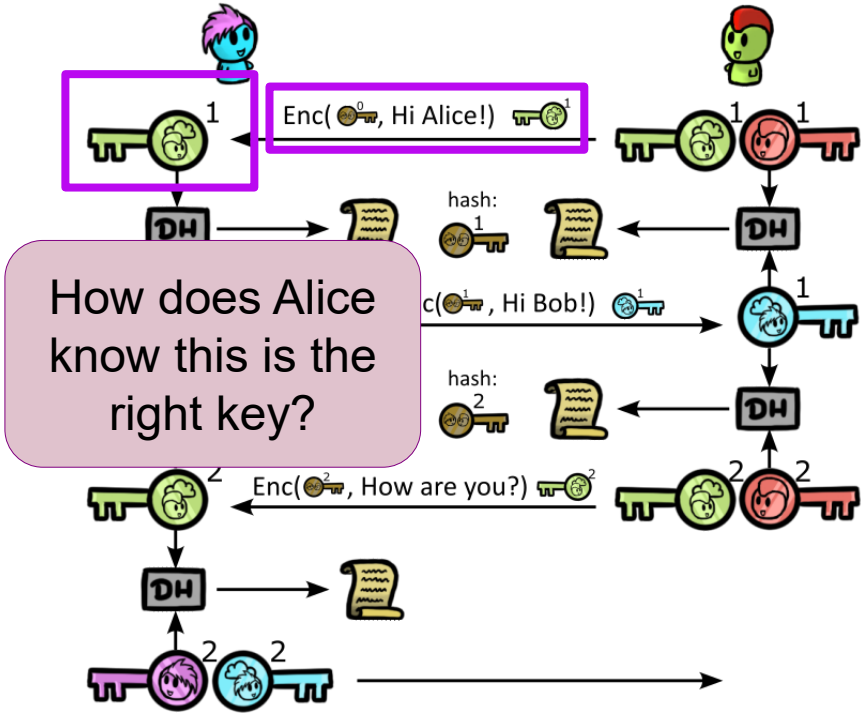


- The OTR DH Ratchet we saw in the previous slides is broken!

Can you spot the Mallory-in-the-Middle attack?



OTR's DH Ratchet (incomplete)



How does Alice know this is the right key?

- The OTR DH Ratchet we saw in the previous slides is broken!



Can you spot the Mallory-in-the-Middle attack?

What OTR actually does (from the OTR paper)

Alice and Bob securely exchange their public keys 

and 

and use DH to get 



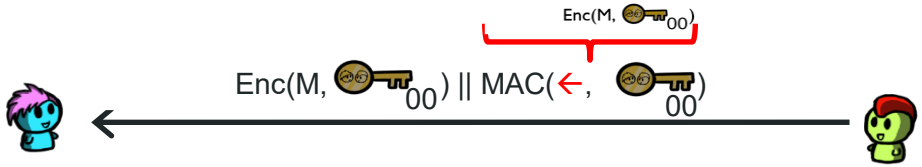


What OTR actually does (from the OTR paper)

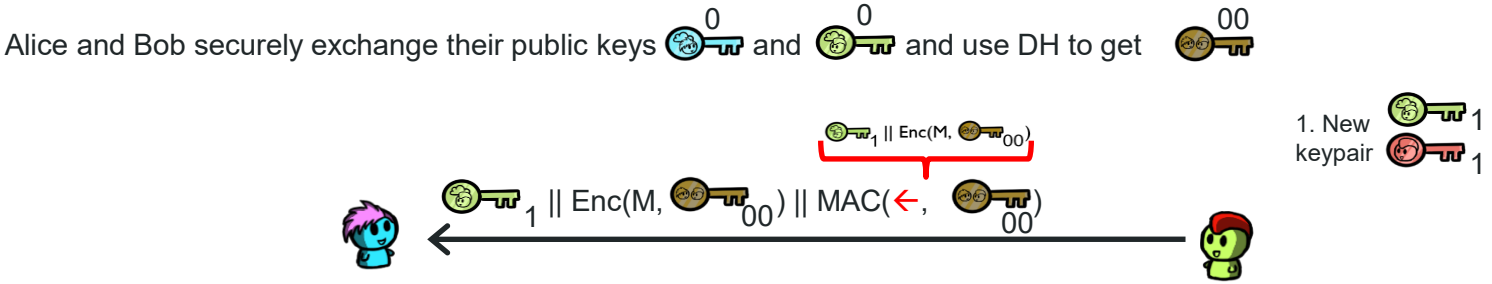
Alice and Bob securely exchange their public keys 

and 

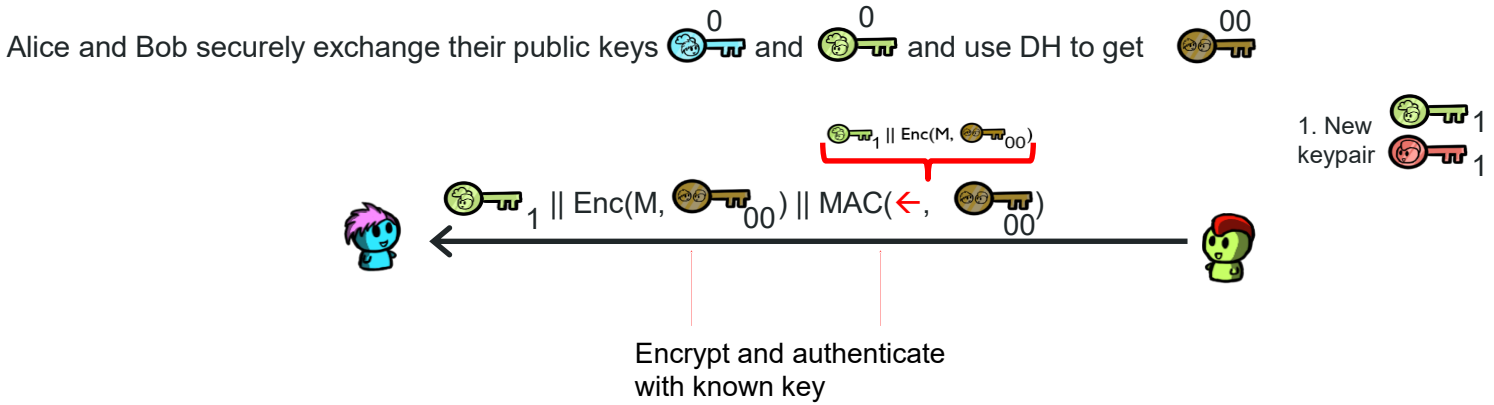
and use DH to get 



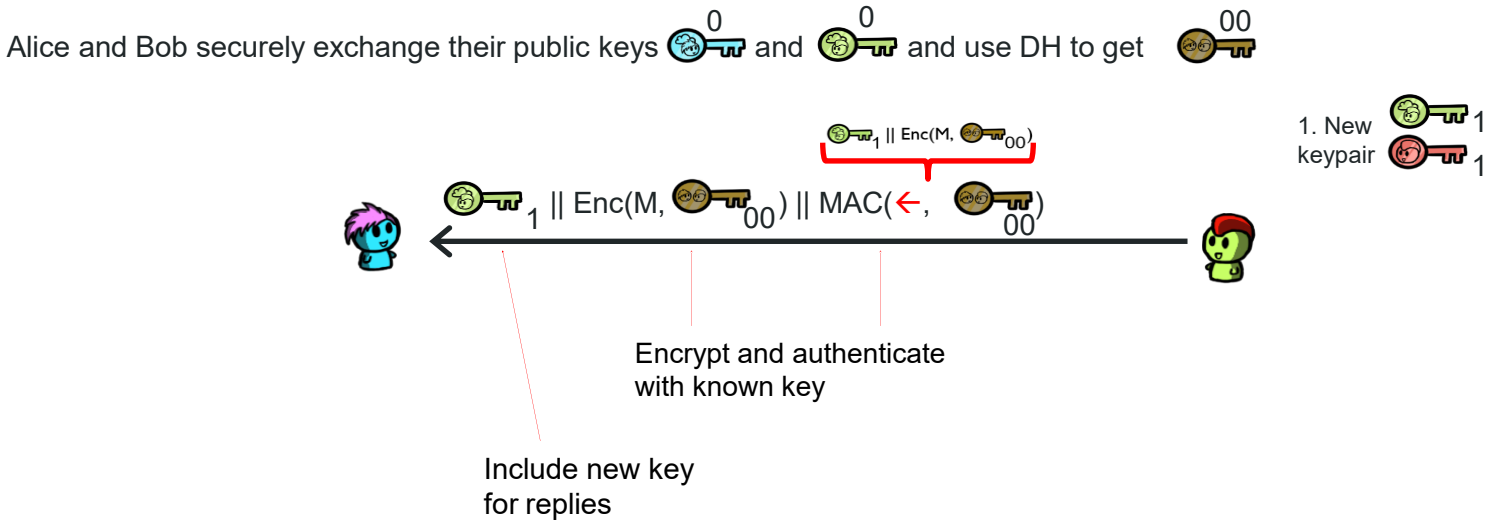
What OTR actually does (from the OTR paper)



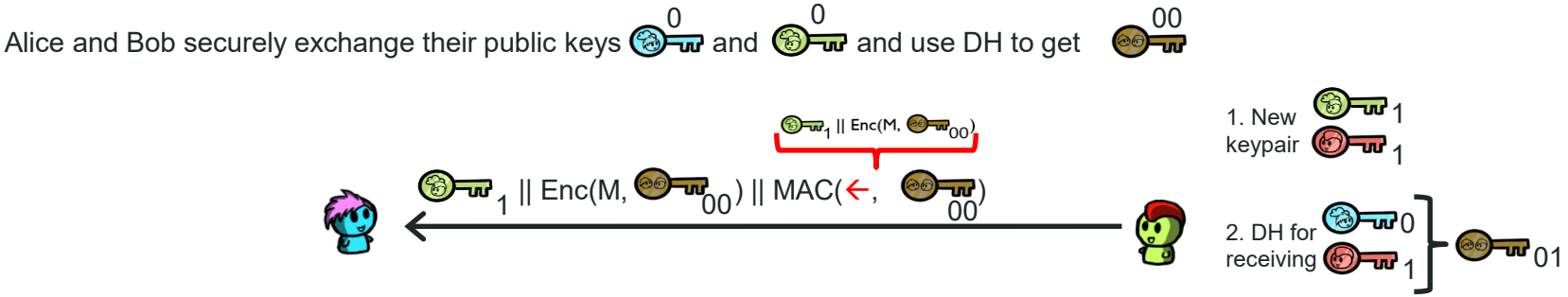
What OTR actually does (from the OTR paper)



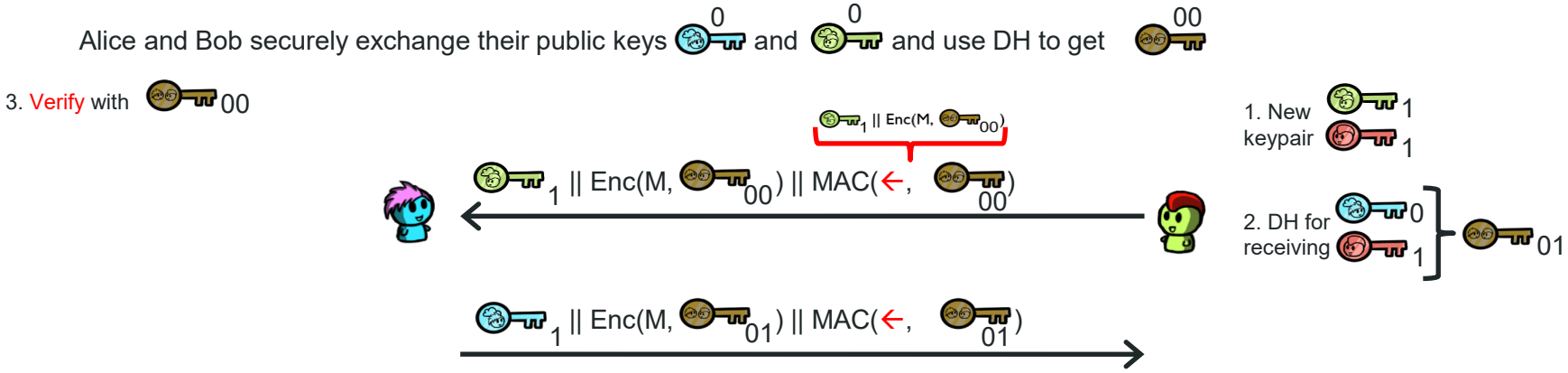
What OTR actually does (from the OTR paper)



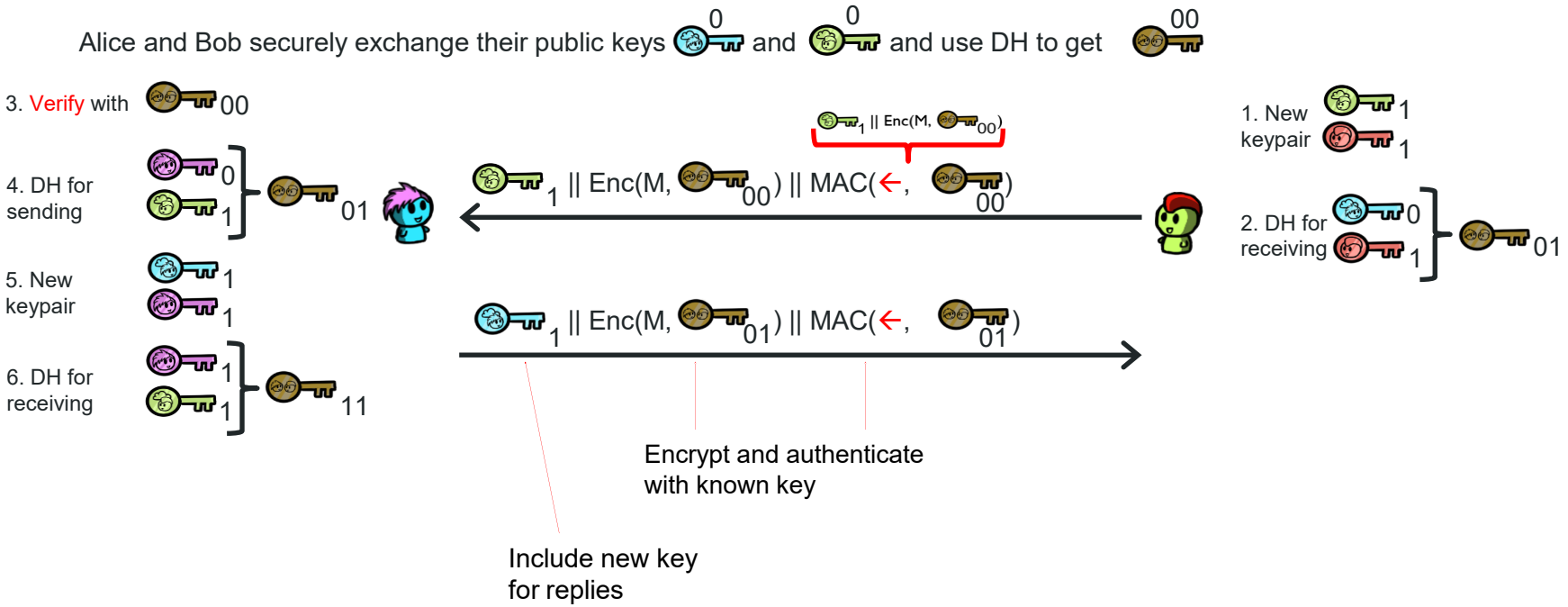
What OTR actually does (from the OTR paper)



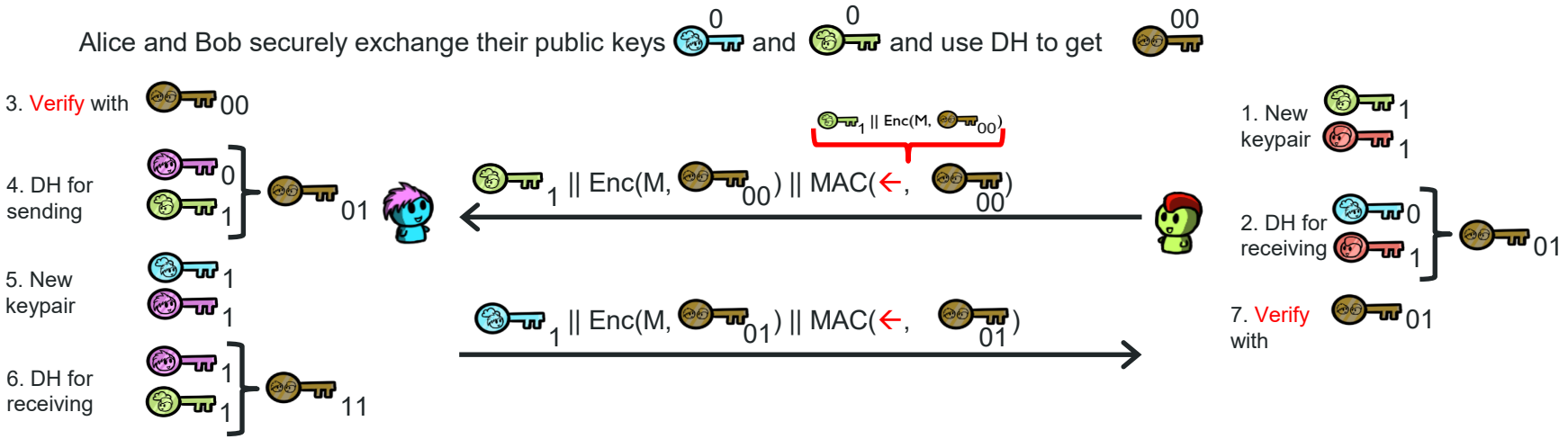
What OTR actually does (from the OTR paper)



