CS459/698
Privacy, Cryptography, Network and Data Security

Secure Messaging

Spring 2025, Monday/Wednesday 2:30pm-3:50pm

Today

- Secure Messaging Goals
- PGP
  - PGP Keys
  - Problems with PGP
- OTR
- Signal

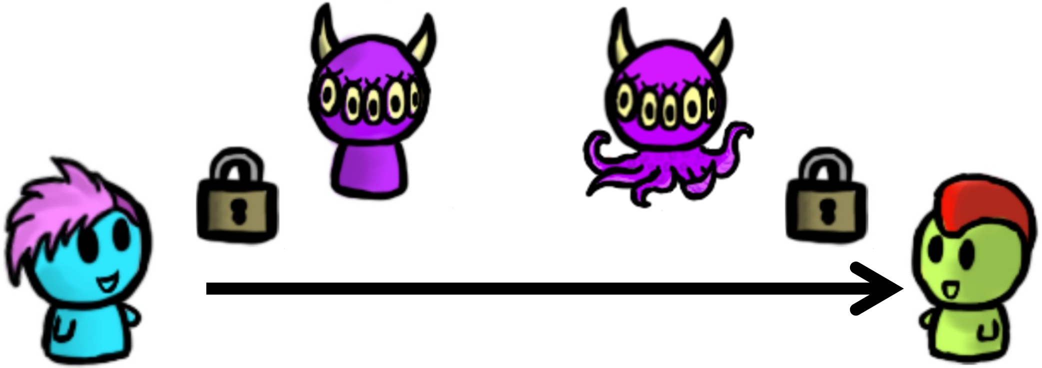
Secure Messaging Goals

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)
- **Authentication:** Bob knows Alice wrote the message
    - Non-repudiation?

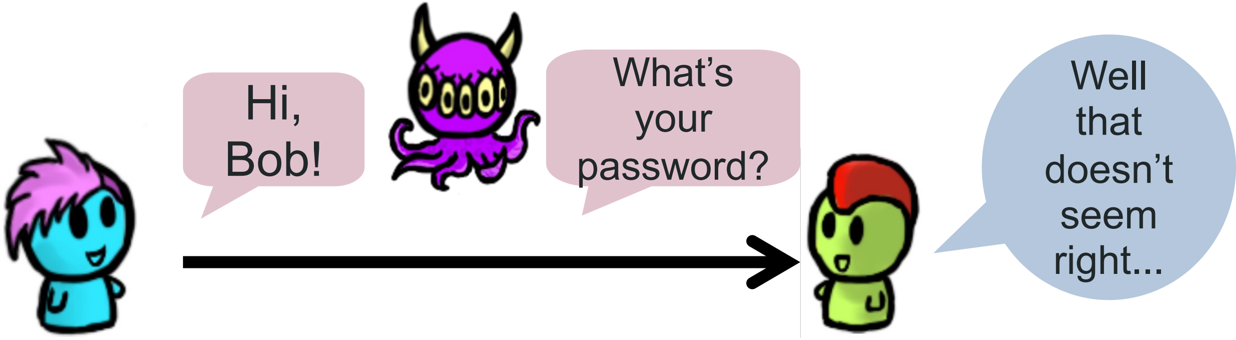
Secure Messaging Goals

- **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)
- **Authentication:** Bob knows Alice wrote the message
  - Non-repudiation?



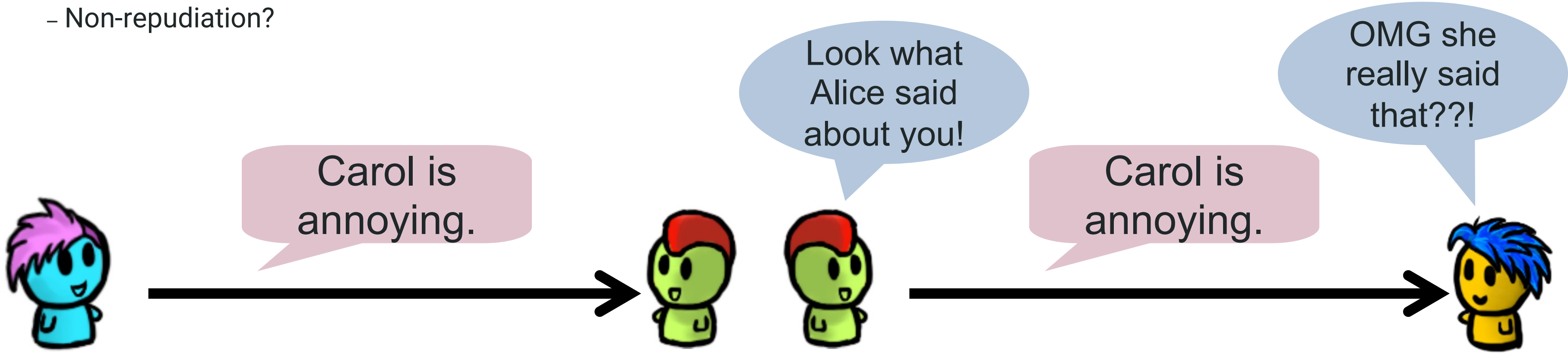
Secure Messaging Goals

- **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)
- **Authentication:** 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)
- **Authentication:** Bob knows Alice wrote the message
  - Non-repudiation?



Pretty Good Privacy

A bit of history on PGP

- Public-key (actually 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 munitions**.
    - 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

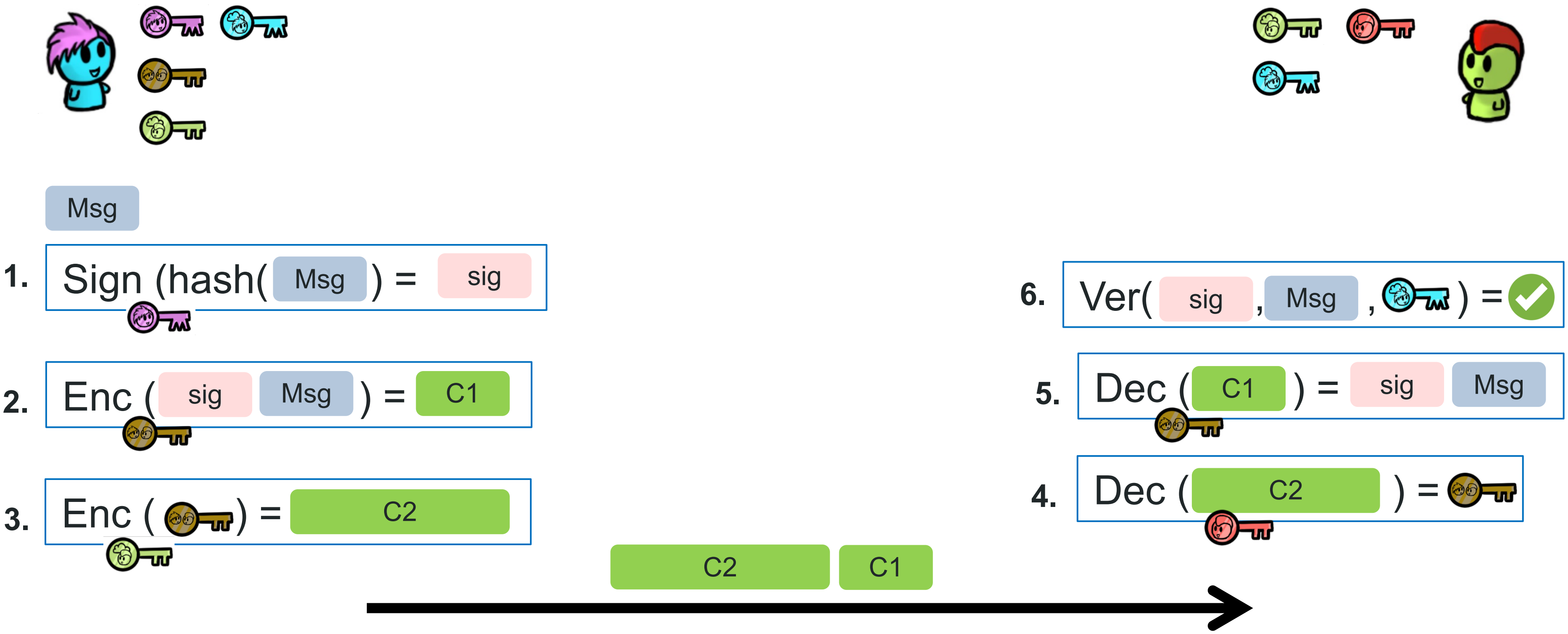
What do you mean by “PGP”?

- **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, ...

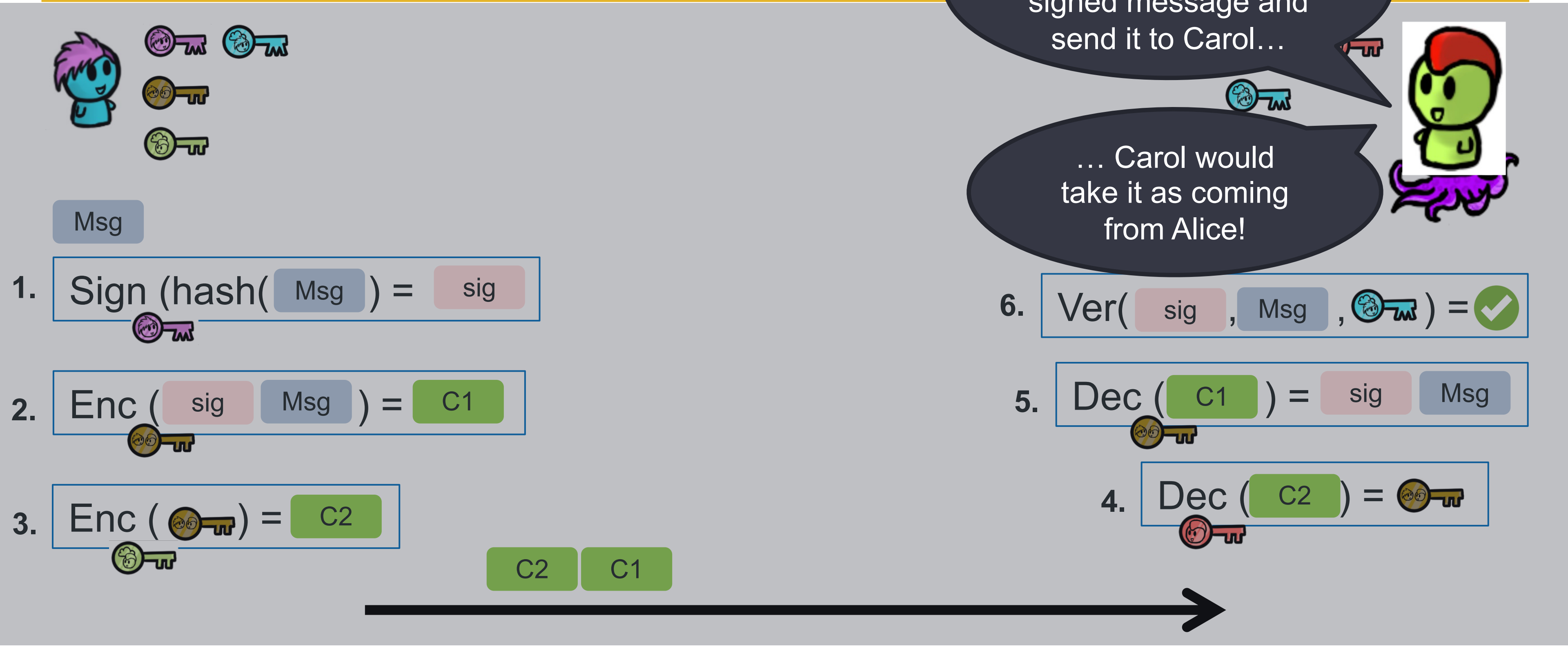
PGP is a hybrid crypto scheme!