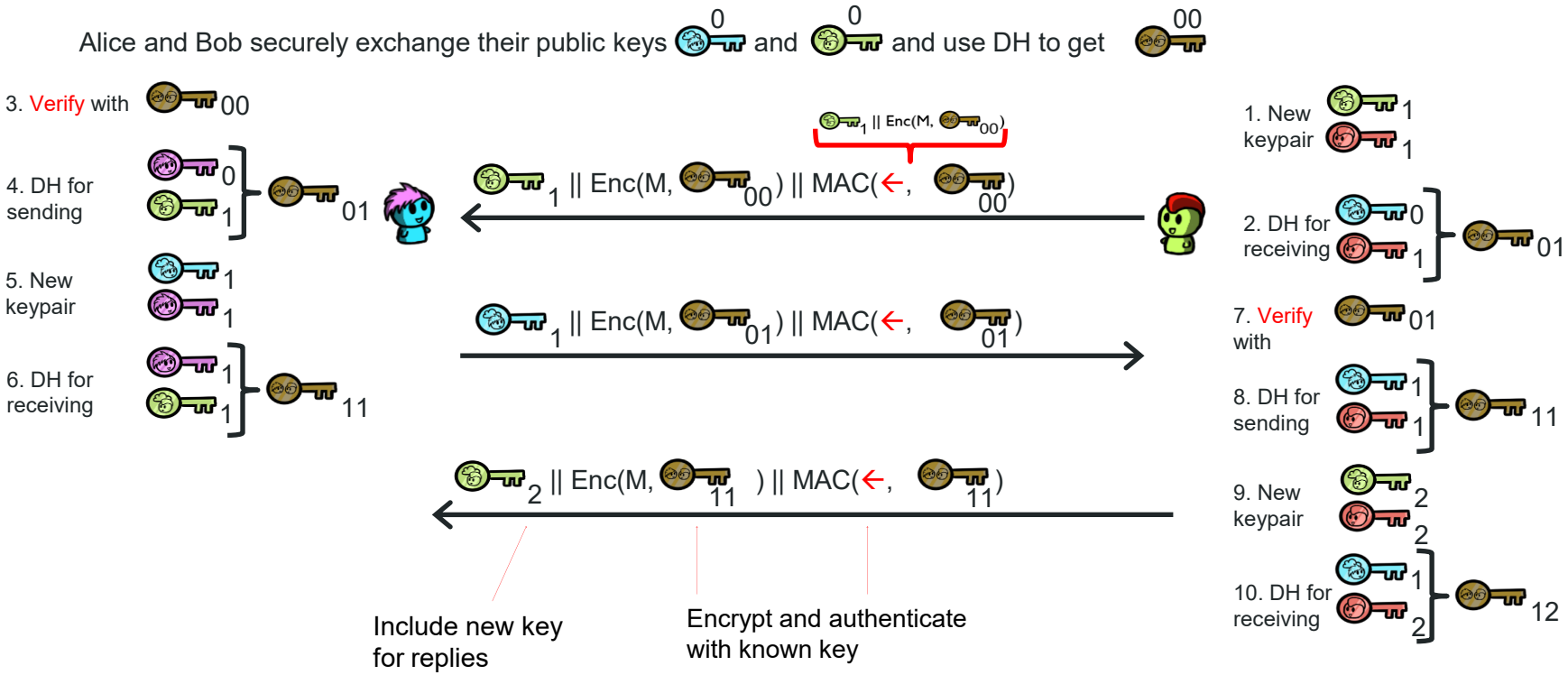
What OTR actually does (from the OTR paper)



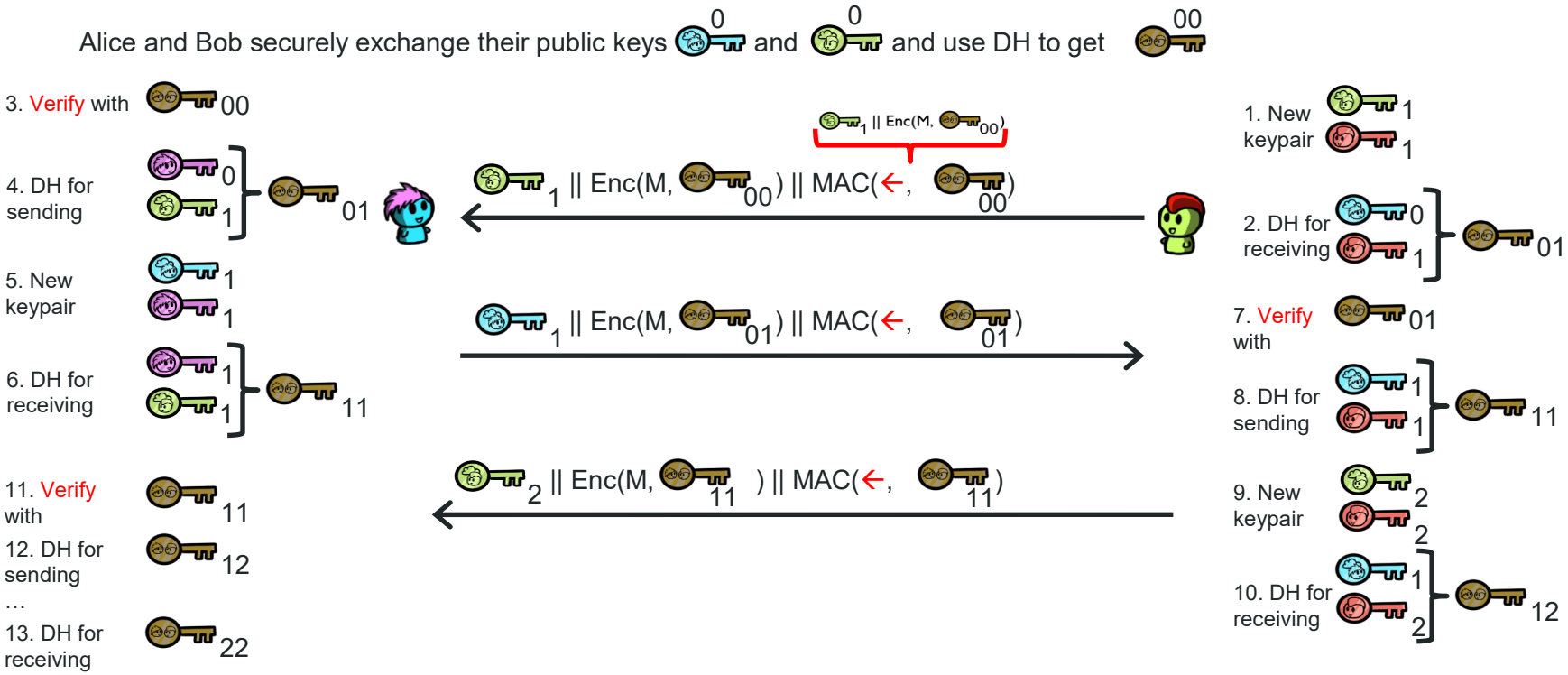
What OTR actually does (from the OTR paper)



What OTR actually does (from the OTR paper)



What OTR actually does (from the OTR paper)



OTR Summary

- OTR provides:
  - forward secrecy
  - post-compromise security
  - repudiation (deniable authentication)

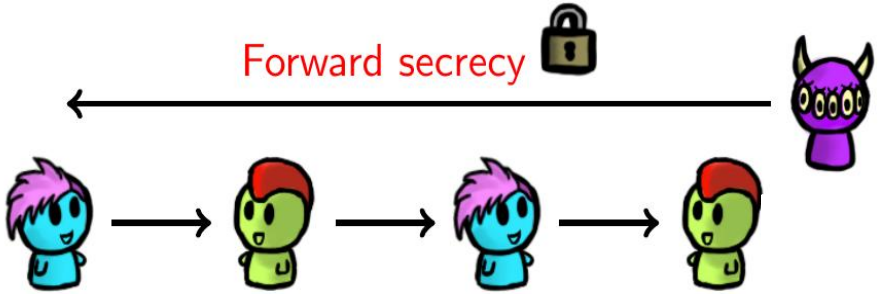
- But the ratchet only advances if the sender changes

Security Goals in Practice

Practical Limitations of OTR’s Goals

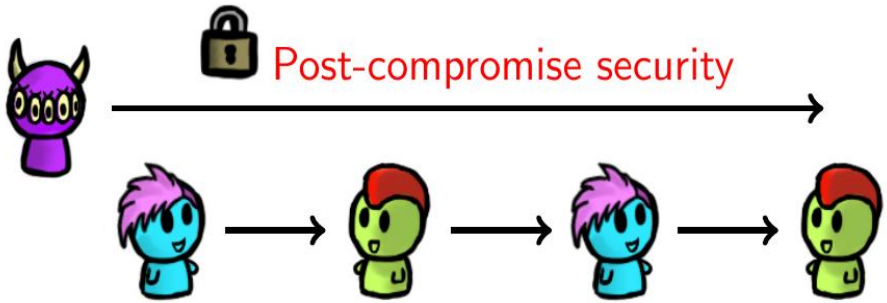
Security Goals in Practice – Forward Secrecy

- The main way Eve might learn Bob's key is by compromising Bob's device
- If she does this, can she just read the plaintext messages on Bob's device?
- What if Bob uses **disappearing messages**?
  - Now forward secrecy is useful



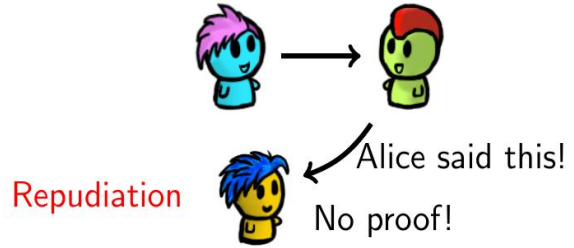
Security Goals in Practice – Post-Compromise Security

- If Eve was malicious, she could perform active attacks:
  - Impersonate Bob/perform an ongoing MITM attack
  - Add an extra device to the chat
- But if Eve is a passive adversary, Alice and Bob regain confidentiality



Security Goals in Practice – Repudiation

- Cryptographic repudiability is lack of cryptographic proof
- Most communication is already repudiable
  - Because it's not cryptographically authenticated at all!
  - And yet, people accept it as evidence
  - Deniability is also a social problem, not just a technical one
- Does this mean we should start adding cryptographic proof to messages?
  - No! Why provide *more* evidence than default?



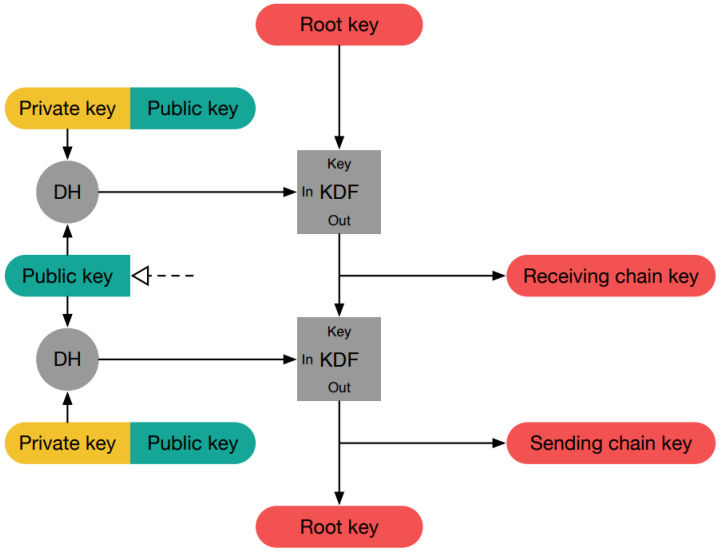
Signal

Signal

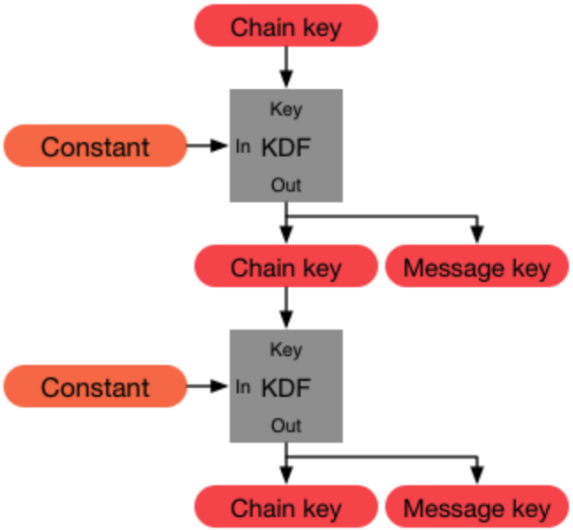
- Mobile app with companion desktop (Electron) client
  - OTR was less mobile-friendly
- Encryption protocol based on OTR
  - Double Ratchet Algorithm builds on OTR DH ratchet
  - Deniability ideas from OTR
- Protocol also used in other apps like WhatsApp, OMEMO extension for XMPP, etc.

The Double Ratchet

DH ratchet
(asymmetric, like in OTR)



KDF chain ratchet
(symmetric)

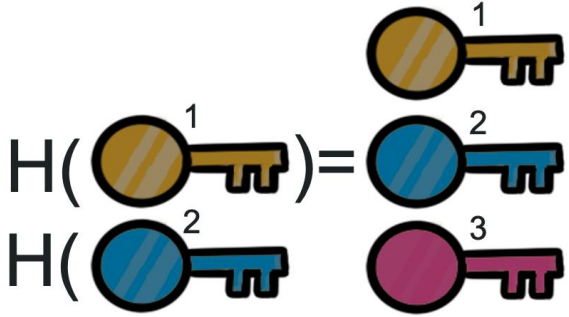


Rationale for the KDF Chain Ratchet

- What if instead of session keys, we had a new key for *each message*?

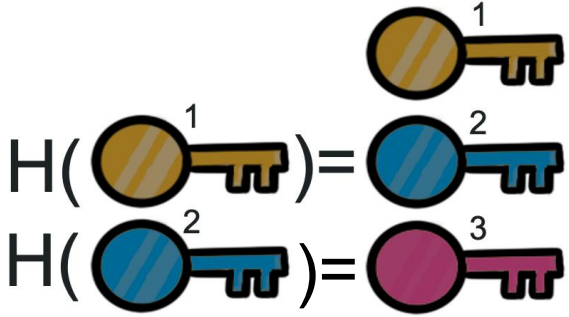
- We can do this deterministically
  - Simplified ratchet: $K_{n+1} = H(K_n)$

- **Q:** What happens if Eve compromises a key?



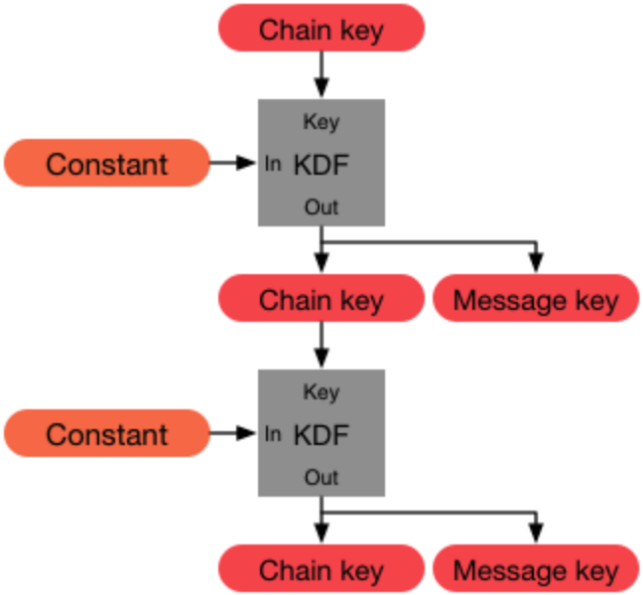
Rationale for the KDF Chain Ratchet

- What if instead of session keys, we had a new key for *each message*?
- We can do this deterministically
  - Simplified ratchet: $K_{n+1} = H(K_n)$
- **Q:** What happens if Eve compromises a key?
  - Eve can derive future keys!
  - (but not past keys)



KDF Ratchet

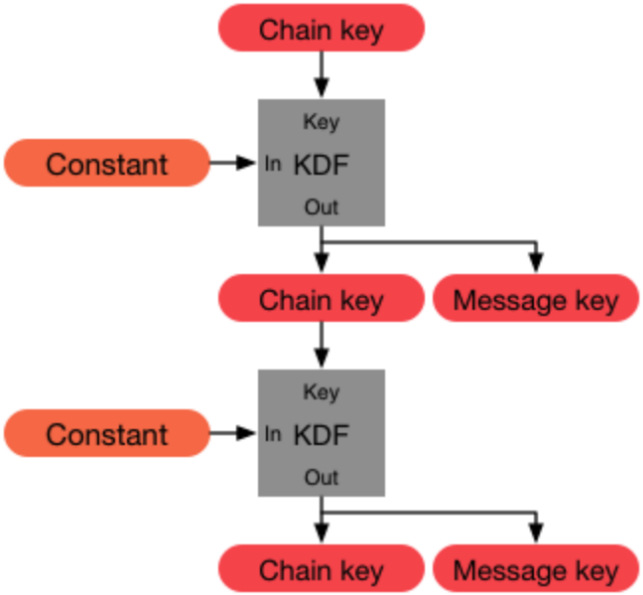
- KDF = Key Derivation Function
  - (think hashing – it only goes one way)
- Outputs **message key**
  - **Used to encrypt a single message**
- Outputs **chain key**
  - **Used to derive future keys**
- Why separate chain & message keys?
  - What if messages are out-of-order?