- 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

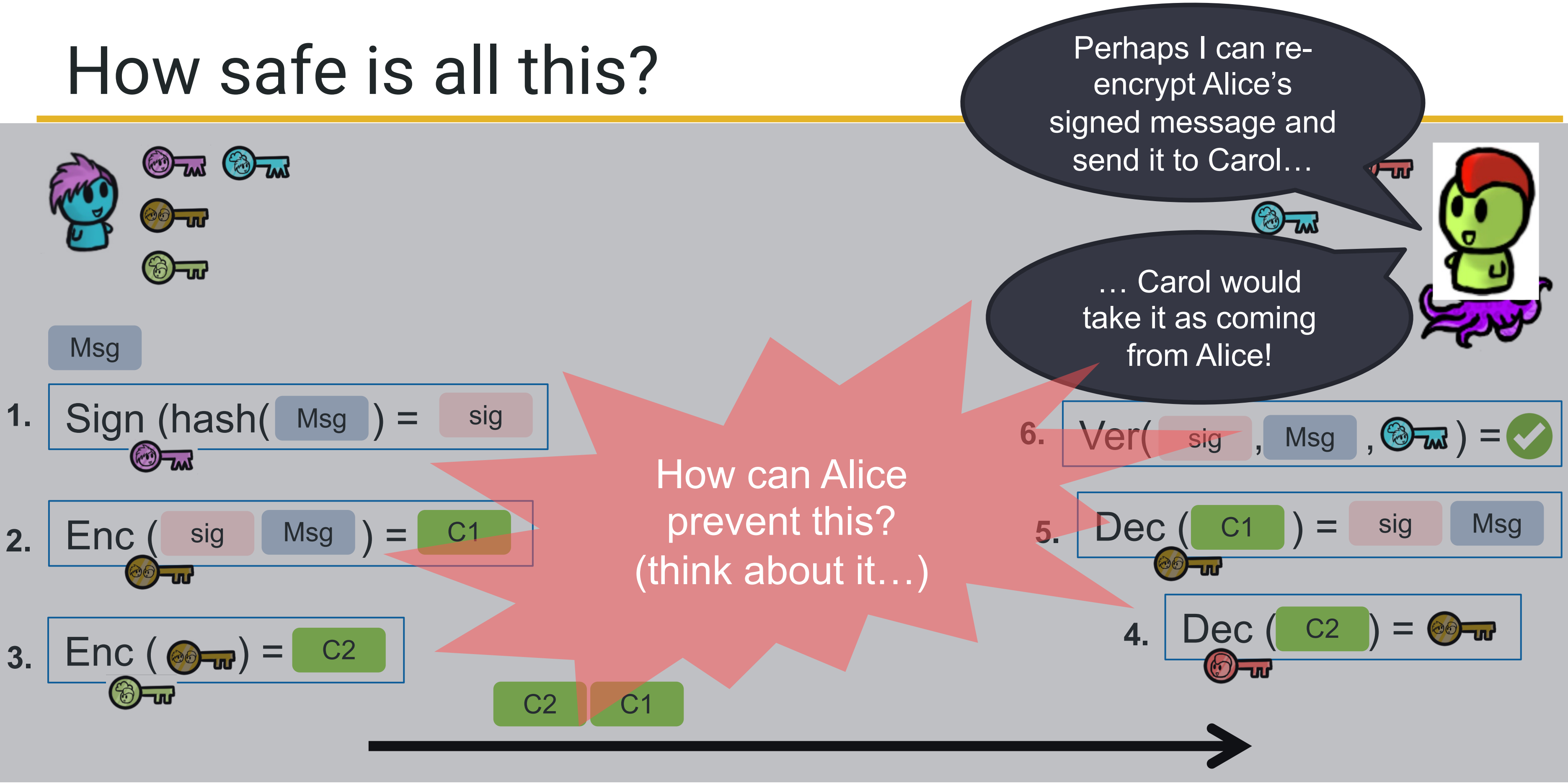
PGP is a hybrid crypto scheme!



How safe is all this?



How safe is all this?



Encrypted Messaging Goals and PGP

- Confidentiality

  C2 C1

- Integrity

  sig

- Authentication

  sig

  - Non-repudiation

    sig

PGP Keys

PGP Keys

Each person has at least 2 keypairs:

- One for signatures
  - Public key used to verify
  - Private key used to sign
- One for encryption
  - Public key used to encrypt
  - Private key used to decrypt

```
pub   rsa4096 2023-01-27 [SC] [expires: 2023-02-26]
      EF22E516EA9C43B7A67E4FB41CD25603C14C0D05
uid           [ultimate] Alice <alice@example.com>
sub   rsa4096 2023-01-27 [E] [expires: 2023-02-26]
```

Obtaining Keys

- How does Alice get Bob’s public key?
  - –Download from Bob’s website
  - –Download from a keyserver
  - –Bob sends it via email
  - –Other channel
- How does Alice know it’s Bob’s **authentic** key?

Verifying Public Keys

- Alice and Bob would rather not have to trust CAs
- They can **compare keys** (e.g., in-person)

- But keys are big and unwieldy!

-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGPUBx4BEADa3JsMGX9GKriACgI1vvokxOc8ltbHSl7aYYMZu5UzgCxYy29n
7YDGDiwN23ibyi8Gf36HNJ6mQuzgUBJ7T54ed8pEf1rtMWL+7OoMNRNaFX6vosT5
3pFn+CiRY5avIGPkut8YdYrkaLixshjakYehmwwWVcVMBBGfrP3pR93dKWbET2EN
RMDSVBO6AzPnjedZmGpJUqp8UPxEP8JoTCn0xAv4ugjM6VE6xxb/Cj15I/5PsIhx
76LPqSsPUwRzKQ9stP8YjTX+Ol91+GNqLhtdmy5yXPD9F/NO+fhQVwvUZ0oJ544a
KeFDQ/G9GKJfJzTIhvQn9BdkZpff5Kjzun0+4HNk0msB5S8BItdPpuc3qs+rkL6W
aAnXUS9j7mB3Gf58fjJu+1gMP5dXG16nduB/W3SuH2/XSympjSm6PkuNcSMI0XEN
FCUH/aoRjZQV/Xi5laQHg+cbEtLRACdkaAHNNjxGDXkzjbuYzjtv3hPMvNiBF897
PvihCO2w4pXBQ7rpxzn6OvU1iawfrmdZQA2tRZOSN2Cpti3KJ0OzKzfGT0VFRaVq
NfEy26ZtEPAZjhgBJDo8SLxJkshrMLhNnIobR/BLng1v/xSrjPTAVE/sK032GfqZ
uynR6zO+rVcwAKz3g/aK5kknPG/Or4KdEhsmOKuPgATSduGo96t299dRqQARAQAB
tBlBbGljZSA8YWxpY2VAZXhhbXBsZS5jb20+iQJXBBMBCABBFiEE7yLlFuqcQ7em
fk+0HNJWA8FMDQUFAmPUBx4CGwMFCQAnjQAFCwkIBwICIgIGFQoJCAsCBBYCAwEC
HgcCF4AACgkQHNJWA8FMDQV3LQ/8CnyOARm+seUp4ShUo5xqIlEMPG6F+VbBE45G
XGiEr/PeMbdTJtkrO0Qzsx0/tVYKJGiLE5D9W/1TaqzAkmnsyvhF0wp3XZQGeqlt
U9mPpBQkzAfzwW21++3CK48WcCtb5mRh+O9Z7jwF0aEYDOKxO2og6a9132kUp66n
CctBy+h6ucBVMMTZS0jFr5YHFZJKa/IyQ6ODgkv+fIwfPZm2N93jHejIdrKSVtzi
Yb5tiXqGDwoljSlxhlVA6pX03CtENKqrpDPS0tM70AdmVSmjQgn7AR3UtBJn4JMb
iC+/yKD2JIGLS1R5RKvovJ1BBQHU7FATcrKFL4SORQ5o5iaEteMsFLLbBMomrs23
oNuS/wmeWkUOG76uvjQnuAr/Bc7DF4lhY/WpZGDAIayA9v9TWMUMzxDjMwmfeK+j
OlcJwj0BO6GbMBBNlr76ae+zWpJeqZrjv7S7H+h0bOi8n0PBKrTxbGLM7wg/r9ii
qEm4pHT5P0i6WBr3PYu/PoyEnPlKonxSv9kOJXGyjDcdV6vjBA6c37mFFs0Ffk8A
s/x3V85+0YK34RbDVDqm5+V42Lo5DP49KdBV1dp+O07nWRJDsOroFarbMcPCCWiJ
i0p4+r9nU9Hx8k6mjustyjZBgplmDhBnCo5hAaAytuOLTU3wKwmhq8ONCJhKYRXo
+88+0P65Ag0EY9QHHgEQAOFF4x8GKiSCjk5jUxL87s0nkm9OGxtpx8L4drn9rFtu
u6cP7XcOJ0ngxF4HufcL6vNfPMF5knU6ezXUgMvOseFVT30VC6uF39OrqOj26va/
LcCYzKaIWFLKyuBvtLDuPUdANhplQhH7s4FQIvTPUO+saCAqJDJtOsq/F/n+Gttz
DxNdPbsTC5oESkgfhyednT9gZpCsxc9Gd3mDyDDkMGyWaEf4bWjdjX2NEj6TuezY
ijyqtYBHKf9eNSmPY9SEbV9HIMLgZa/R4mrtZ+AMya2lTuyBXi6oo+oElS71cefD
BFajeOKH0MHtPKQvkagyetI6I5Ta+6Ekqoy5Oc90s85UdUIZZkCaZ5zA8vrkhLNh
KvJ90Uf5IVuoe+Ci6wpvZZQhpIumX+eRMSX1U4hBahB5z+fLe3YUCn5rDwEFmSG2
EAMRDF5QG7L5dDMS6Z3PRD4a4ZPzF/1TyjiTpNUbF3N3uOUIT/1rChghJLfm79Dl
O9MSYRdOFPVIIumqWIiv862zXOr8dqwnIKB9uDWMHGnEkFtlseC0WrsbRaeMHDFc
7A/bNCocDrA8x18GieIkVTMhuFMc77WiN43rjYSLr17W2V0KqIN0NHYCSsGOhC4z
0aJcDDJLvdkt4AriXpmhSmMOWZsvbIrT9i5voY8GIEbltQ5xppOUGZ+3vfq0UwER
ABEBAAGJAjwEGAEIACYWIQTvIuUW6pxDt6Z+T7Qc0lYDwUwNBQUCY9QHHgIbDAUJ
ACeNAAAKCRAc0lYDwUwNBR0JEACAJ8LSN8YlnrKq/9JqJy6qkoLTr0r5Yvz7Fm/F
KRP7vDicOiKGH3NwsrBE3+r7UB8MWWjOrdtWLd7a5AaswEtTSXKHrpzSC/s8kn1m
POtR/vSaIlfb6qjXAQrK0ZhWhoD4YsRBY57Xe9EhOup5y6eUeFbGMS80HvLrApju
IUvKJNdpD+21U0Ohu16JKAuIhyKFfpXVtjH3lxnagBl9UOlLG0h4y9aMa4RwAmY0
Z4h9StZcQhMOoKeL0dovHoS5BvyDIa91TpennGhM+AeEI1VPdRfpaa1O4srGMUQX
kjtnHNdMVHEzMSy5vwygJEIXMBpkFqZF/CCOhqvqM+RQgh0sTATa6ixVRNyml241
PqMbZn7JYMZ0flbMPtD2qd9lT6rKfXUzLtRQswhXpcVi+8Mgsb53JyKQlpigIdu0
z+VOq7ObHuwwPCi1ohJ8Q3SfaKIynfhACVOlDr8l89rZ3mVbTiLMvKKyKYEijpB/
idbN3QtUuPYlnALlcN4883DwzMO5ZQ8CPc3/6yOQOUytTUpNo143XcQ//OwC3Tmm
YsMnvZVhlY6MoiQ7cXDJvwRUOTU4IlG6qkwmbeEO7zatGHXv/agSxpRuLzIhZHem
fI11i44fYII2ZxWWVr2vQ6T9oELTyCjJTeGxaot0thOxxQ3pdXavxuYdG84zZyMd
i96dvg==
=tJAW
-----END PGP PUBLIC KEY BLOCK-----

Verifying Public Keys

- Alice and Bob would rather <u>not have to</u> trust CAs
- They can **compare keys** (e.g., in-person)

Can we do better?

- But keys are big and unwieldy!

-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGPUBx4BEADa3JsMGX9GKriACgI1vvokxOc8ltbHSl7aYYMZu5UzgCxYy29n
7YDGDiwN23ibyi8Gf36HNJ6mQuzgUBJ7T54ed8pEf1rtMWL+7OoMNRNaFX6vosT5
3pFn+CiRY5avIGPkut8YdYrkaLixshjakYehmwwWVcVMBBGfrP3pR93dKWbET2EN
RMDSVBO6AzPnjedZmGpJUqp8UPxEP8JoTCn0xAv4ugjM6VE6xxb/Cj15I/5PsIhx
76LPqSsPUwRzKQ9stP8YjTX+Ol91+GNqLhtdmy5yXPD9F/NO+fhQVwvUZ0oJ544a
KeFDQ/G9GKJfJzTIhvQn9BdkZpff5Kjzun0+4HNk0msB5S8BItdPpuc3qs+rkL6W
aAnXUS9j7mB3Gf58fjJu+1gMP5dXG16nduB/W3SuH2/XSympjSm6PkuNcSMI0XEN
FCUH/aoRjZQV/Xi5laQHg+cbEtLRACdkaAHNNjxGDXkzjbuYzjtv3hPMvNiBF897
PvihCO2w4pXBQ7rpxzn6OvU1iawfrmdZQA2tRZOSN2Cpti3KJ0OzKzfGT0VFRaVq
NfEy26ZtEPAZjhgBJDo8SLxJkshrMLhNnIobR/BLng1v/xSrjPTAVE/sK032GfqZ
uynR6zO+rVcwAKz3g/aK5kknPG/Or4KdEhsmOKuPgATSduGo96t299dRqQARAQAB
tBlBbGljZSA8YWxpY2VAZXhhbXBsZS5jb20+iQJXBBMBCABBFiEE7yLlFuqcQ7em
fk+0HNJWA8FMDQUFAmPUBx4CGwMFCQAnjQAFCwkIBwICIgIGFQoJCAsCBBYCAwEC
HgcCF4AACgkQHNJWA8FMDQV3LQ/8CnyOARm+seUp4ShUo5xqIlEMPG6F+VbBE45G
XGiEr/PeMbdTJtkrO0Qzsx0/tVYKJGiLE5D9W/1TaqzAkmnsyvhF0wp3XZQGeqlt
U9mPpBQkzAfzwW21++3CK48WcCtb5mRh+O9Z7jwF0aEYDOKxO2og6a9132kUp66n
CctBy+h6ucBVMMTZS0jFr5YHFZJKa/IyQ6ODgkv+fIwfPZm2N93jHejIdrKSVtzi
Yb5tiXqGDwoljSlxhlVA6pX03CtENKqrpDPS0tM70AdmVSmjQgn7AR3UtBJn4JMb
iC+/yKD2JIGLS1R5RKvovJ1BBQHU7FATcrKFL4SORQ5o5iaEteMsFLLbBMomrs23
oNuS/wmeWkUOG76uvjQnuAr/Bc7DF4lhY/WpZGDAIayA9v9TWMUMzxDjMwmfeK+j
OlcJwj0BO6GbMBBNlr76ae+zWpJeqZrjv7S7H+h0bOi8n0PBKrTxbGLM7wg/r9ii
qEm4pHT5P0i6WBr3PYu/PoyEnPlKonxSv9kOJXGyjDcdV6vjBA6c37mFFs0Ffk8A
s/x3V85+0YK34RbDVDqm5+V42Lo5DP49KdBV1dp+O07nWRJDsOroFarbMcPCCWiJ
i0p4+r9nU9Hx8k6mjustyjZBgplmDhBnCo5hAaAytuOLTU3wKwmhq8ONCJhKYRXo
+88+0P65Ag0EY9QHHgEQAOFF4x8GKiSCjk5jUxL87s0nkm9OGxtpx8L4drn9rFtu
u6cP7XcOJ0ngxF4HufcL6vNfPMF5knU6ezXUgMvOseFVT30VC6uF39OrqOj26va/
LcCYzKaIWFLKyuBvtLDuPUdANhplQhH7s4FQIvTPUO+saCAqJDJtOsq/F/n+Gttz
DxNdPbsTC5oESkgfhyednT9gZpCsxc9Gd3mDyDDkMGyWaEf4bWjdjX2NEj6TuezY
ijyqtYBHKf9eNSmPY9SEbV9HIMLgZa/R4mrtZ+AMya2lTuyBXi6oo+oElS71cefD
BFajeOKH0MHtPKQvkagyetI6I5Ta+6Ekqoy5Oc90s85UdUIZZkCaZ5zA8vrkhLNh
KvJ90Uf5IVuoe+Ci6wpvZZQhpIumX+eRMSX1U4hBahB5z+fLe3YUCn5rDwEFmSG2
EAMRDF5QG7L5dDMS6Z3PRD4a4ZPzF/1TyjiTpNUbF3N3uOUIT/1rChghJLfm79Dl
O9MSYRdOFPVIIumqWIiv862zXOr8dqwnIKB9uDWMHGnEkFtlseC0WrsbRaeMHDFc
7A/bNCocDrA8x18GieIkVTMhuFMc77WiN43rjYSLr17W2V0KqIN0NHYCSsGOhC4z
0aJcDDJLvdkt4AriXpmhSmMOWZsvblrT9i5voY8GIEbltQ5xppOUGZ+3vfq0UwER
ABEBAAGJAjwEGAEIACYWIQTvIuUW6pxDt6Z+T7Qc0lYDwUwNBQUCY9QHHgIbDAUJ
ACeNAAAKCRAc0lYDwUwNBR0JEACAJ8LSN8YlnrKq/9JqJy6qkoLTr0r5Yvz7Fm/F
KRP7vDicOiKGH3NwsrBE3+r7UB8MWWjOrdtWLd7a5AaswEtTSXKHrpzSC/s8kn1m
POtR/vSaIlfb6qjXAQrK0ZhWhoD4YsRBY57Xe9EhOup5y6eUeFbGMS80HvLrApju
IUvKJNdpD+21U0Ohu16JKAuIhyKFfpXVtjH3lxnagBl9UOlLG0h4y9aMa4RwAmY0
Z4h9StZcQhMOoKeL0dovHoS5BvyDIa91TpennGhM+AeEI1VPdRfpaa1O4srGMUQX
kjtnHNdMVHEzMSy5vwygJEIXMBpkFqZF/CCOhqvqM+RQgh0sTATa6ixVRNyml241
PqMbZn7JYMZ0flbMPtD2qd9lT6rKfXUzLtRQswhXpcVi+8Mgsb53JyKQlpigIdu0
z+VOq7ObHuwwPCi1ohJ8Q3SfaKIynfhACVOlDr8l89rZ3mVbTiLMvKKyKYEijpB/
idbN3QtUuPYlnALlcN4883DwzMO5ZQ8CPc3/6yOQOUytTUpNo143XcQ//OwC3Tmm
YsMnvZVhlY6MoiQ7cXDJvwRUOTU4IlG6qkwmbeEO7zatGHXv/agSxpRuLzIhZHem
fI11i44fYII2ZxWWVr2vQ6T9oELTyCjJTeGxaot0thOxxQ3pdXavxuYdG84zZyMd
i96dvg==
=tJAW
-----END PGP PUBLIC KEY BLOCK-----

Verifying Fingerprints

- Hash the key to get the key **fingerprint**, and compare key fingerprints instead!

- Much shorter strings to compare:
  - EF22 E516 EA9C 43B7 A67E 4FB4 1CD2 5603 C14C 0D05

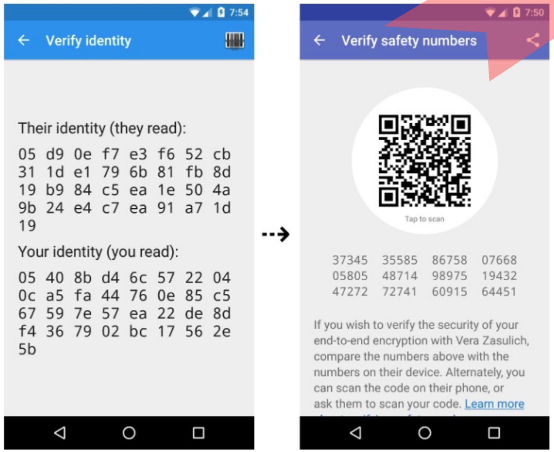
- With a good hash function, no two key fingerprints should collide
  - **Q:** What if you only use part of the fingerprint?