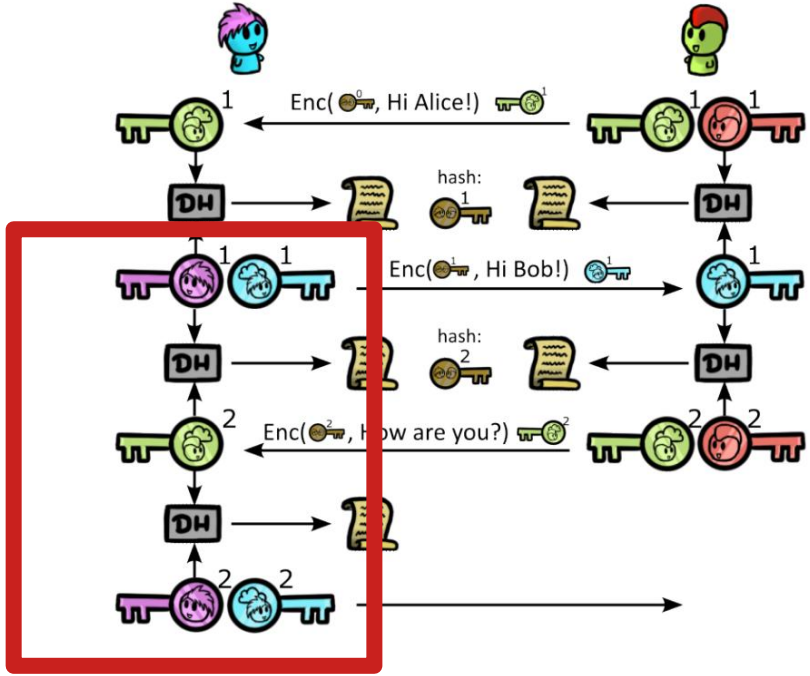
KDF Ratchet

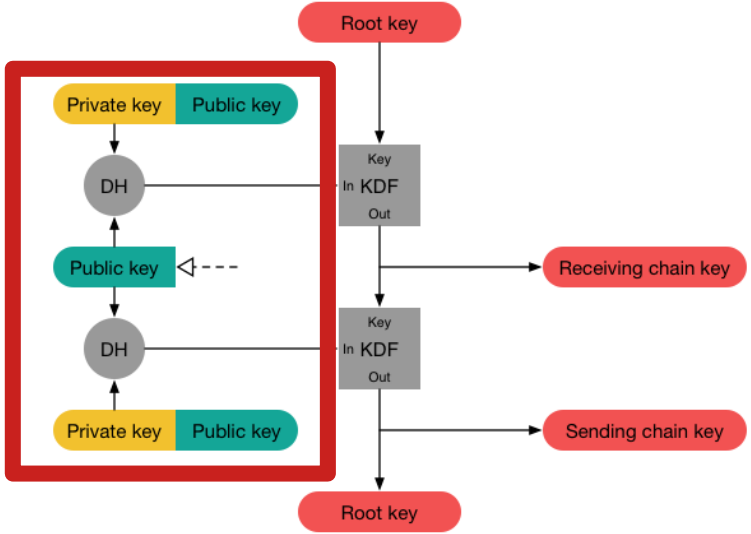
Takeaway: Message keys cannot be used to generate other chain or message keys, so it's safe to store old message keys for not-yet-delivered messages.

- KDF = Key Derivation Function
  - (think hashing – it only goes one way)
- Outputs **message key**
  - **Used to encrypt a single message**
- Outputs **chain key**
  - **Used to derive future keys**
- Why separate chain & message keys?
  - What if messages are out-of-order?



DH Ratchet

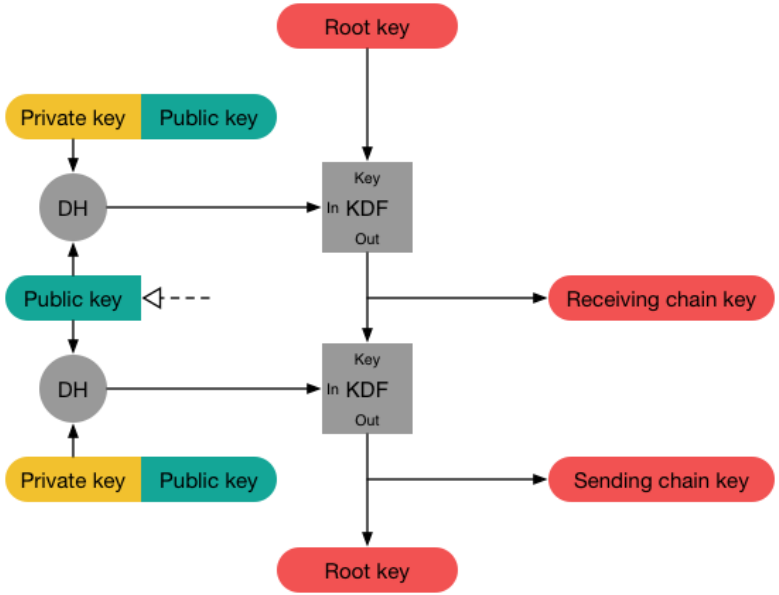

Enc(, Hi Alice!)
hash:
DH
Enc(, Hi Bob!)
hash:
Enc(, How are you?)


Root key
Private key
Public key
DH
Key In KDF Out
Public key
Receiving chain key
DH
Key In KDF Out
Private key
Public key
Sending chain key
Root key

DH Ratchet

- Like the incomplete version of OTR we saw earlier. But now also:

- Output is used for generating receiving chain and sending chain keys
  - These are used as input for the KDF ratchet

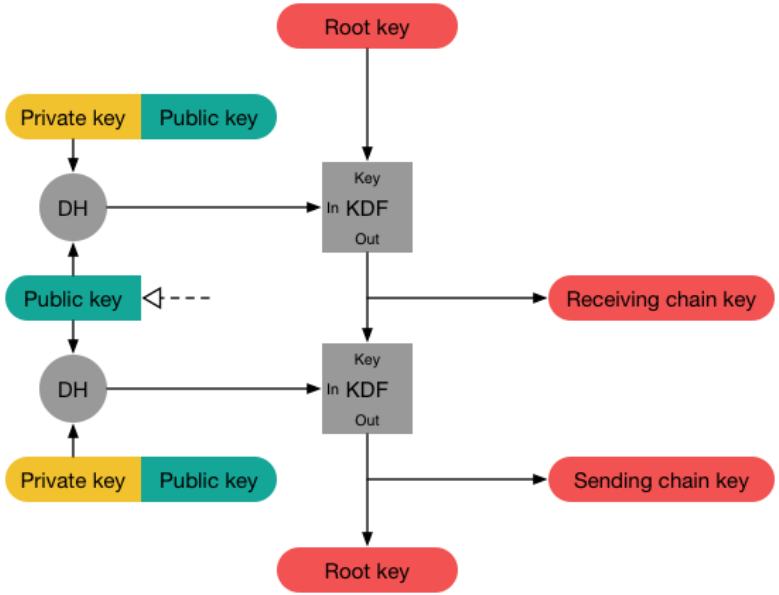


DH Ratchet

Note: There are two KDFs, and the DH ratchet's receiving and sending chain keys are inputs to the KDF ratchet.

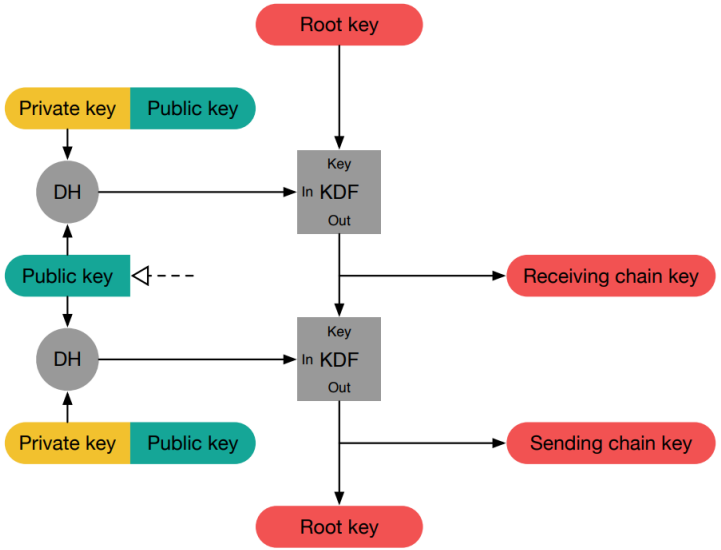
- Like the incomplete version of OTR we saw earlier. But now also:

- Output is used for generating receiving chain and sending chain keys
  - These are used as input for the KDF ratchet

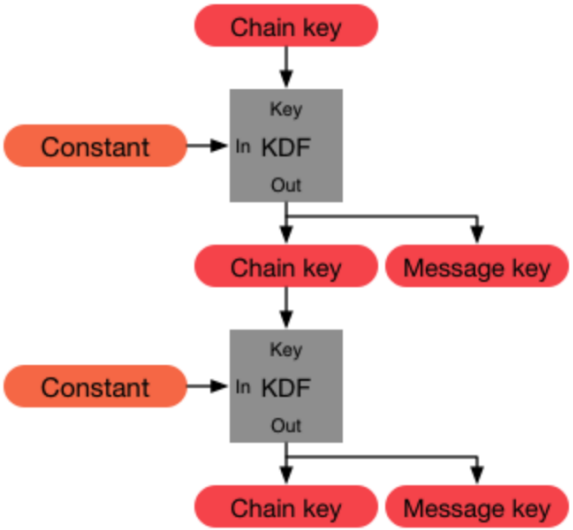


The Double Ratchet

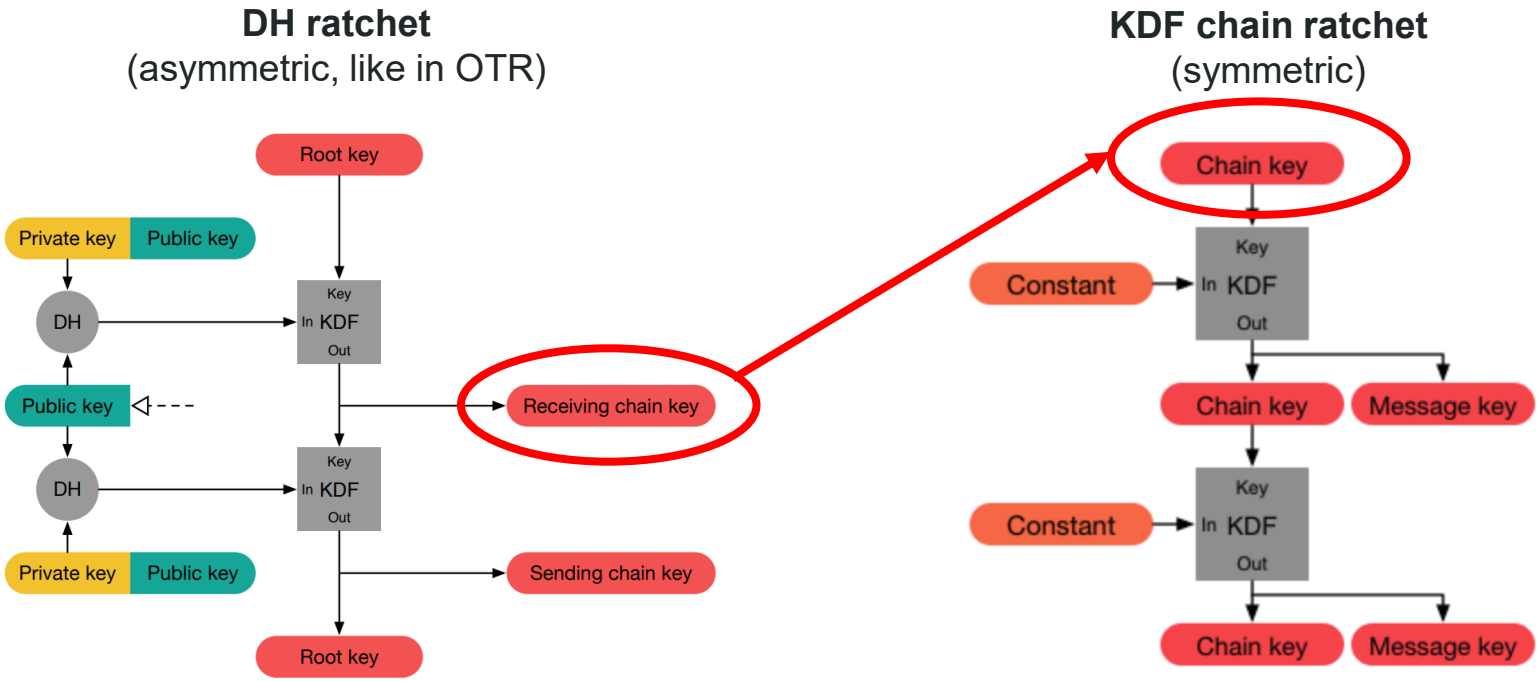
DH ratchet
(asymmetric, like in OTR)



KDF chain ratchet
(symmetric)