Schemes for Manual Fingerprint Verification

- QR Codes & Safety Numbers

Signal

Alice and Bob's safety numbers are combined in a single number

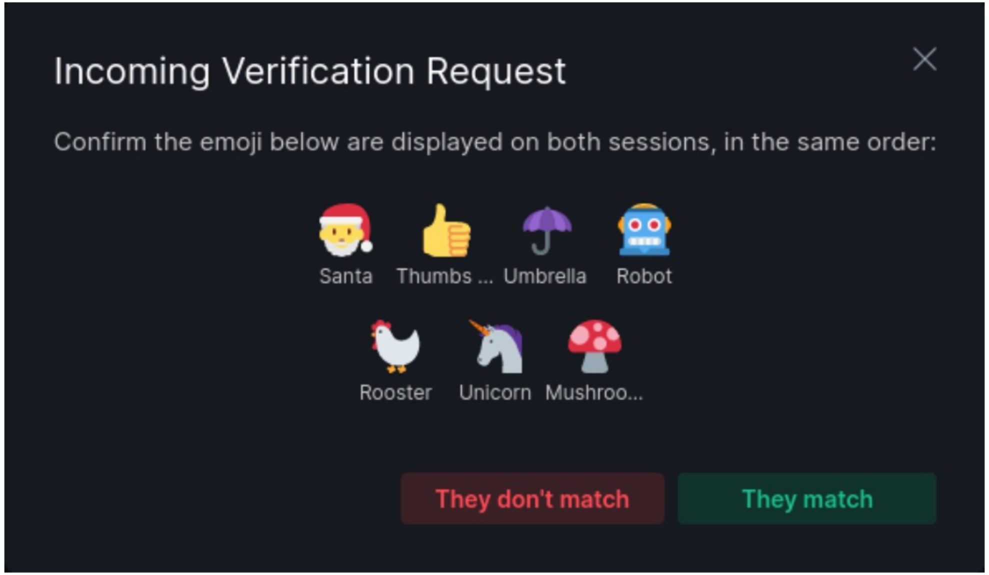


XMPP

Schemes for Manual Fingerprint Verification

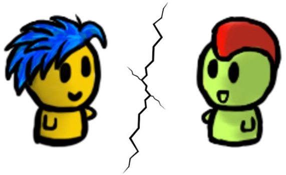
- Emoji

[matrix]

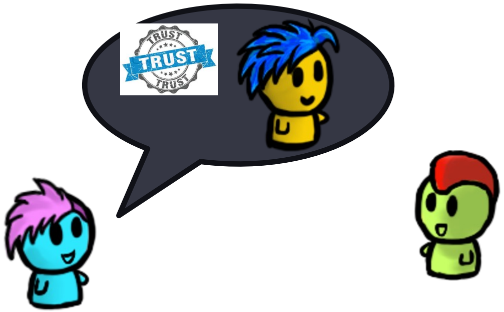


Verifying Public Keys

- Overall, verifying public keys is hard
  - Inconvenient if possible at all
  - Bob and Carol may be far apart and unable to do manual verification…



- **Q:** Would it help if Alice has verified Carol?)



Signing Keys

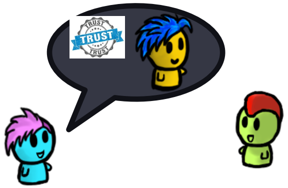
- Once Alice has verified Carol's key, she uses her certification key to **sign** Carol's key (certification key == signature key)

- This is effectively the same as Alice signing a message saying *"I have verified that the key with [Carol's fingerprint] belongs to Carol"*

- Carol can then attach Alice's signature to the key she has published

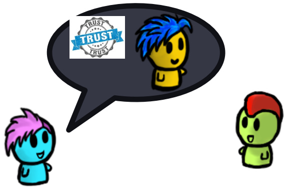
- **Q:** Do you see any potential issues here?

Web of Trust



- Now Alice can act as an introducer for Carol
- If Bob can’t verify Carol herself, but he has already verified Alice (and trusts Alice to introduce him to other people):
  - Bob downloads Carol’s key
  - He sees Alice’s signature on it
  - He is able to use Carol’s key without verifying it himself
- This is called the Web of Trust

Web of Trust



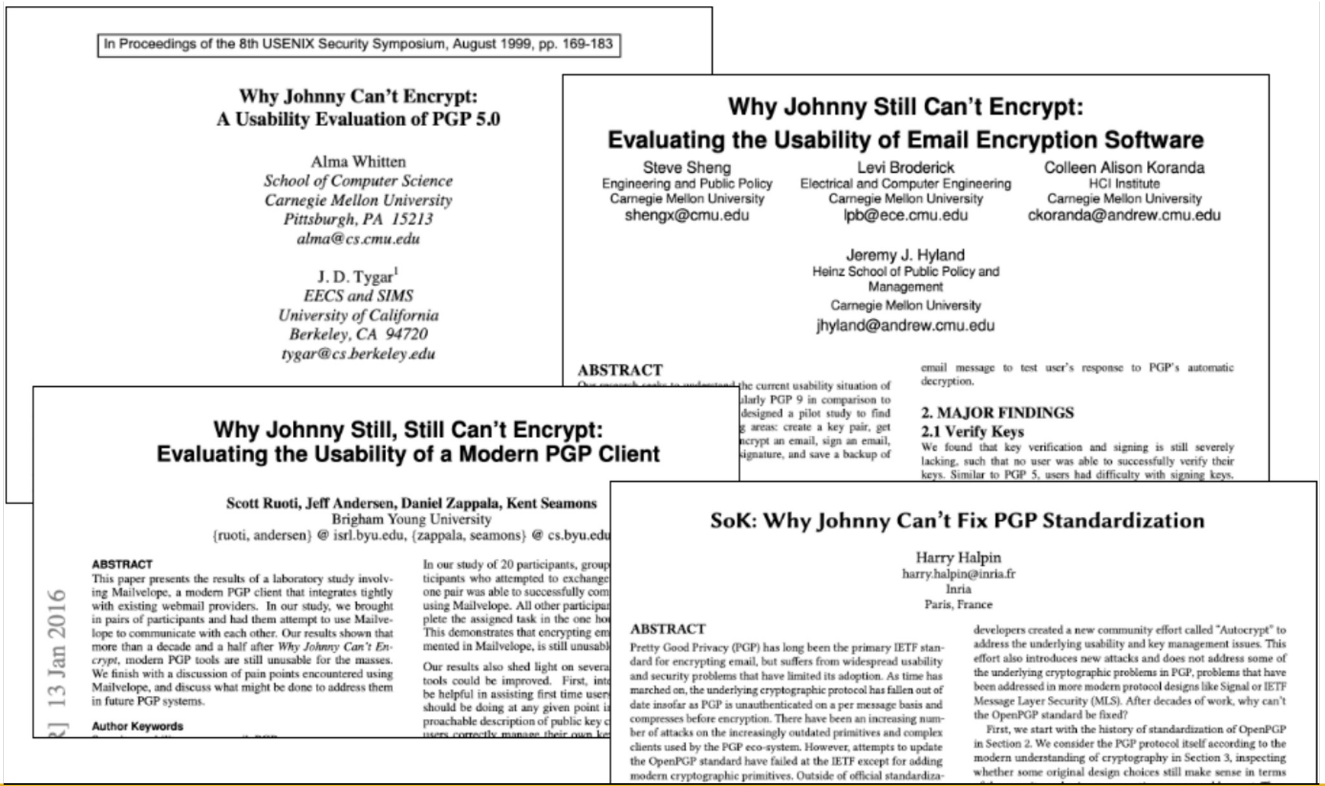
- Now Alice can act as an introducer for Carol
- If Bob can't verify Carol herself, but he has already verified Alice (and trusts Alice to introduce him to other people):
  - Bob downloads Carol's key
  - He sees Alice's signature on it
  - He is able to use Carol's key without verifying it himself
- This is called the Web of Trust

Pretty good, right?

Problems with PGP

Problem #1: Usability

- Hard to use
- Low adoption



Problem #1: Usability

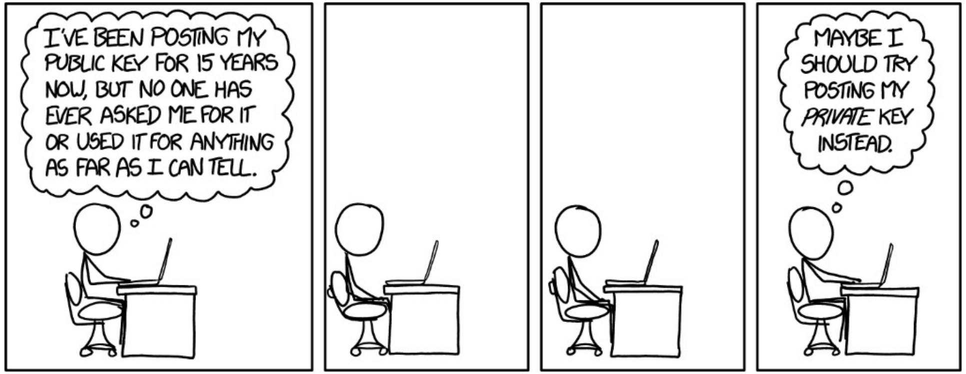
- https://moxie.org/2015/02/24/gpg-and-me.html

–"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 <u>“off-the-record”</u>
    - 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

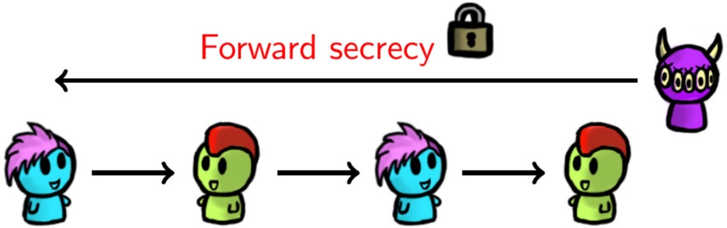
Off-The-Record (OTR) Messaging

OTR

- Messaging (XMPP) extension for encryption with:
  - Forward secrecy
  - Post-compromise security
  - Deniability

Goals of Off-The-Record Messaging

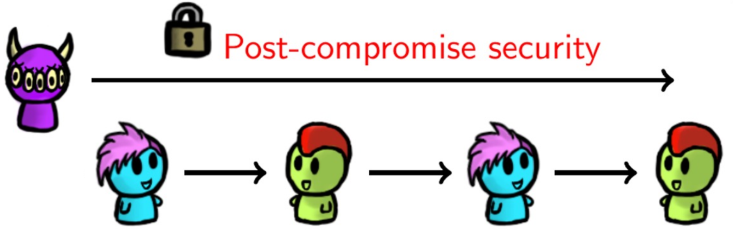
- **(Perfect) Forward secrecy:** a key compromise does not reveal past communication



Goals of Off-The-Record Messaging

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

- **Post-compromise security ~~Backward secrecy~~ ~~Future secrecy~~ ~~Self-healing~~:** a key compromise does not reveal future communication

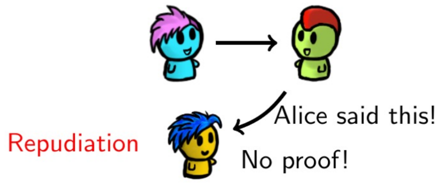


Goals of Off-The-Record Messaging

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

- **Post-compromise security ~~Backward secrecy~~ ~~Future secrecy~~ ~~Self-healing~~:** a key compromise does not reveal future communication

- **Repudiation (deniable authentication):** authenticated communication, but a participant cannot prove *to a third party* that another participant said something

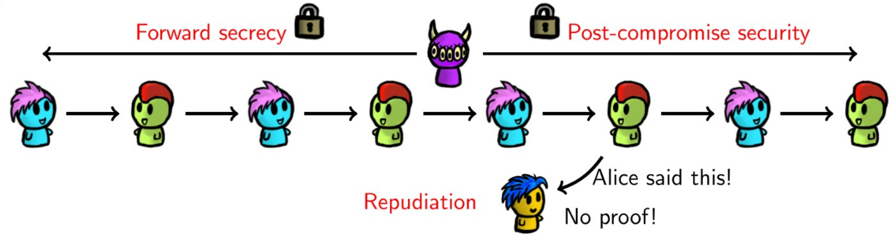


Goals of Off-The-Record Messaging

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

- **Post-compromise security ~~Backward secrecy~~ ~~Future secrecy~~ ~~Self-healing~~:** a key compromise does not reveal future communication

- **Repudiation (deniable authentication):** authenticated communication, but a participant cannot prove *to a third party* that another participant said something



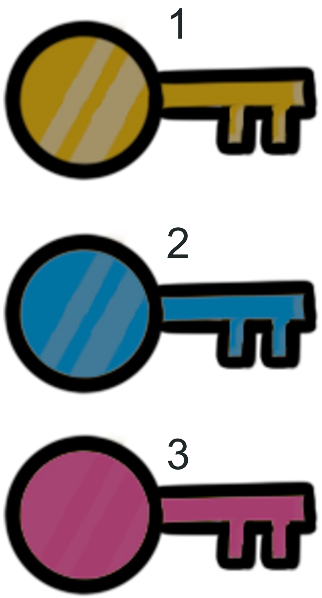
Forward Secrecy

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

  Q: How can we accomplish that?

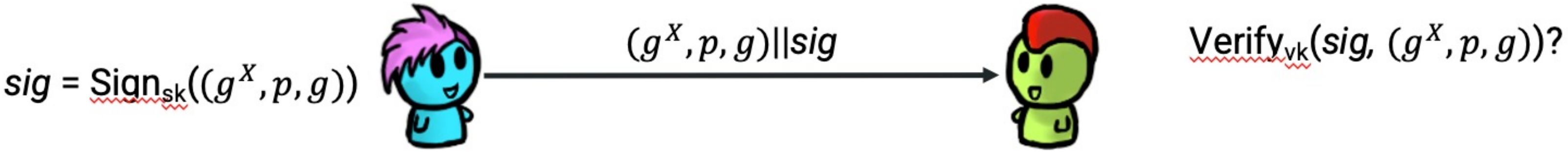
  Change the key!

  Old keys must be securely deleted



Forward Secrecy (one approach)

- Recall Authenticated Diffie-Hellman...

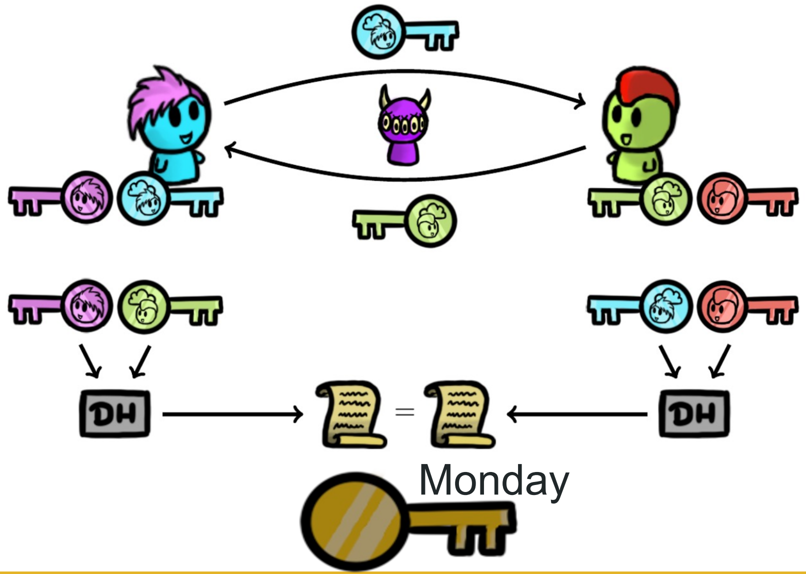


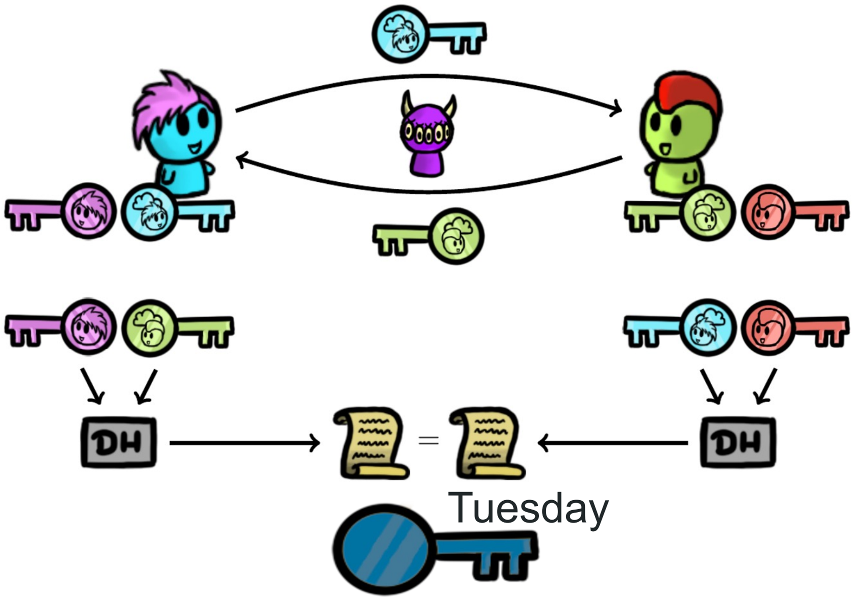
- Alice and Bob find a shared secret used to create a symmetric key

- DH keys can be used for ephemeral (temporary) communication "sessions"
  - Alice and Bob can always make new keys later
  - Call these "session keys"

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…)

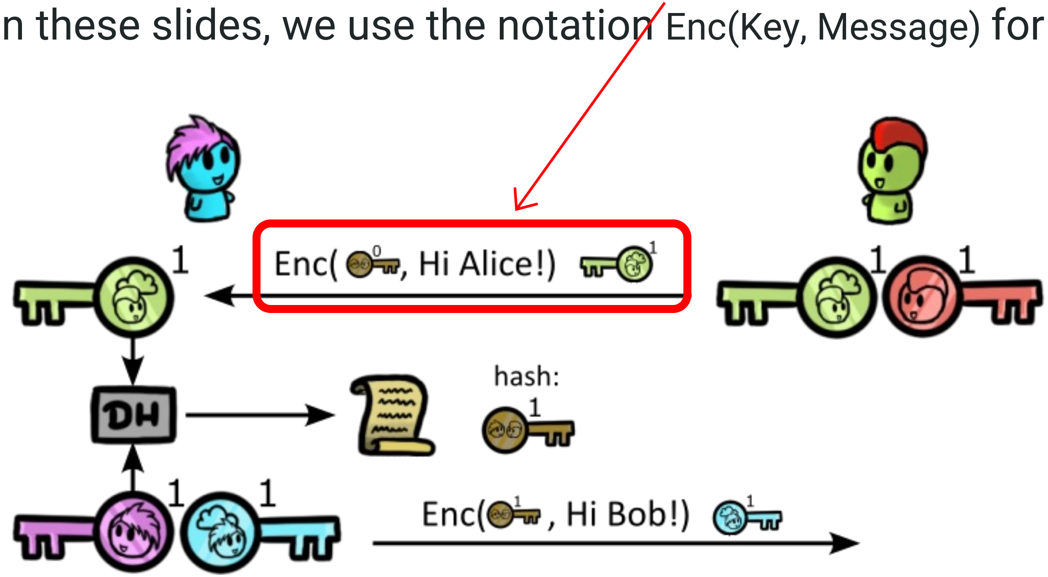
Forward Secrecy in OTR

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

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

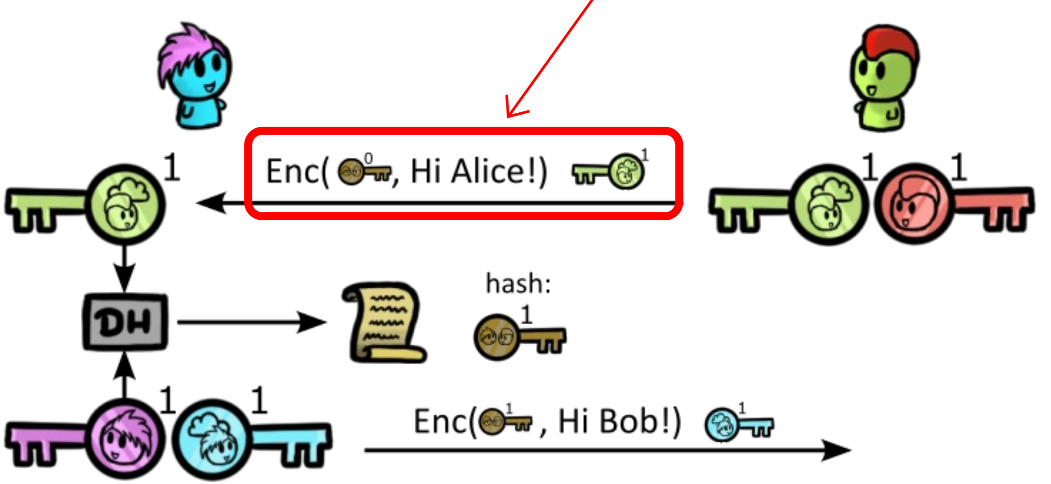
OTR's DH Ratchet (incorrect)

- Assume Alice and Bob have a pre-shared key 
- Assume Alice and Bob can have each other's long-term verification keys and there is a way to "**magically**" authenticate the first message Bob sends (for simplicity)…
- In these slides, we use the notation Enc(Key, Message) for Encryption+MAC with a Key.