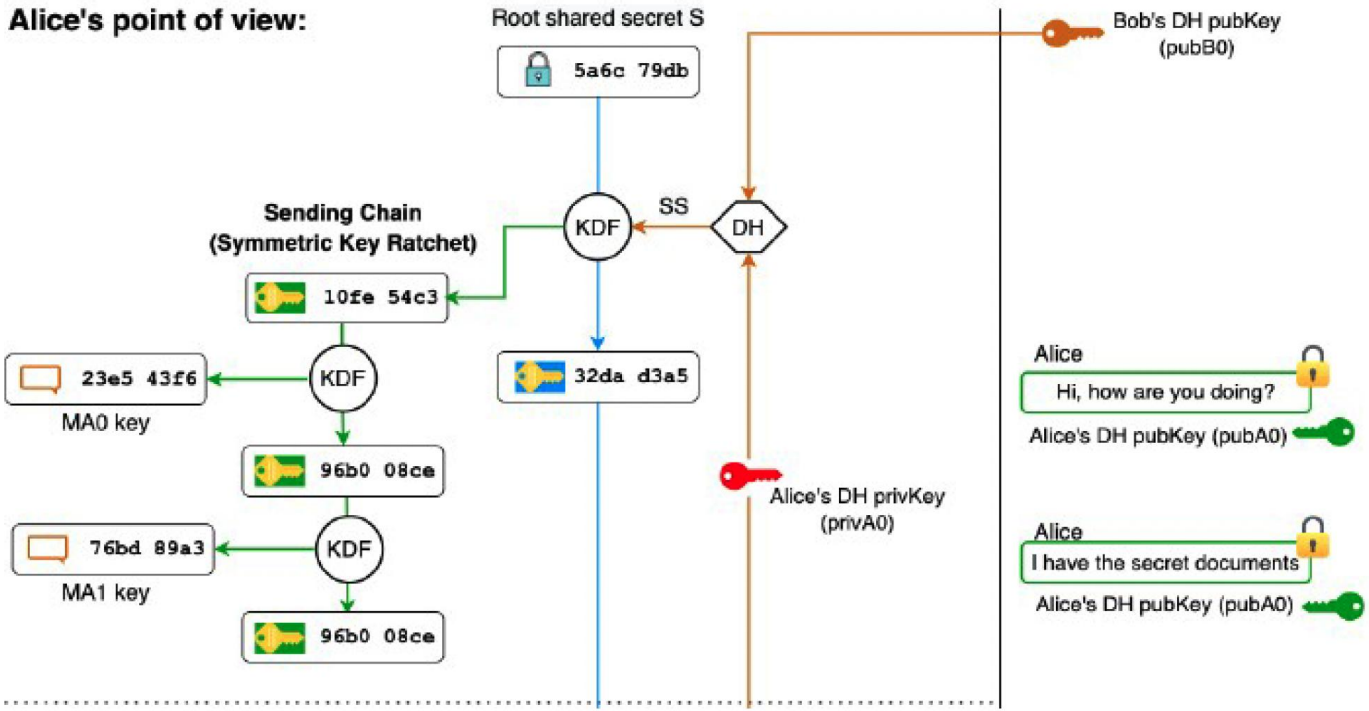


The Double Ratchet



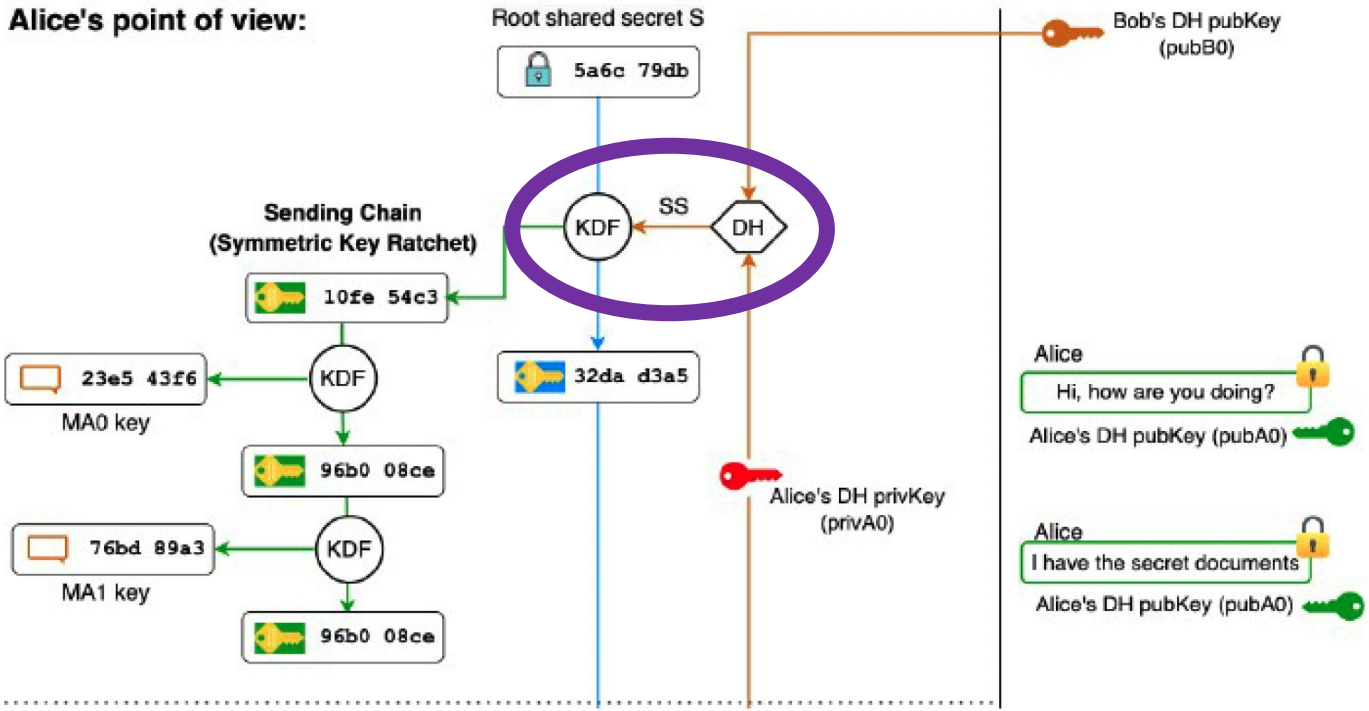
Double Ratchet Algorithm

- Alice -> Bob
- Alice and Bob do **DH** and get Alice's sending chain/Bob's receiving chain
- Alice **derives a key** with her sending chain
- Alice uses this **MA0** key to encrypt her message to Bob



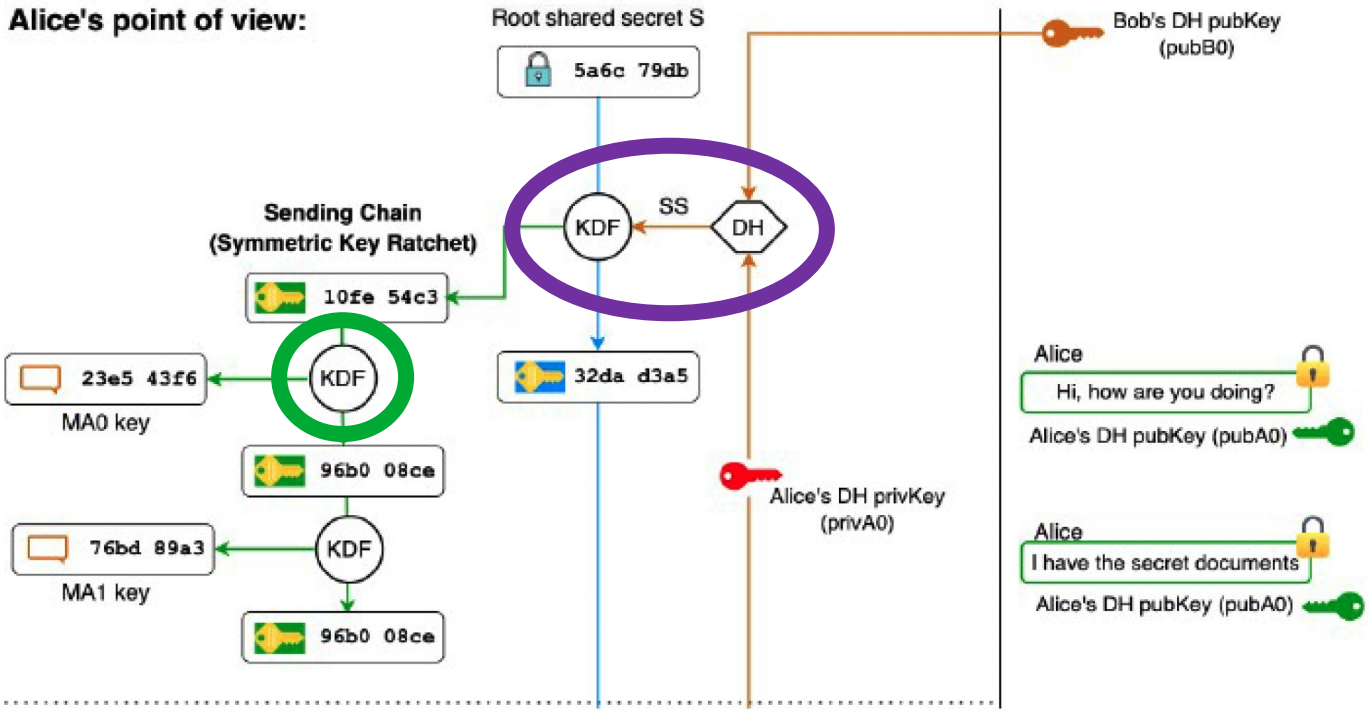
Double Ratchet Algorithm

- Alice -> Bob
- Alice and Bob do **DH** and get Alice's sending chain/Bob's receiving chain
- Alice **derives a key** with her sending chain
- Alice uses this **MA0** key to encrypt her message to Bob



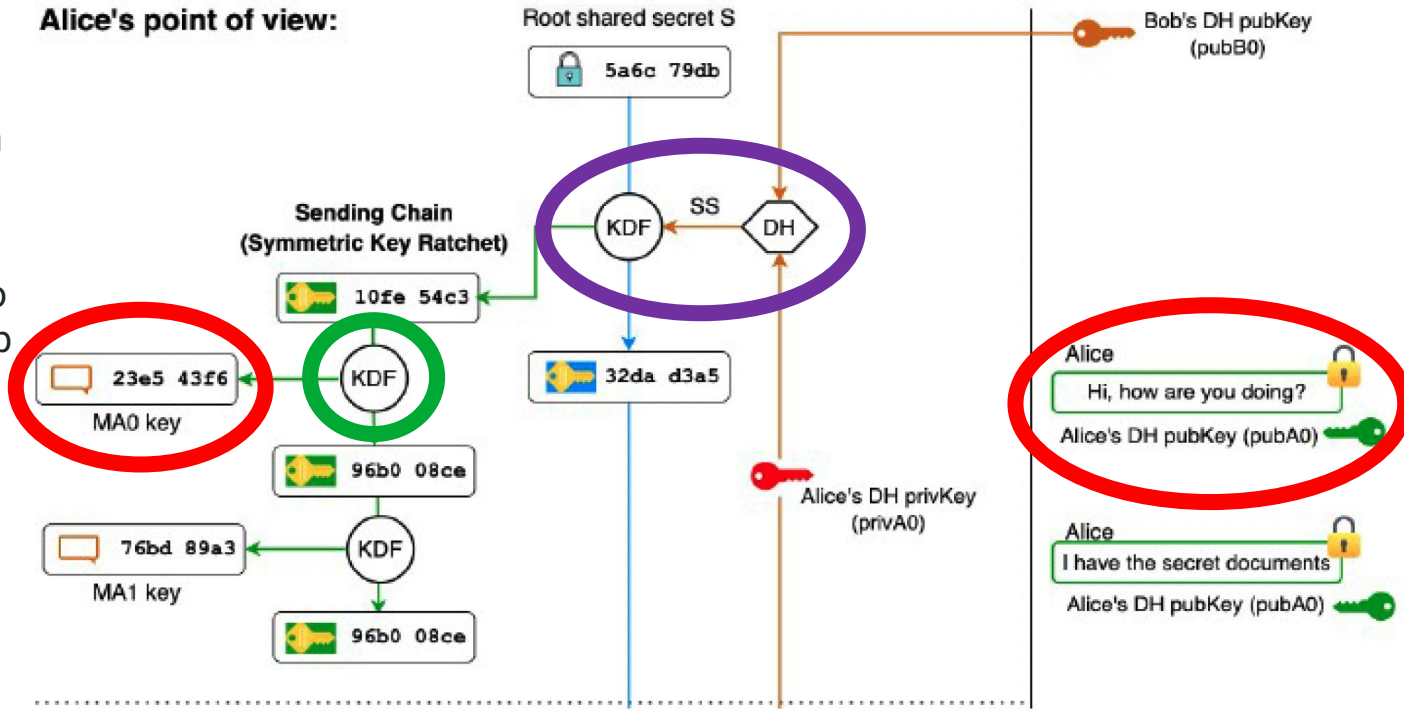
Double Ratchet Algorithm

- Alice -> Bob
- Alice and Bob do **DH** and get Alice's sending chain/Bob's receiving chain
- Alice **derives a key** with her sending chain
- Alice uses this **MA0** key to encrypt her message to Bob



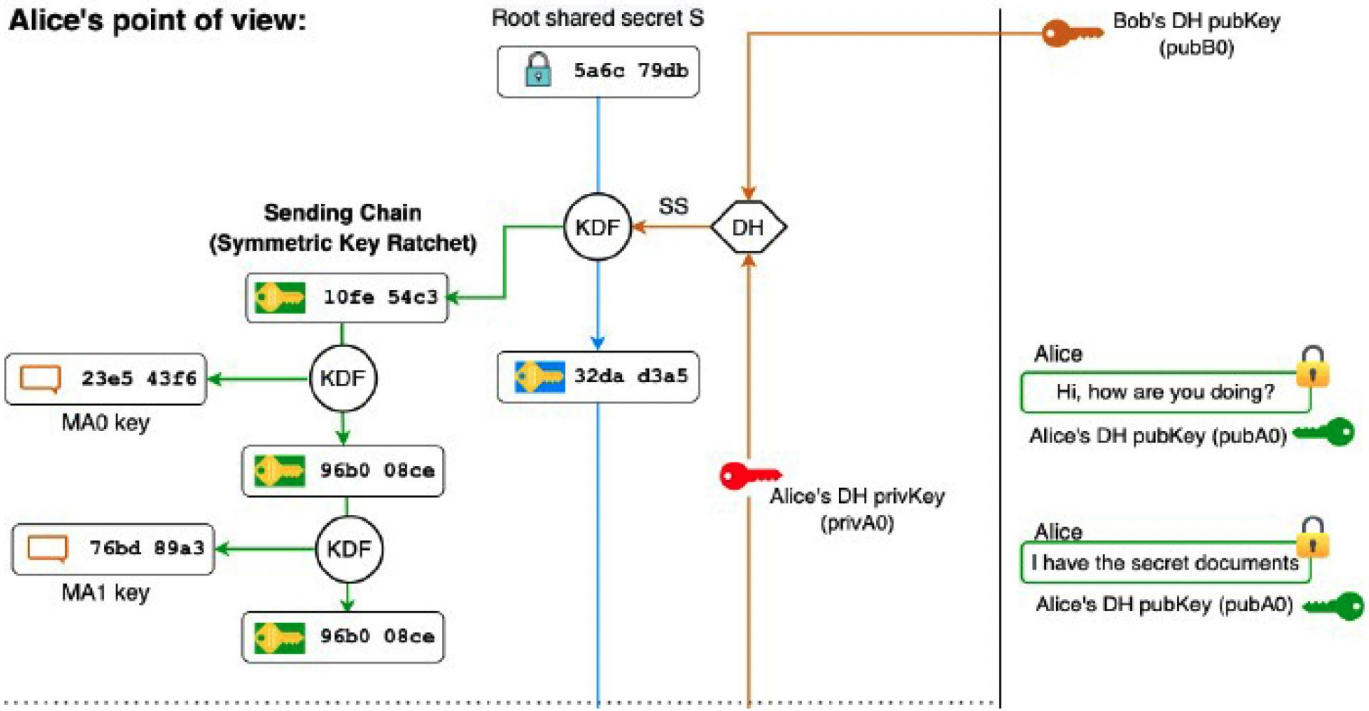
Double Ratchet Algorithm

- Alice -> Bob
- Alice and Bob do **DH** and get Alice's sending chain/Bob's receiving chain
- Alice **derives a key** with her sending chain
- Alice uses this **MA0** key to encrypt her message to Bob



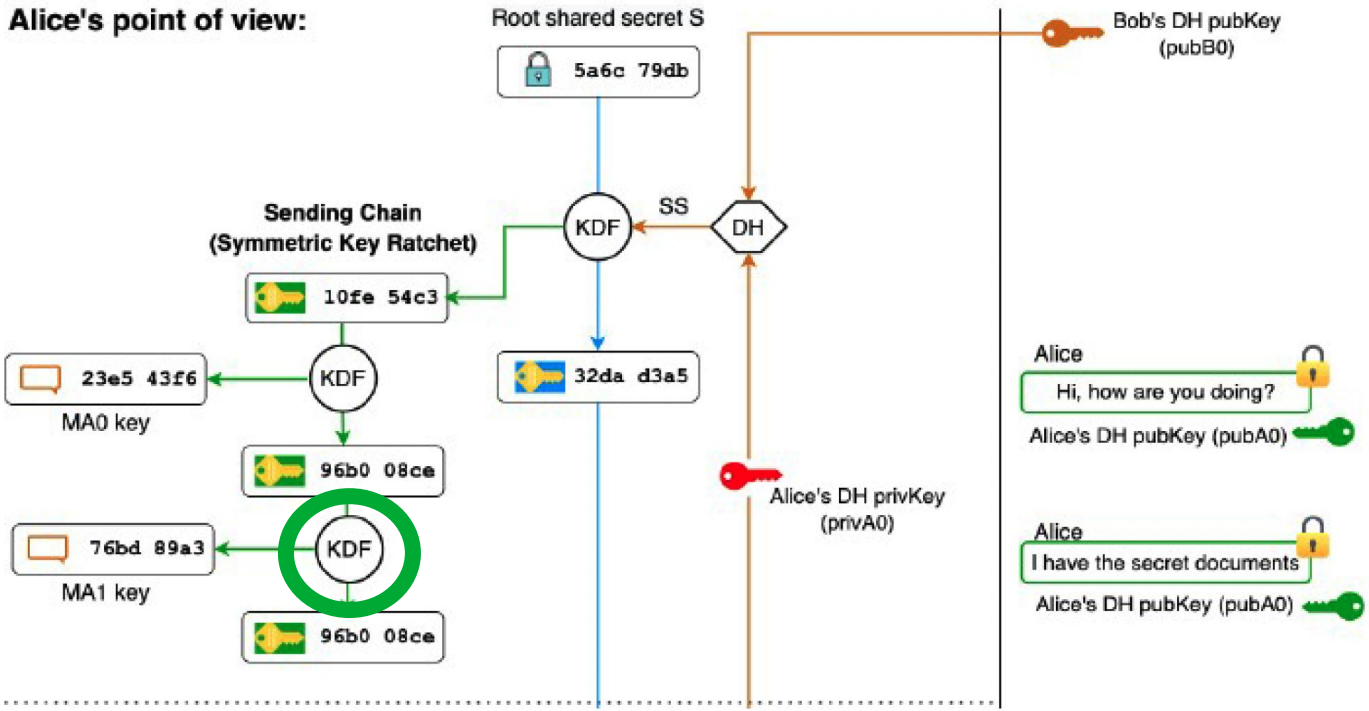
Double Ratchet Algorithm

- Alice -> Bob **(again)**
- **No new DH until Bob replies**
- Alice **derives another key** with her sending chain
- Alice uses **MA1** key to encrypt her message to Bob



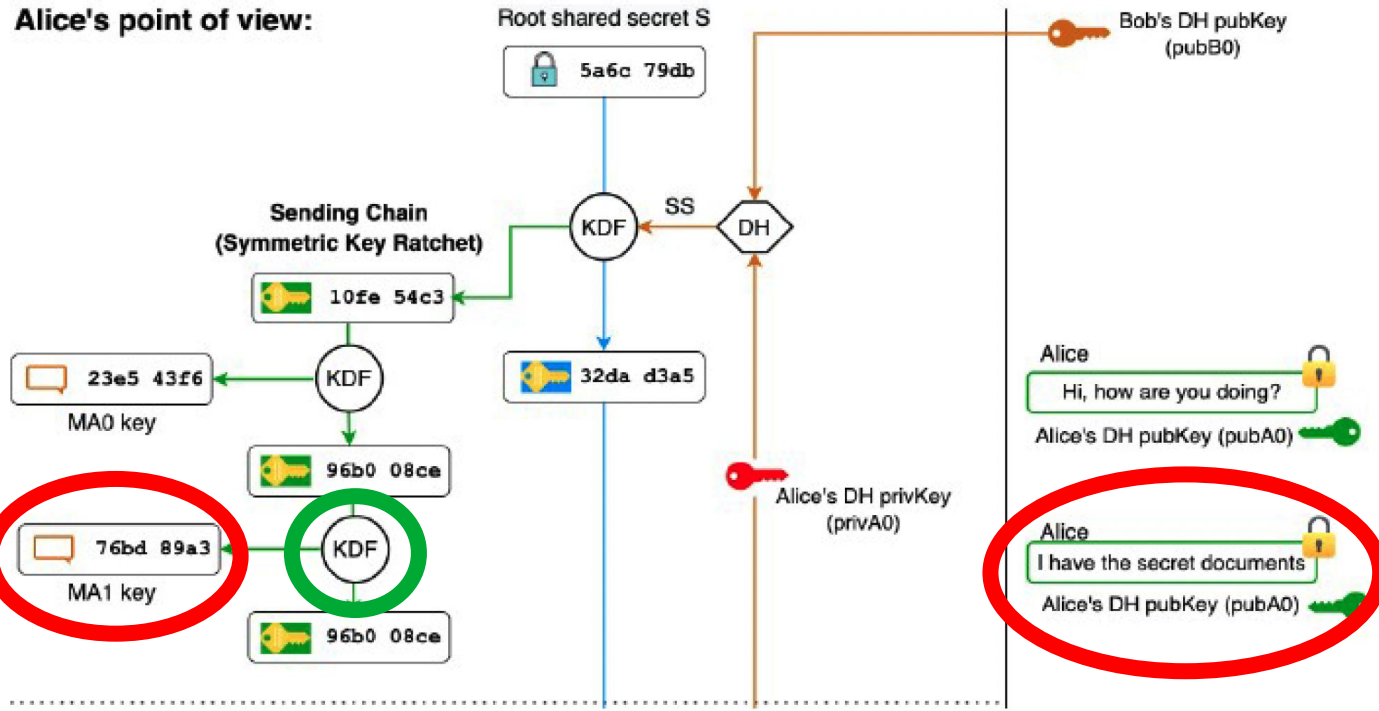
Double Ratchet Algorithm

- Alice -> Bob **(again)**
- **No new DH until Bob replies**
- Alice **derives another key** with her sending chain
- Alice uses **MA1** key to encrypt her message to Bob



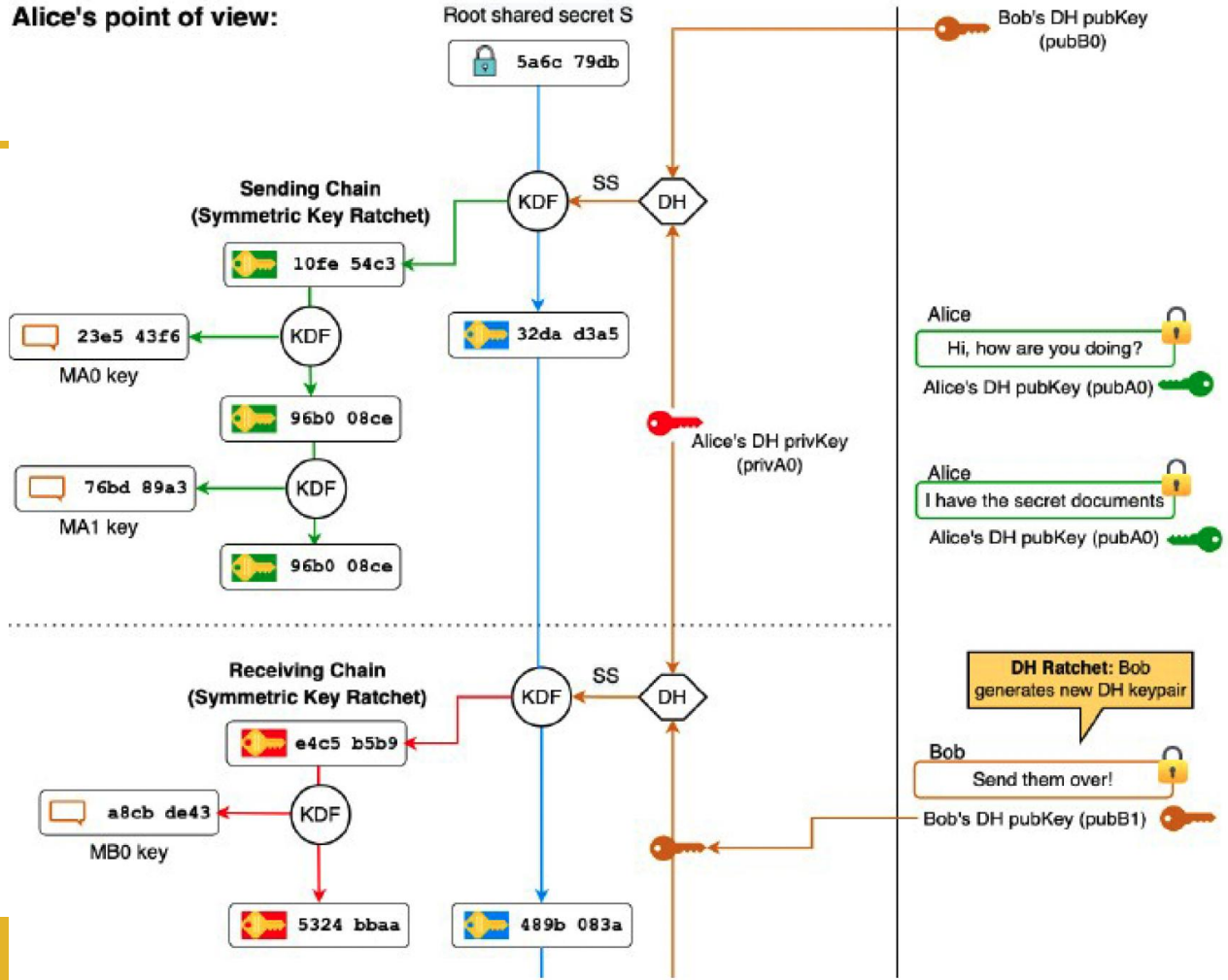
Double Ratchet Algorithm

- Alice -> Bob **(again)**
- **No new DH until Bob replies**
- Alice **derives another key** with her sending chain
- Alice uses **MA1** key to encrypt her message to Bob



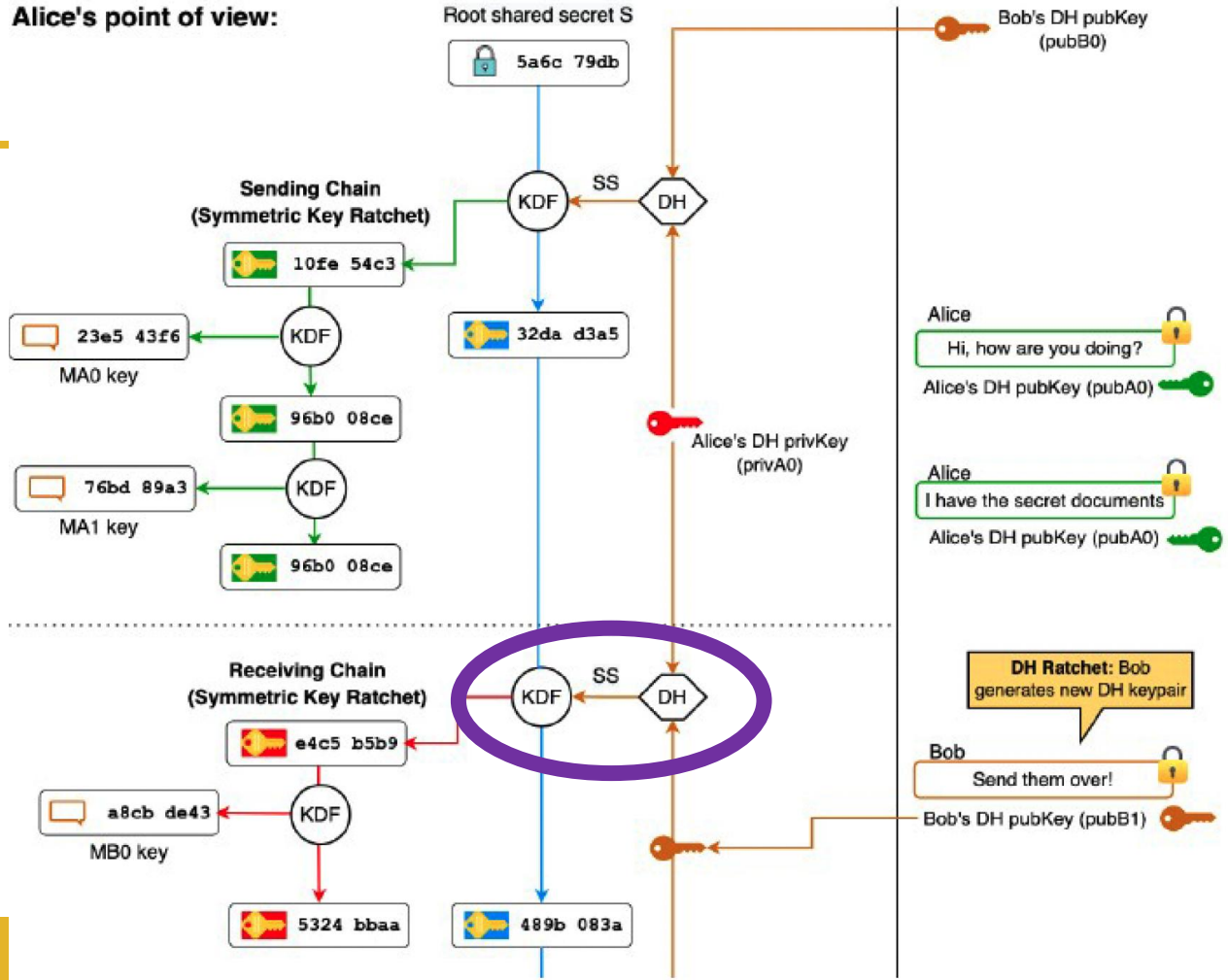
Double Ratchet

- Bob -> Alice
- Alice and Bob do DH and get Alice's receiving chain/Bob's sending chain
- Alice **derives a key** with her receiving chain
- Alice uses **MB0** key to decrypt a message from Bob



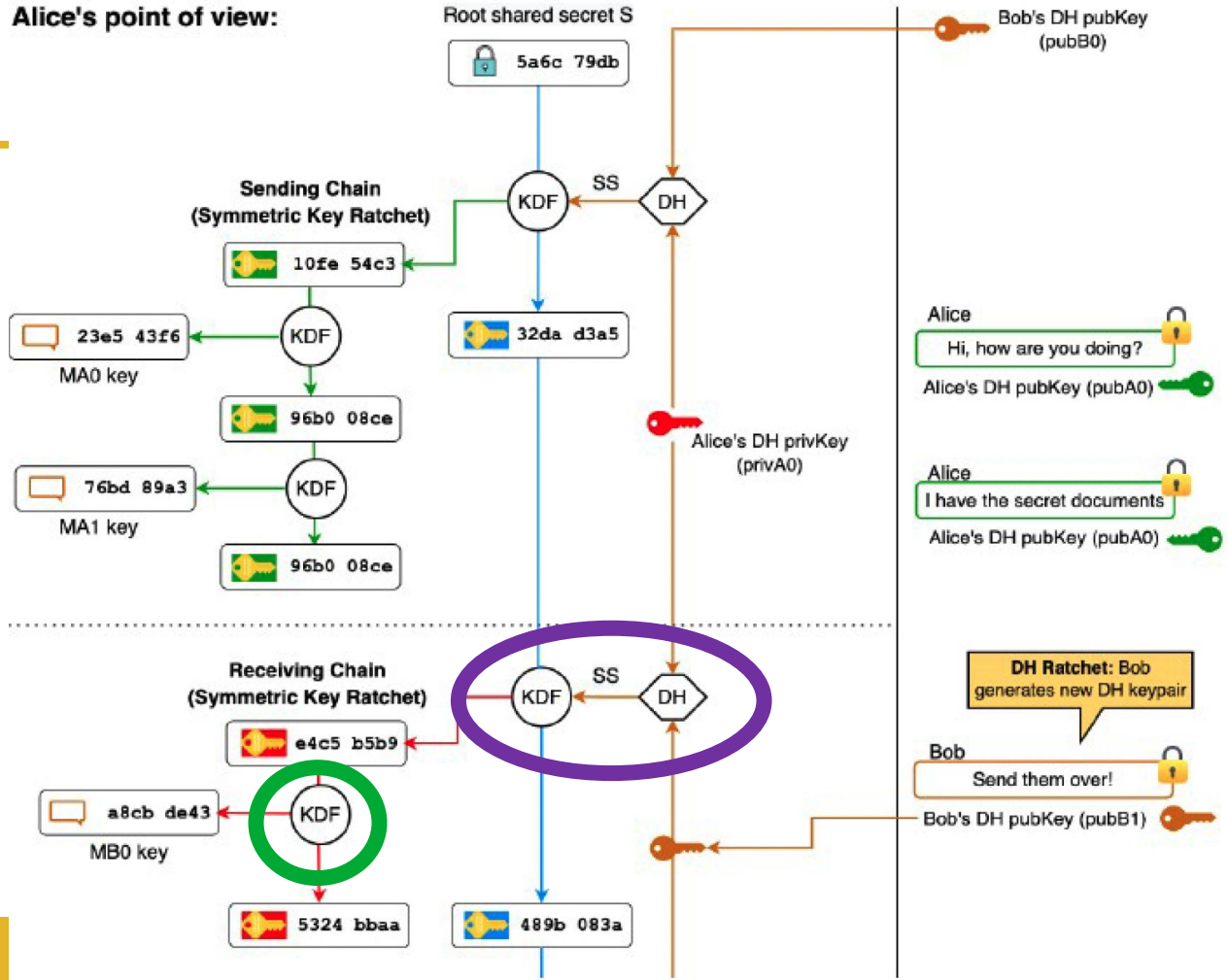
Double Ratchet

- Bob -> Alice
- Alice and Bob do DH and get Alice's receiving chain/Bob's sending chain
- Alice **derives a key** with her receiving chain
- Alice uses **MB0** key to decrypt a message from Bob