OTR's DH Ratchet (incorrect)

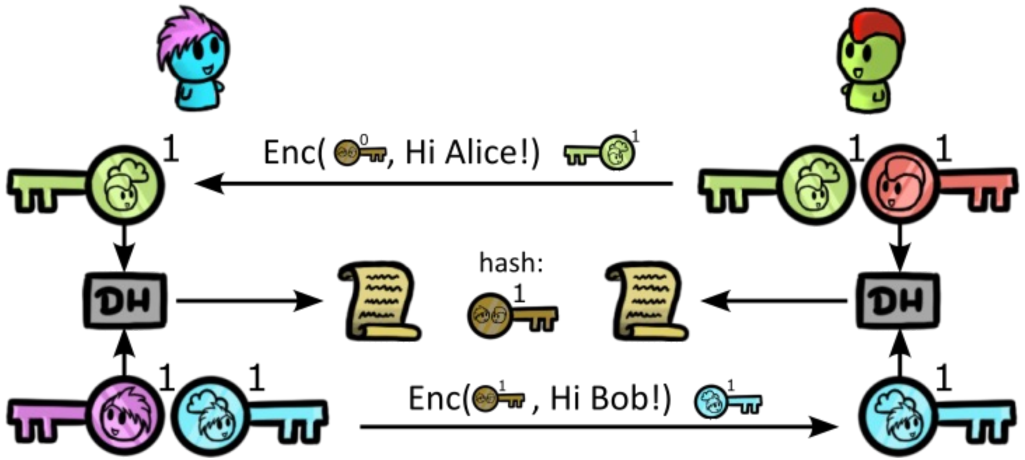
- Assume Alice and Bob have a pre-shared key
- Assume Alice and Bob can have each other's long-term verification keys and there is a way to "**magically**" authenticate the first message Bob sends (for simplicity)…
- In these slides, we use the notation Enc(Key, Message) for Encryption+MAC with a Key.



How does Bob recover the message?

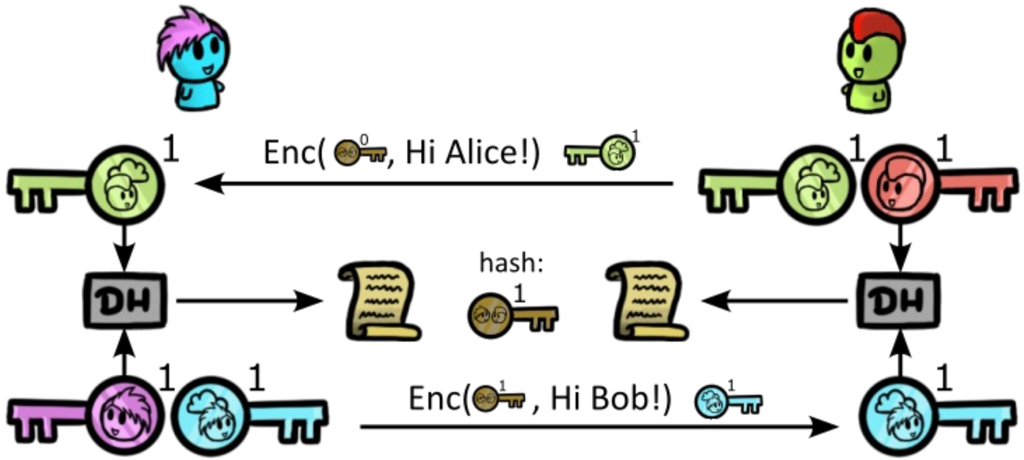
OTR's DH Ratchet (incorrect)

- Assume Alice and Bob have a pre-shared key (key icon, labelled 0)
- Assume Alice and Bob can have each other's long-term verification keys and there is a way to "**magically**" authenticate the first message Bob sends (for simplicity)…
- In these slides, we use the notation Enc(Key, Message) for Encryption+MAC with a Key.



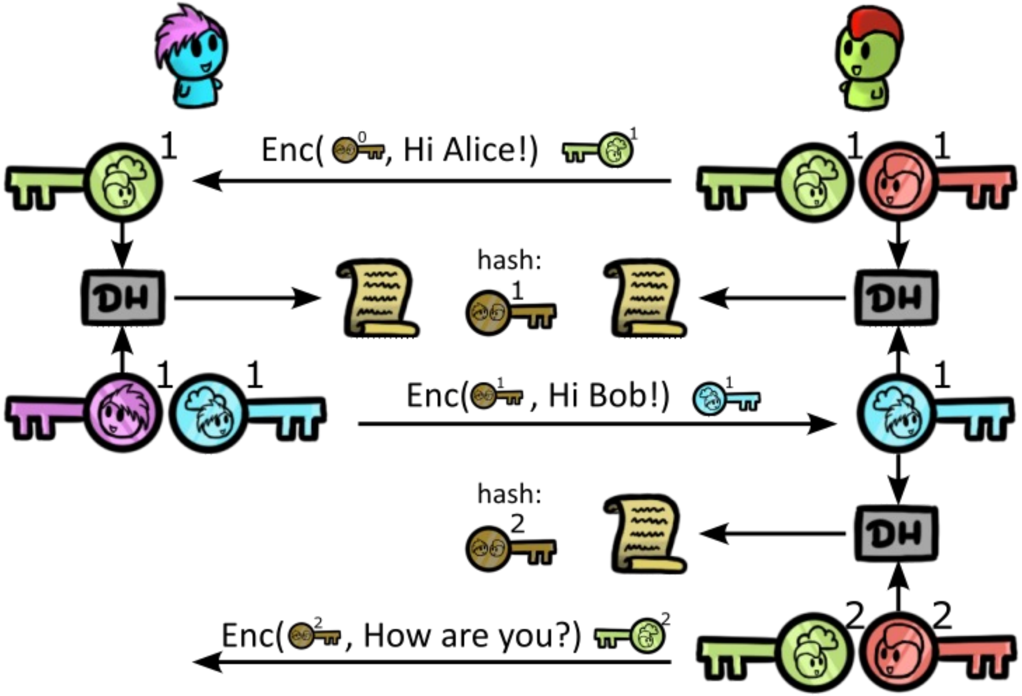
OTR's DH Ratchet (incorrect)

- Assume Alice and Bob have a pre-shared key [key icon, 0]
- Assume Alice and Bob can have each other's long-term verification keys and there is a way to "**magically**" authenticate the first message Bob sends (for simplicity)…
- In these slides, we use the notation Enc(Key, Message) for Encryption+MAC with a Key.



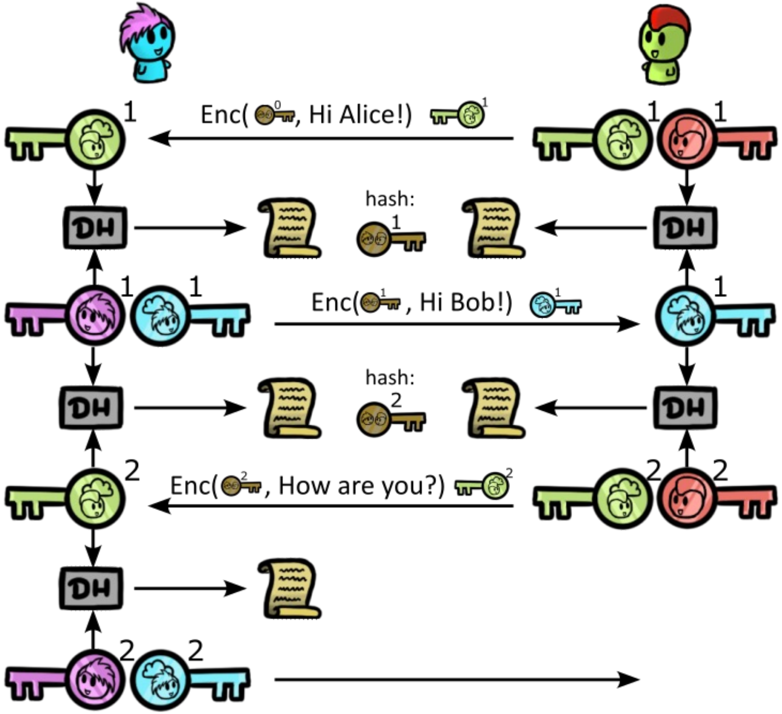
Following this logic, how does Bob reply to Alice?

OTR's DH Ratchet (incorrect)



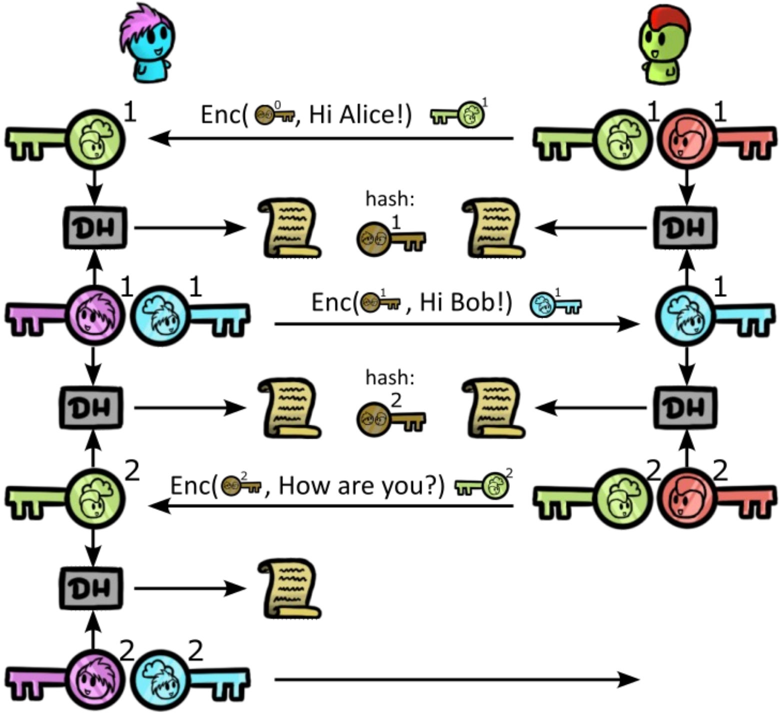
- Alice and Bob automatically create new sessions as they reply to each other
- Also provides post-compromise security
- Awesome! :)
- This is a "ratchet": You can't go backwards

OTR's DH Ratchet (incorrect)