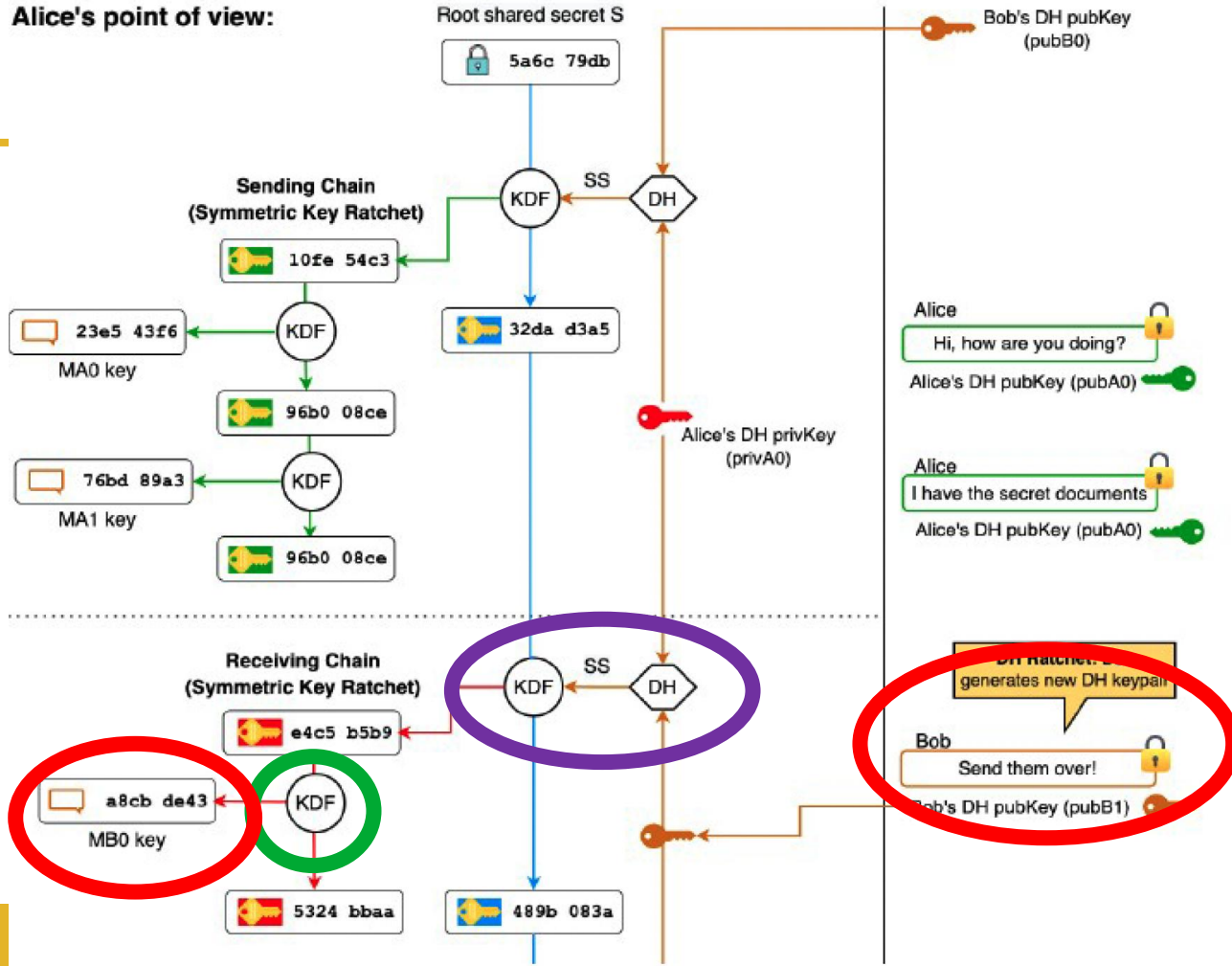
Double Ratchet

- Bob -> Alice
- Alice and Bob do DH and get Alice's receiving chain/Bob's sending chain
- Alice **derives a key** with her receiving chain
- Alice uses **MB0** key to decrypt a message from Bob



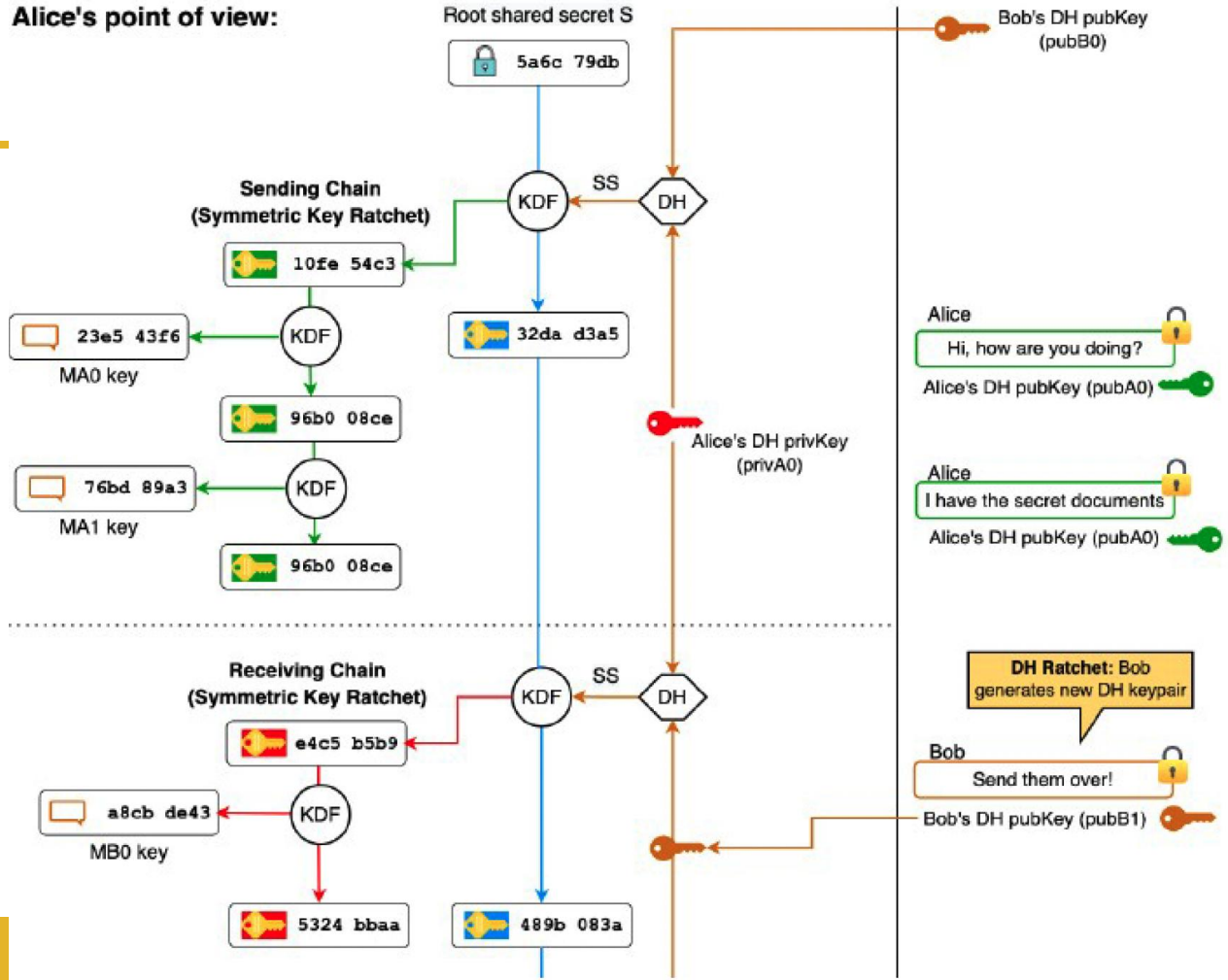
Double Ratchet

- Bob -> Alice
- Alice and Bob do DH and get Alice's receiving chain/Bob's sending chain
- Alice **derives a key** with her receiving chain
- Alice uses **MB0** key to decrypt a message from Bob



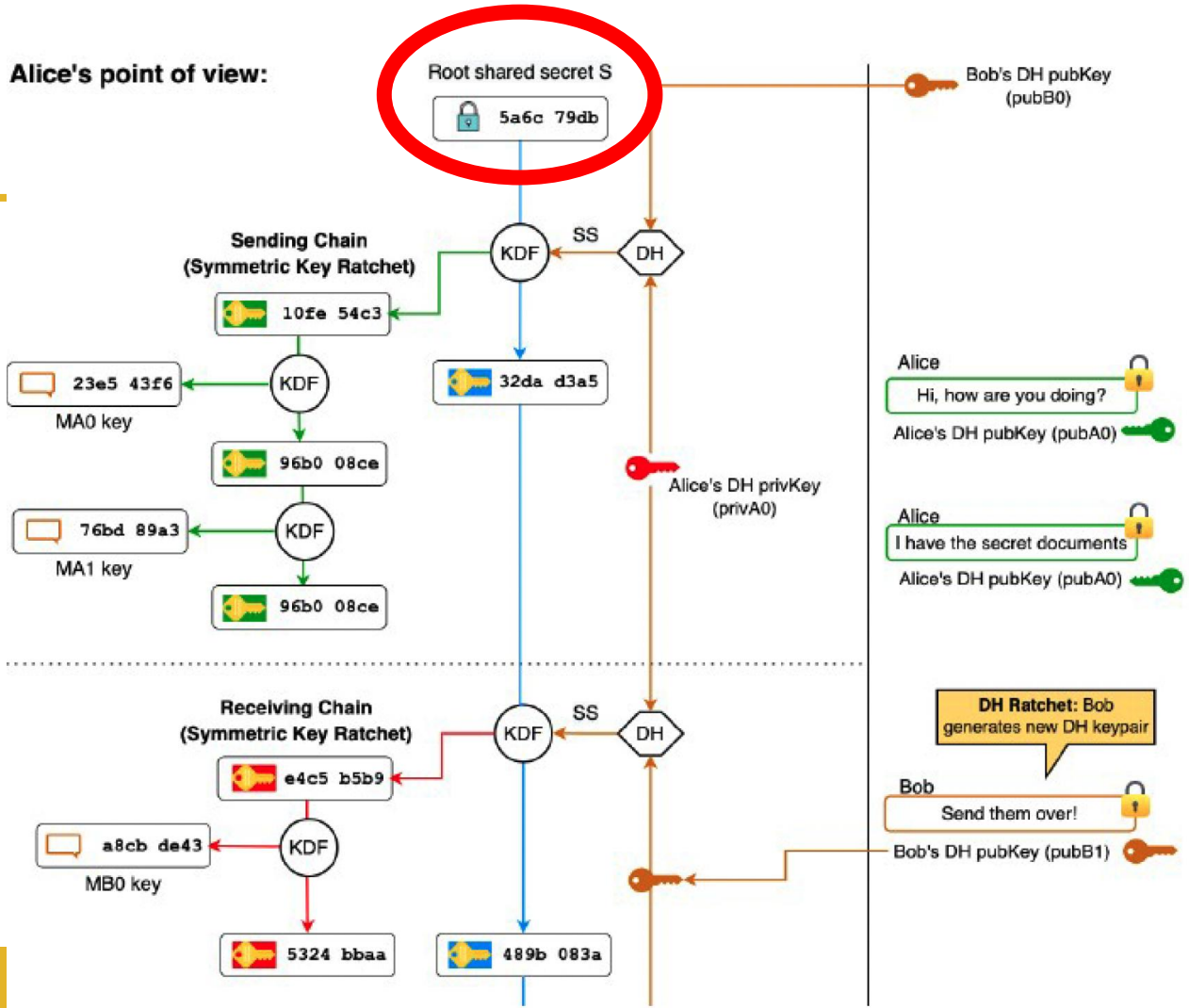
Double Ratchet

- Why is it safe to use new DH keys immediately?



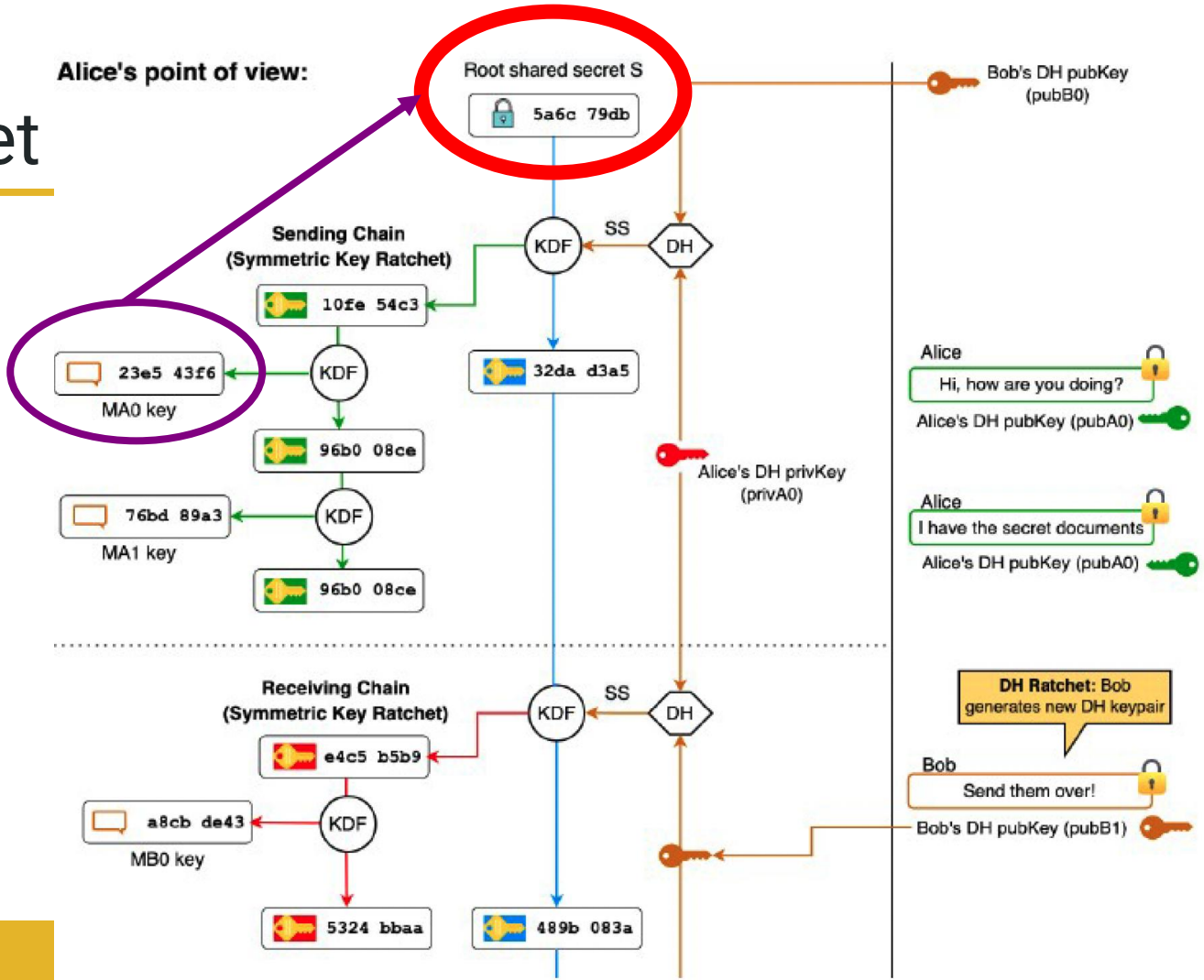
Double Ratchet

- Why is it safe to use new DH keys immediately?
  - Can authenticate with root shared secret



Double Ratchet

- Why is it safe to use new DH keys immediately?
  - Can authenticate with root shared secret
  - Message key can only be computed by someone who knows the root shared secret
  - If the root shared secret remains secure, MITM attempts will be detected



Signal Summary

- Provides forward secrecy
  - For every message due to KDF chain ratchet!
  - (Recall OTR only ratcheted when the sender changed.)
- Provides post-compromise security
  - A leak of past or long-term keys will be healed by introducing new DH ratchet keys.
- Supports out-of-order message delivery
  - Users can store per-message keys until late messages arrive.

Signal Summary

- Provides forward secrecy
  - For every message due to KDF chain ratchet!
  - (Recall OTR only ratcheted when the sender changed.)
- Provides post-compromise security
  - A leak of past or long-term keys will be healed by introducing new DH ratchet keys.
- Supports out-of-order message delivery
  - Users can store per-message keys until late messages arrive.



Deniability in Signal

- Uses MACs like OTR
  - Alice can deny that a message was written by her

Deniability in Signal

- Uses MACs like OTR
  - Alice can deny that a message was written by her
  - Can we do even better?

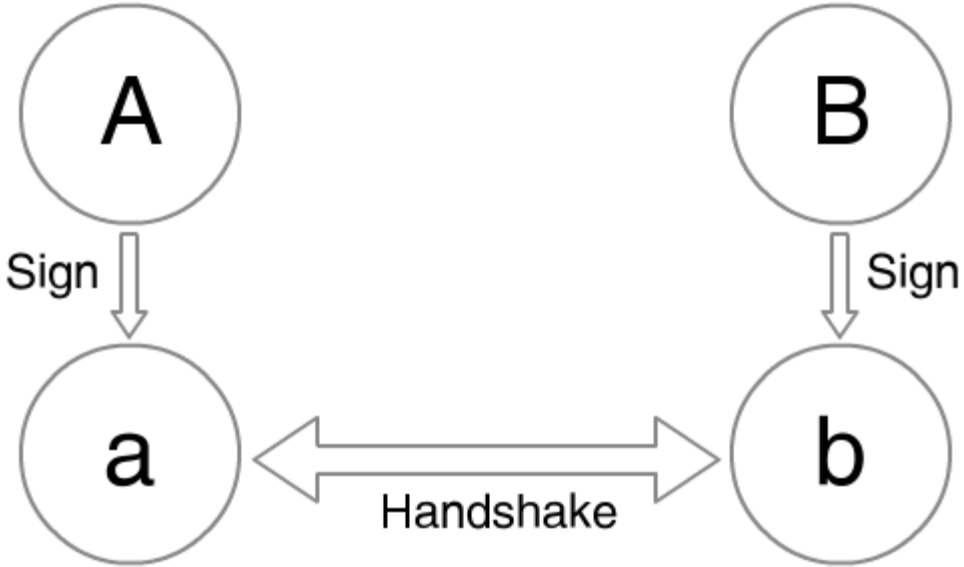
Deniability in Signal

- Uses MACs like OTR
  - Alice can deny that a message was written by her
  - Can we do even better?
  - Can Alice deny that she even talked to Bob?