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

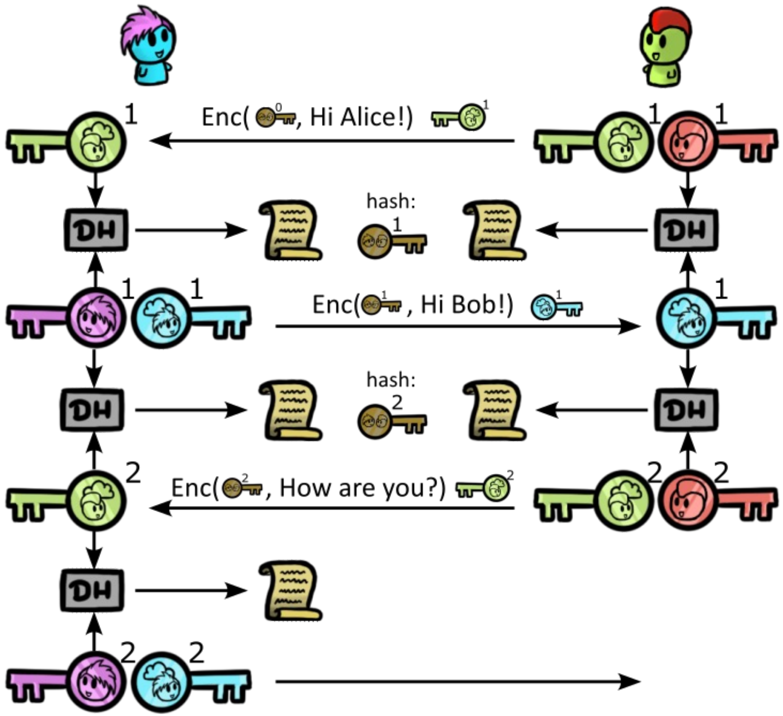
OTR's DH Ratchet (incorrect)



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

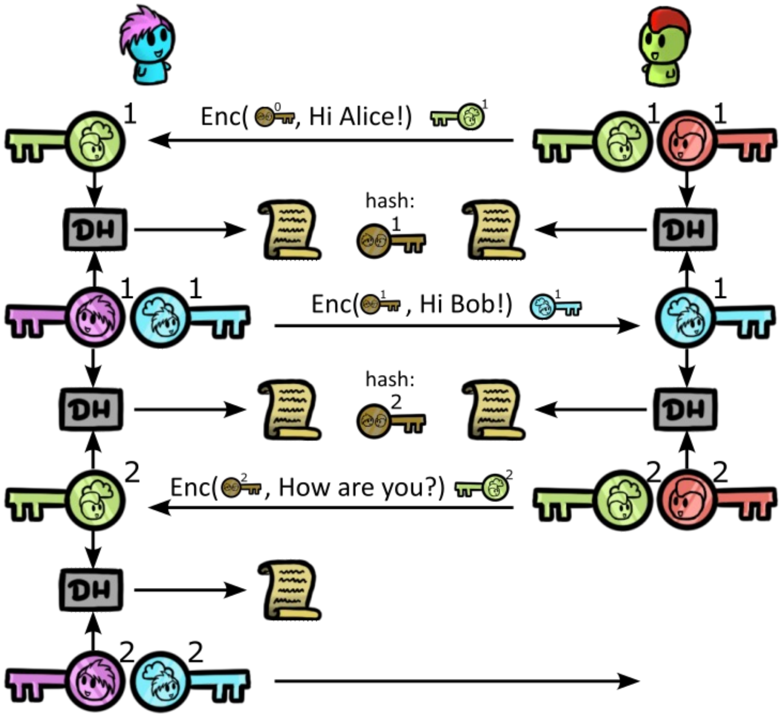
She can decrypt "Hi Bob!" and "How are you?"

OTR’s DH Ratchet (incorrect)



- 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!
- Therefore, forward secrecy is only partially provided.
- Note that we have repudiation!

Deniable Authentication in OTR



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

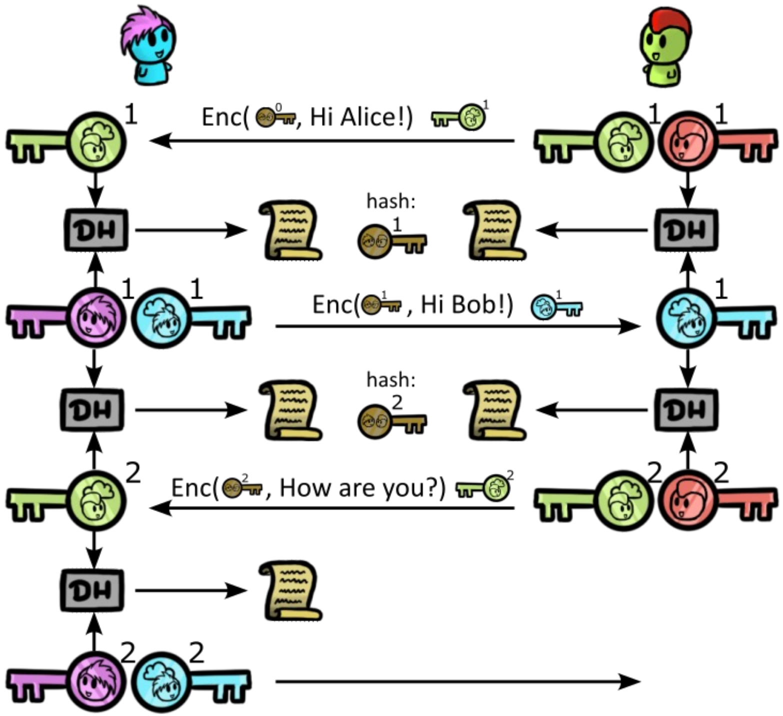
A: With a MAC!

- According to OTR's design, the MAC key is a hash of the encryption key

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

OTR's DH Ratchet (incorrect)

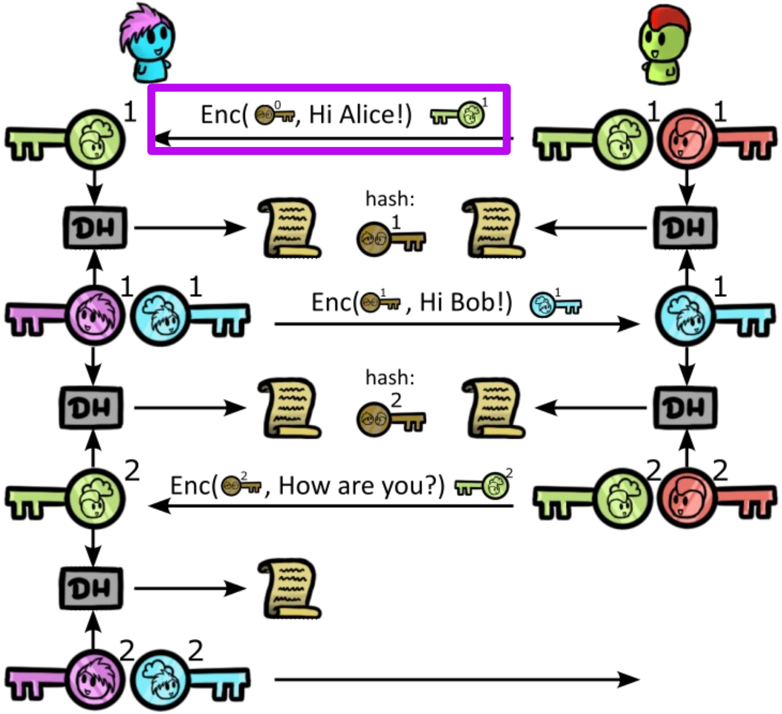


Remember, we assume Alice and Bob have 

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

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

OTR's DH Ratchet (incorrect)

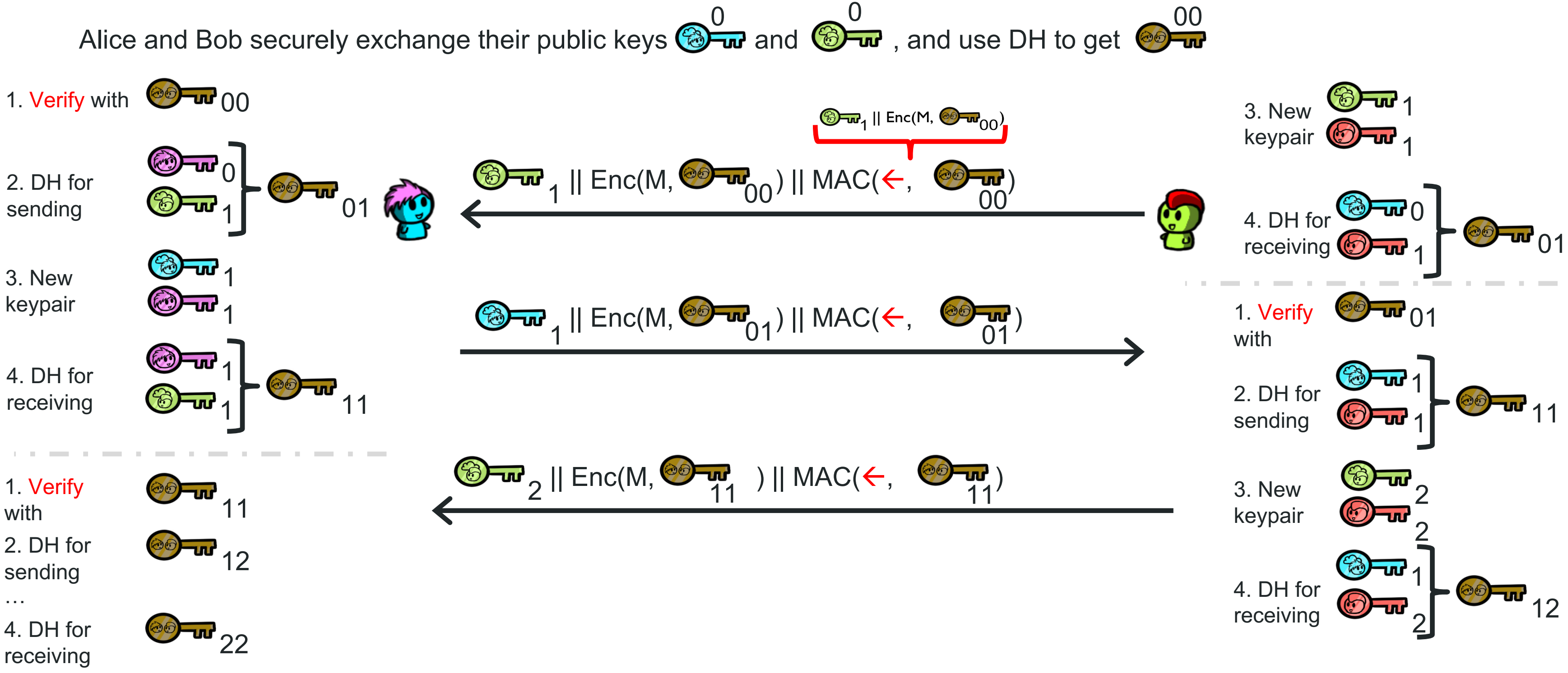


Remember, we assume Alice and Bob have 

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

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

What OTR actually does (from the OTR paper)



OTR: concluding remarks

- Using forward secrecy, post-compromise security, and repudiation (deniable authentication), we can make our online conversations more like face-to-face and "off-the-record" conversations.

- But there is a wrinkle:
    - These techniques require the parties to communicate interactively.
    - This makes them unsuitable for email.
    - But they are still great for instant messaging!

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.