K = DH(a,b)

OTR's Initial Handshake

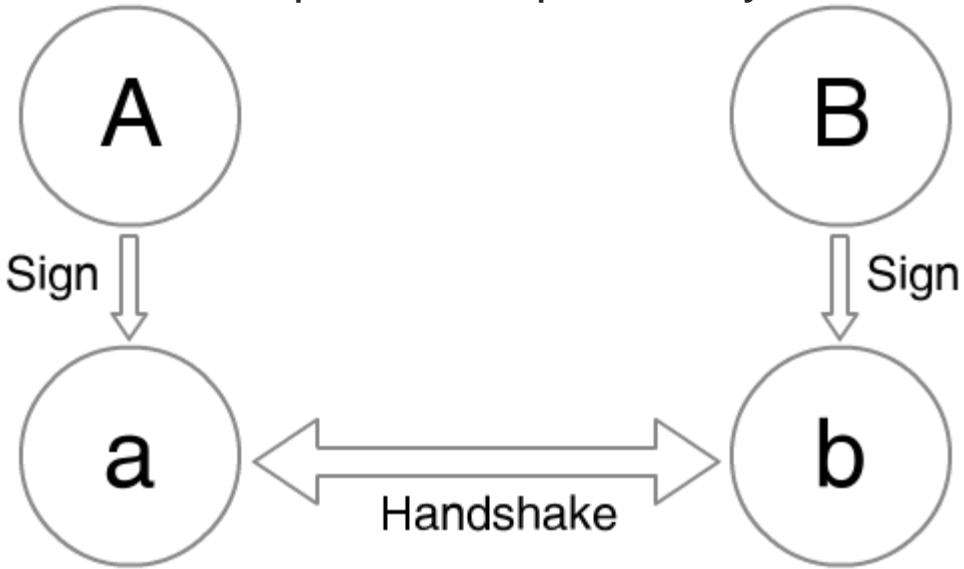
- A, B are identity keys; a, b are ephemeral (short-lived) keys
- Signatures are bad for deniability



K = DH(a,b)

OTR's Initial Handshake

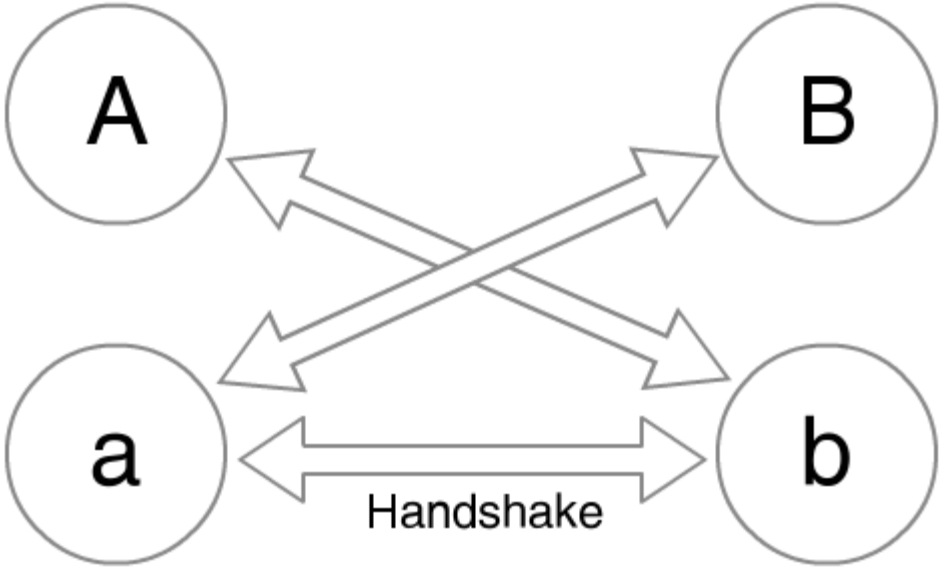
- A, B are identity keys; a, b are ephemeral (short-lived) keys
- Signatures are bad for deniability
- Creating this handshake requires a DH private key from Alice OR Bob



K = KDF(DH(a,B) || DH(A,b) || DH(a,b))

3DH: An Improved Design

- A, B are identity keys; a, b are ephemeral (short-lived) keys
- All arrows indicate DH; no signatures



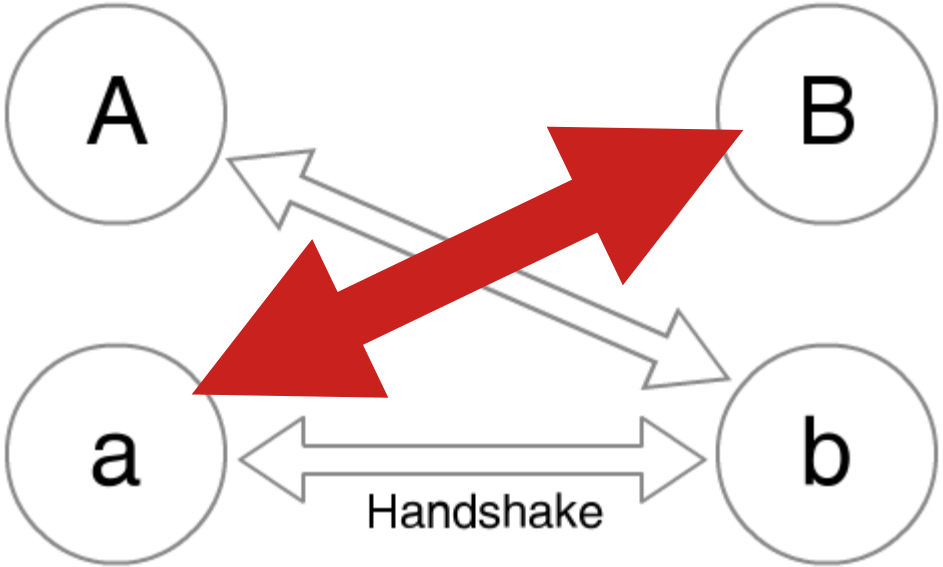
K = KDF(DH(a,B) || DH(A,b) || DH(a,b))

3DH: An Improved Design

- A, B are identity keys; a, b are ephemeral (short-lived) keys
- All arrows indicate DH; no signatures

Need the private key for **a** or **B** to compute this term.

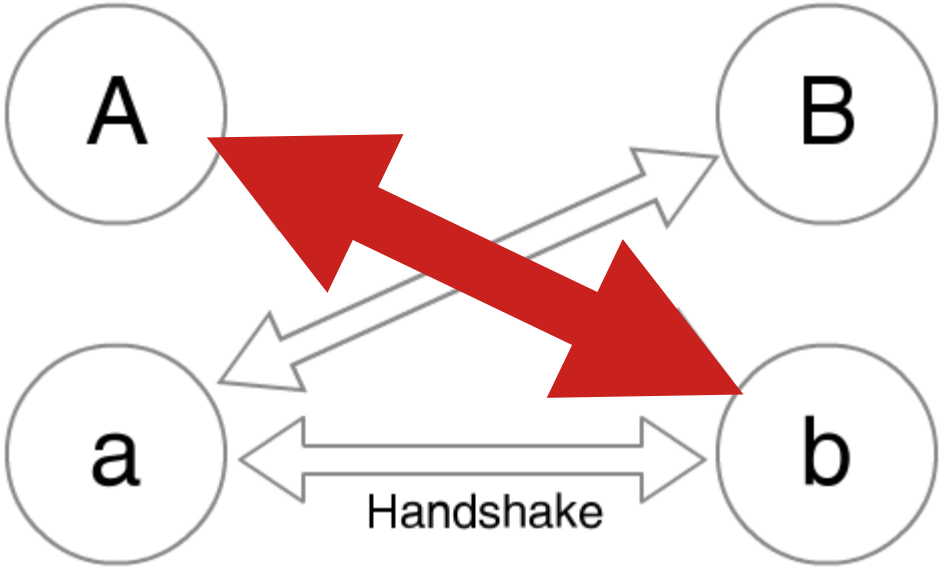
Alice's perspective: "I know **a** belongs to me and **B** belongs to Bob. The other party must be Bob."



K = KDF(DH(a,B) || DH(A,b) || DH(a,b))

3DH: An Improved Design

- A, B are identity keys; a, b are ephemeral (short-lived) keys
- All arrows indicate DH; no signatures



Need the private key for **A** or **b** to compute this term.

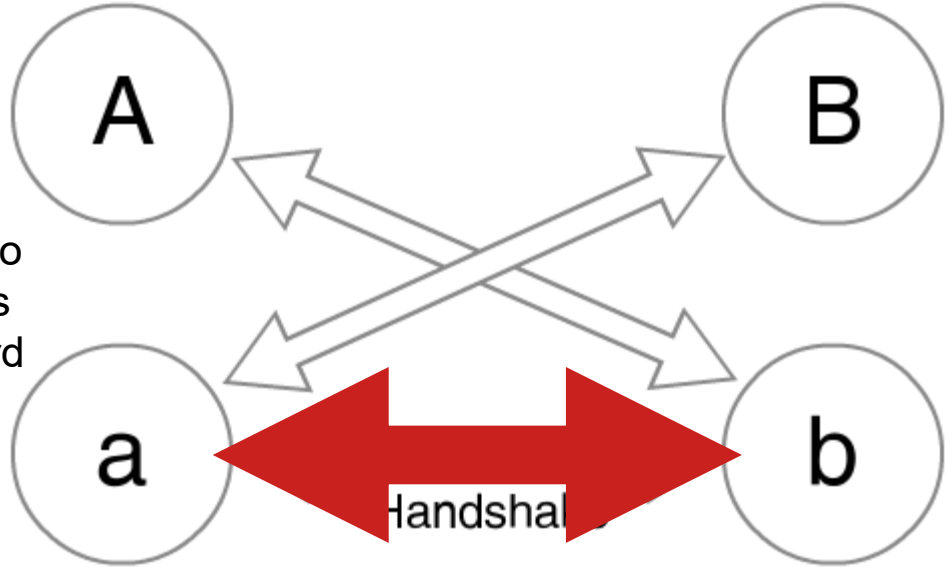
Bob's perspective: "I know **b** belongs to me and **A** belongs to Alice. The other party must be Alice."

K = KDF(DH(a,B) || DH(A,b) || DH(a,b))

3DH: An Improved Design

- A, B are identity keys; a, b are ephemeral (short-lived) keys
- All arrows indicate DH; no signatures

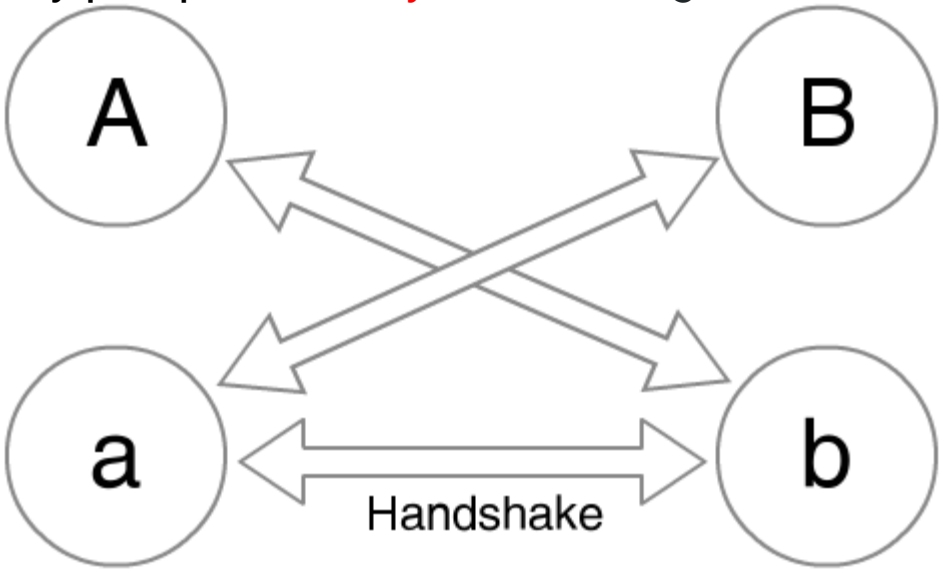
DH between two ephemeral keys provides forward secrecy.



K = KDF(DH(a,B) || DH(A,b) || DH(a,b))

3DH: An Improved Design

- A, B are identity keys; a, b are ephemeral (short-lived) keys
- All arrows indicate DH; no signatures
- From a third-party perspective, anyone can forge this handshake





If Mallory generates **a** and **b** herself, she can compute all the DH terms.

K = KDF(DH(a,B) || DH(A,b) || DH(a,b))

3DH: An Improved Design

- Signal uses a variant of 3DH that supports asynchronous communication

Signal Summary

- Provides forward secrecy
  - For every message due to KDF chain ratchet!
  - (Recall OTR only ratcheted when the sender changed.)
- Provides post-compromise security
  - A leak of past or long-term keys will be healed by introducing new DH ratchet keys.
- Supports out-of-order message delivery
  - Users can store per-message keys until late messages arrive.
- Provides deniability
  - Comparable to OTR

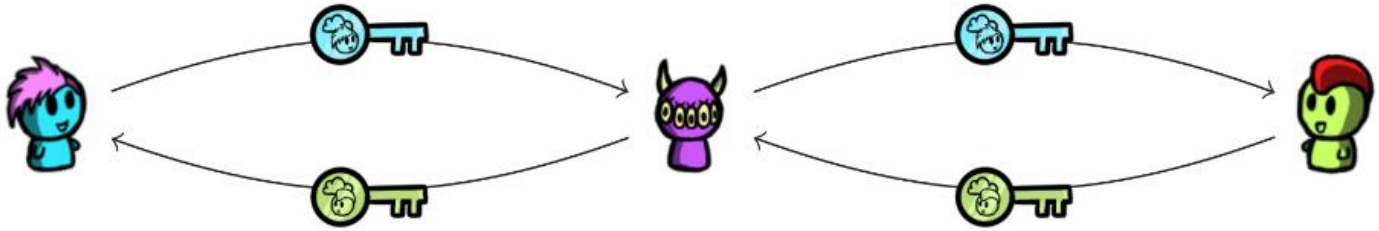
Key Verification

Crypto is (Relatively) Easy; Keys are Hard

- If Alice and Bob have each other's public keys, they can talk securely
- How do they get each other's keys?
  - From a keyserver?
  - How do they know they have the **correct** keys?

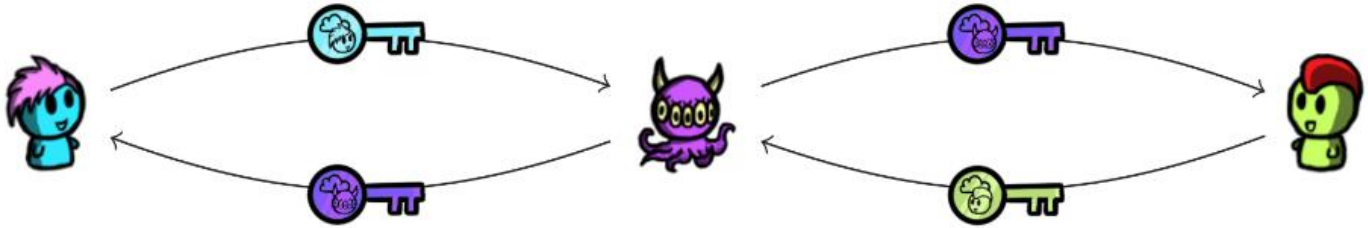
Crypto is (Relatively) Easy; Keys are Hard

- If Alice and Bob have each other's public keys, they can talk securely
- How do they get each other's keys?
  - From a keyserver?
  - How do they know they have the **correct** keys?



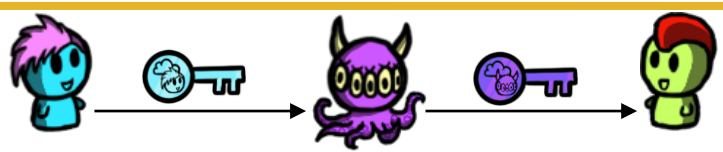
Crypto is (Relatively) Easy; Keys are Hard

- If Alice and Bob have each other's public keys, they can talk securely
- How do they get each other's keys?
  - From a keyserver?
  - How do they know they have the **correct** keys?



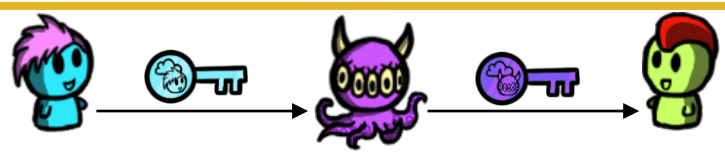
MITM Defenses

- Do nothing and hope for the best

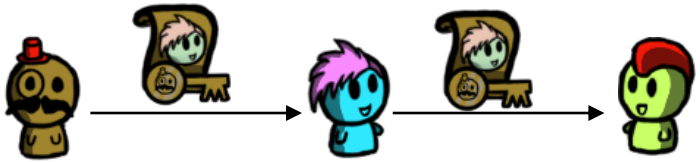


MITM Defenses

- Do nothing and hope for the best

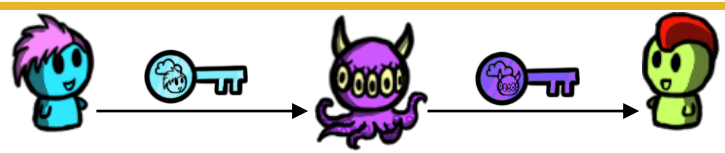


- Certificate Authorities

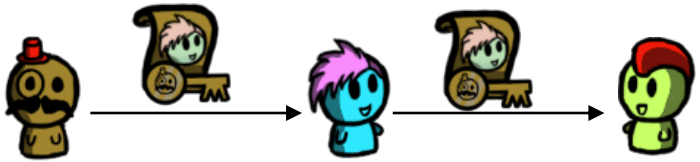


MITM Defenses

- Do nothing and hope for the best



- Certificate Authorities

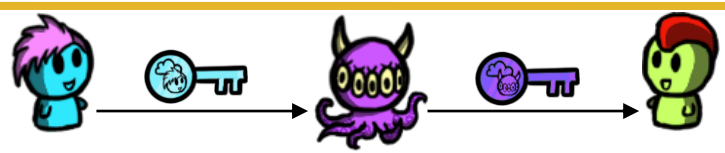


- Trust on First Use (TOFU)

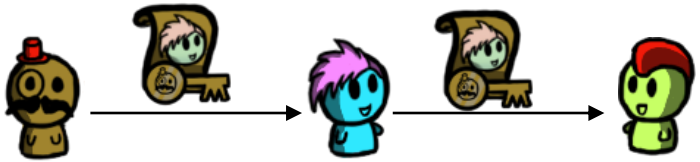


MITM Defenses

- Do nothing and hope for the best



- Certificate Authorities



- Trust on First Use (TOFU)



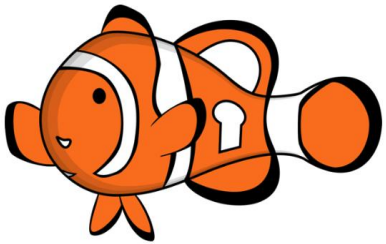
- Manual verification



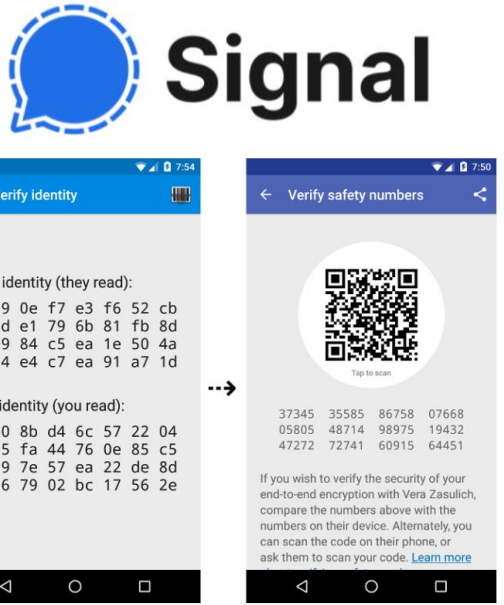
Schemes for Manual Fingerprint Verification

- QR Codes & Safety Numbers



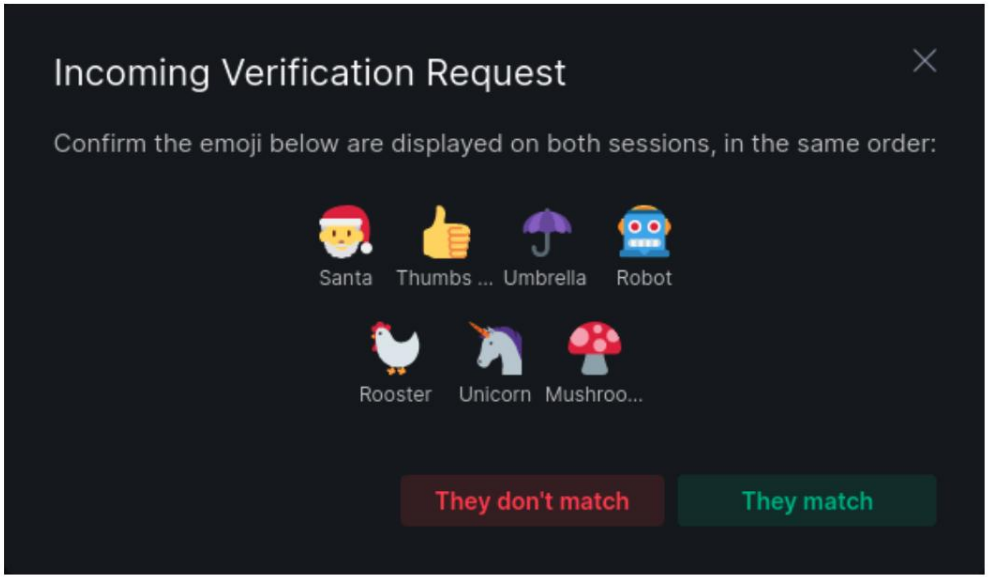






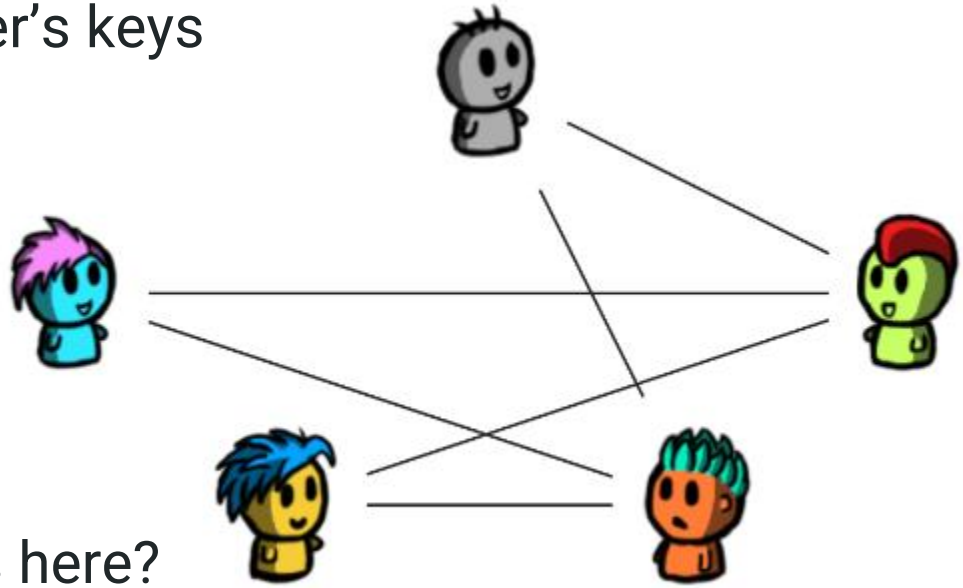
Schemes for Manual Fingerprint Verification

- Emoji



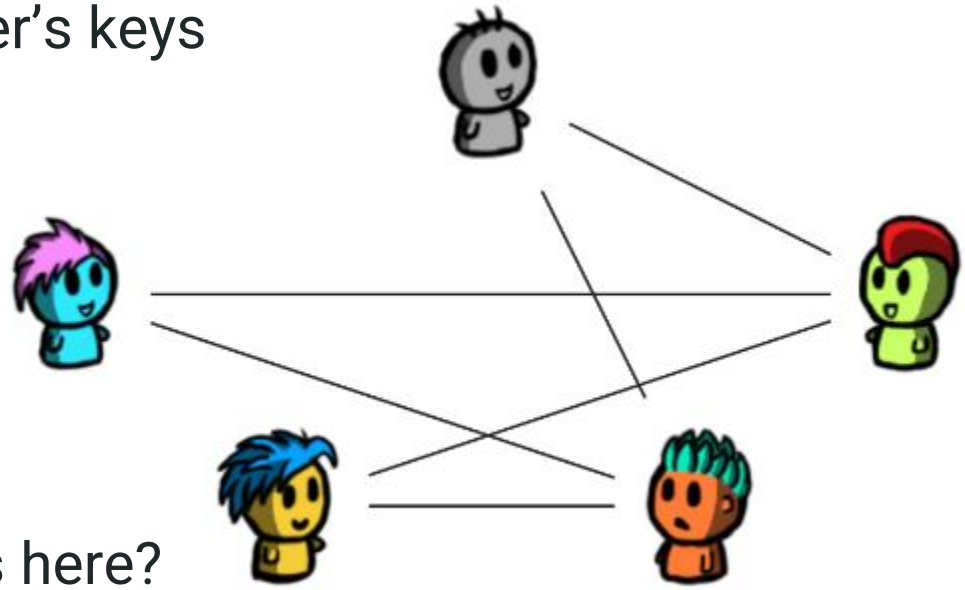
Web of Trust

- Users verify and sign each other's keys
- If Alice has verified Bob
  - and Bob has verified Carol
  - maybe Alice can trust that Carol's key is correct
- PGP uses this
- Can you see any privacy issues here?



Web of Trust

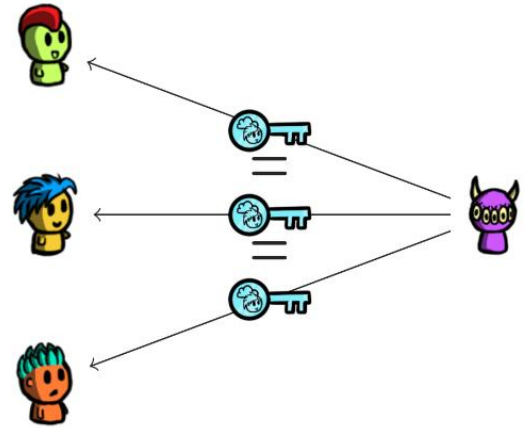
- Users verify and sign each other's keys
- If Alice has verified Bob
  - and Bob has verified Carol
  - maybe Alice can trust that Carol's key is correct
- PGP uses this
- Can you see any privacy issues here?
  - Users publish their social graphs

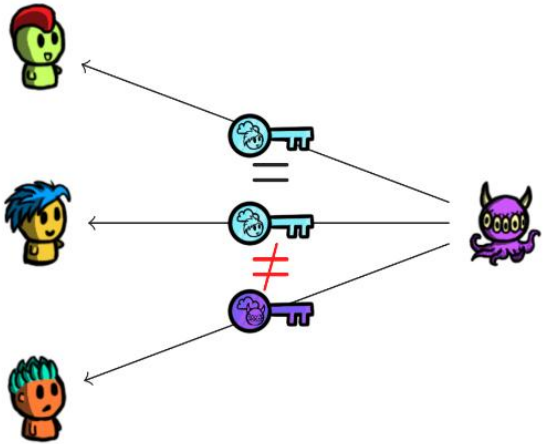


Key Transparency (KT)

Intuition of Key Transparency

- Most users don't do manual verification
- Can't automate key *correctness* checks
  - Correctness is a social question
- Can automate key *consistency* checks
  - If multiple users query for the same user's key, do they get the same result?
  - How? With a Verifiable Key Directory



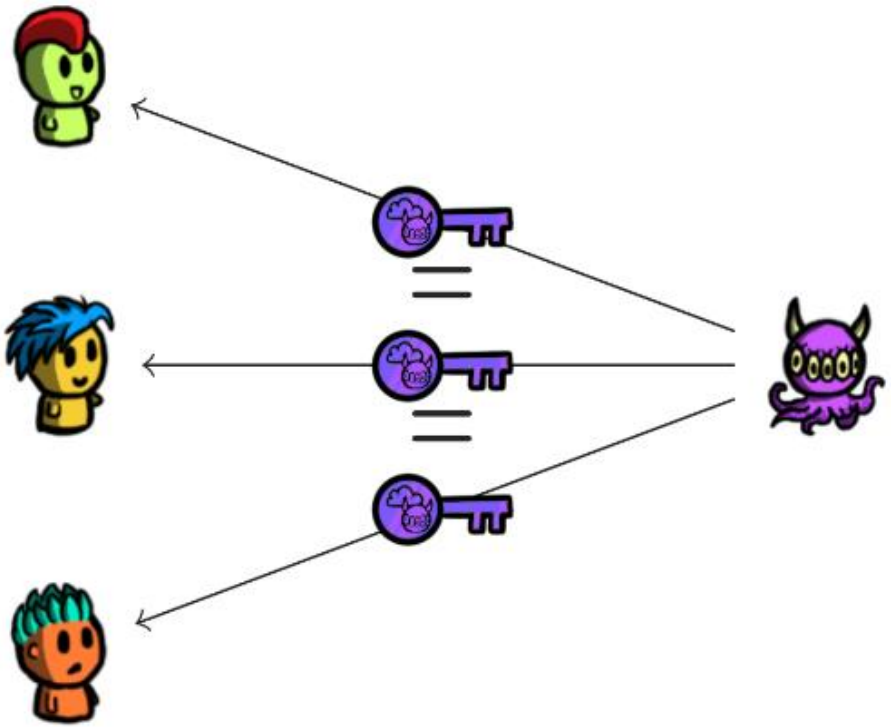


Verifiable Key Directory (VKD)

- Key server commits to the set of keys it knows
  - Before the keys are actually requested
  - Using an efficient data structure called a Merkle tree
- When Bob retrieves Alice's key from the VKD, he checks that the key is consistent with the server commitment
  - If the key is not consistent with the commitment, he rejects the key
- Users compare commitments with each other and/or with public auditors to ensure that everyone has the same view of the VKD

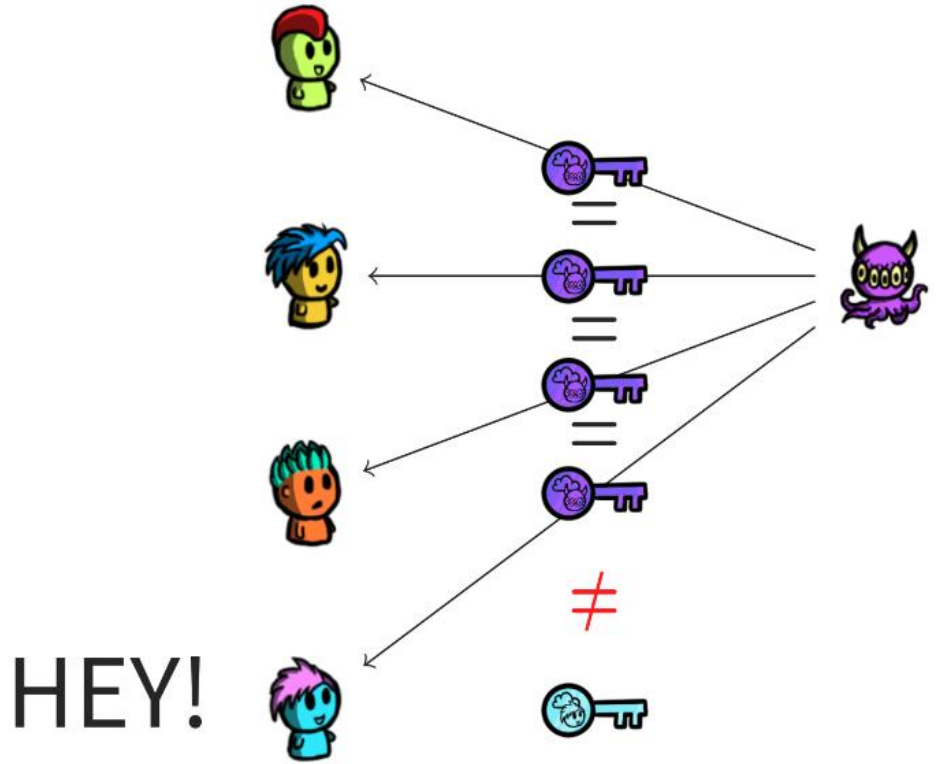
Verifiable Key Directory (VKD)

- Of course, the server might consistently give the wrong key to everyone...



Verifiable Key Directory (VKD)

- Of course, the server might consistently give the wrong key to everyone...
  - Alice's device should request Alice's own key to check that it's correct



Key Transparency (KT) Summary

- Aims to automate *consistency* checks
  - So users have reasonable baseline security against MITM attacks without having to take manual action
- Coming Soon™ to a messaging app near you
  - iMessage and WhatsApp both have implementations
    - (...with some issues)
  - Signal is working on implementing KT