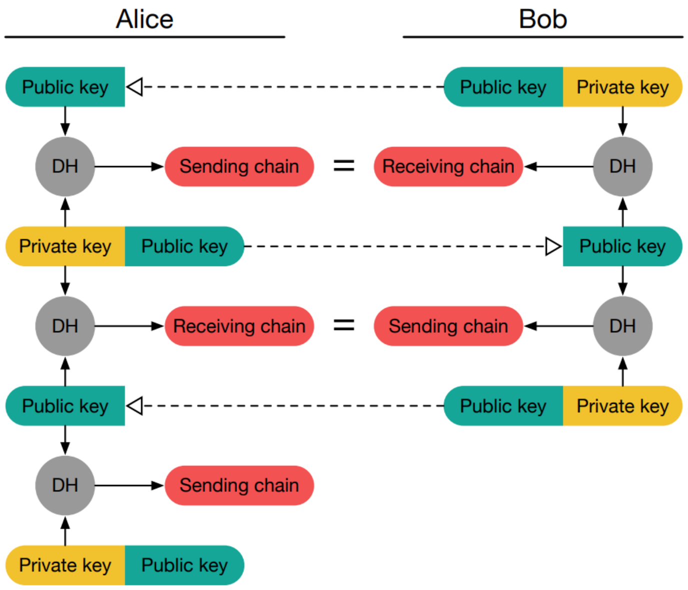
Signal

- Provides forward secrecy
  - Similar to OTR, it uses a “ratchet” technique to constantly rotate session keys.
- Provides post-compromise security
  - A leak of past or long-term keys will be healed by introducing new DH ratchet keys.
- Provides improved deniability
  - It uses a “Triple Diffie-Hellman” deniable authenticated key exchange.
- Supports out-of-order message delivery
  - Users can store per-message keys until late messages arrive.
- Uses a double ratchet (asymmetric and symmetric ratchets) that:
  - Generate ephemeral per-message keys.
  - Tolerates message loss and re-ordering.

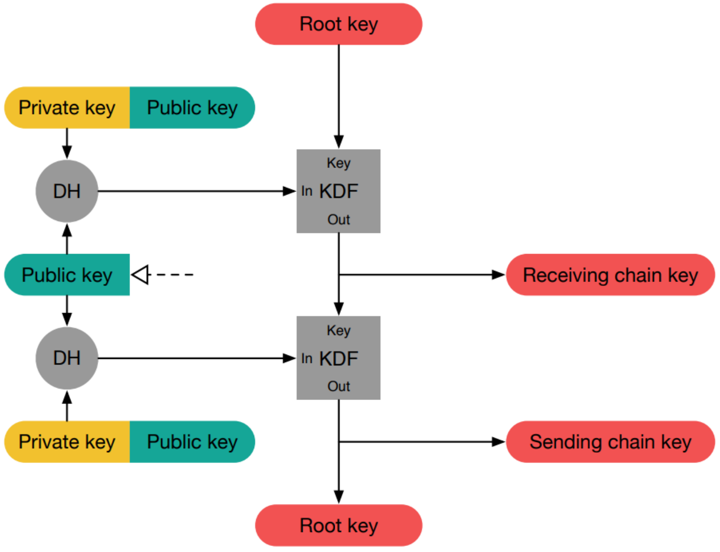
The double ratchet

Signal

DH ratchet
(asymmetric, like in OTR)



Double ratchet
DH ratchet + symmetric-key ratchet (KDF)

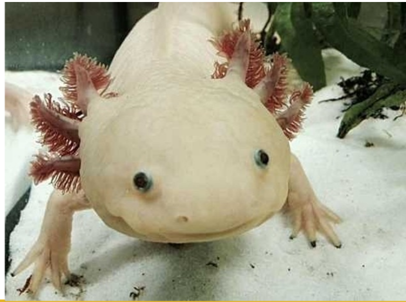


The double ratchet

- Originally called **Axolotl ratchet** for its “self-healing” property (from the DH ratchet)

- It is very well explained on the Signal website.

Photo: th1098

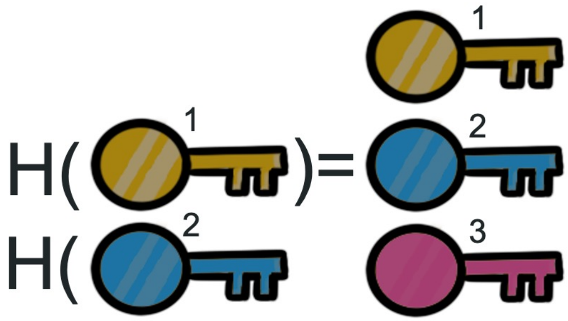


“Axolotl” is a Nahuatl word. (pronunciation)
"ah-sho-lotch"

Rationale for the KDF Ratchet

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

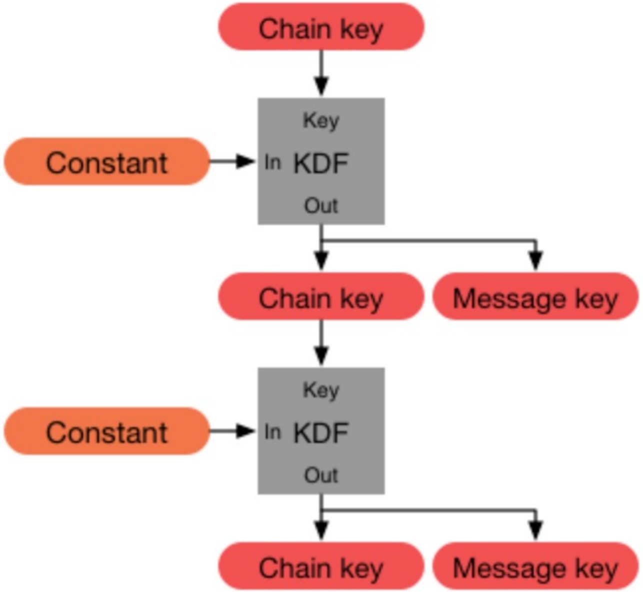
- We can do this deterministically
  - Simplified ratchet: $K_{n+1} = H(K_n)$



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

KDF Ratchet

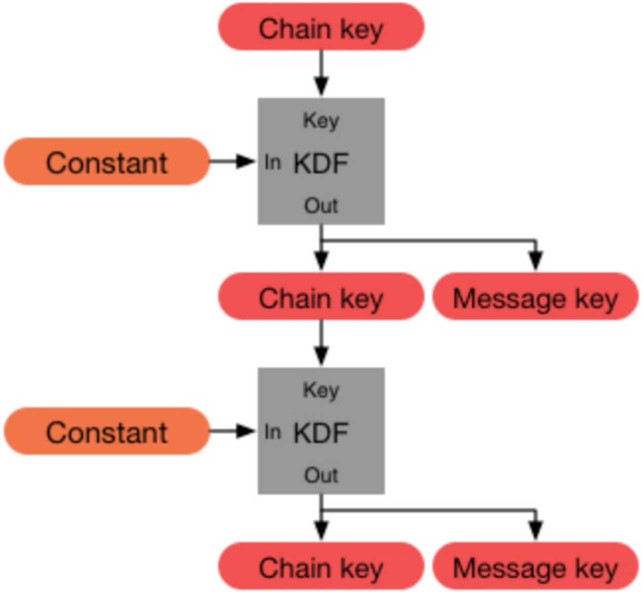
- KDF = Key Derivation Function
  - (think hashing – it only goes one way)
- Outputs **message key**
  - **Used to encrypt a single message**
- Outputs **chain key**
  - **Used to derive future keys**
- Why separate chain & message keys?
  - What if messages are out-of-order?



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 Ratchet

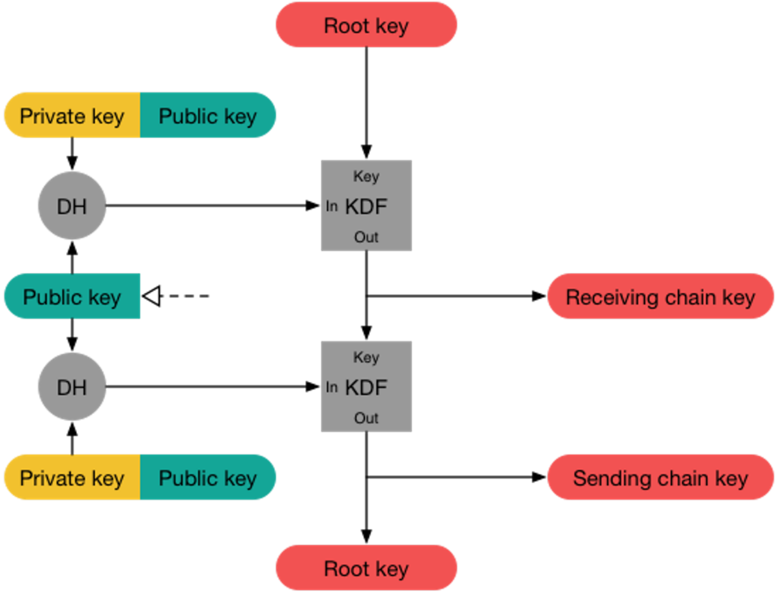
- KDF = Key Derivation Function
  - (think hashing – it only goes one way)
- Outputs **message key**
  - **Used to encrypt a single message**
- Outputs **chain key**
  - **Used to derive future keys**
- Why separate chain & message keys?
  - What if messages are out-of-order?



DH Ratchet

- Just like OTR. But now also:

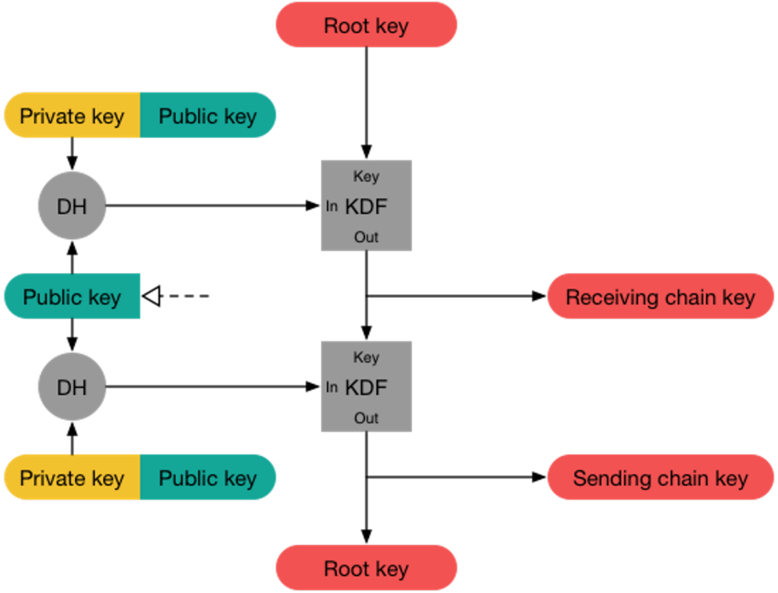
- Output is used for generating receiving chain and sending chain keys
  - These are used as input for the KDF ratchet



DH Ratchet

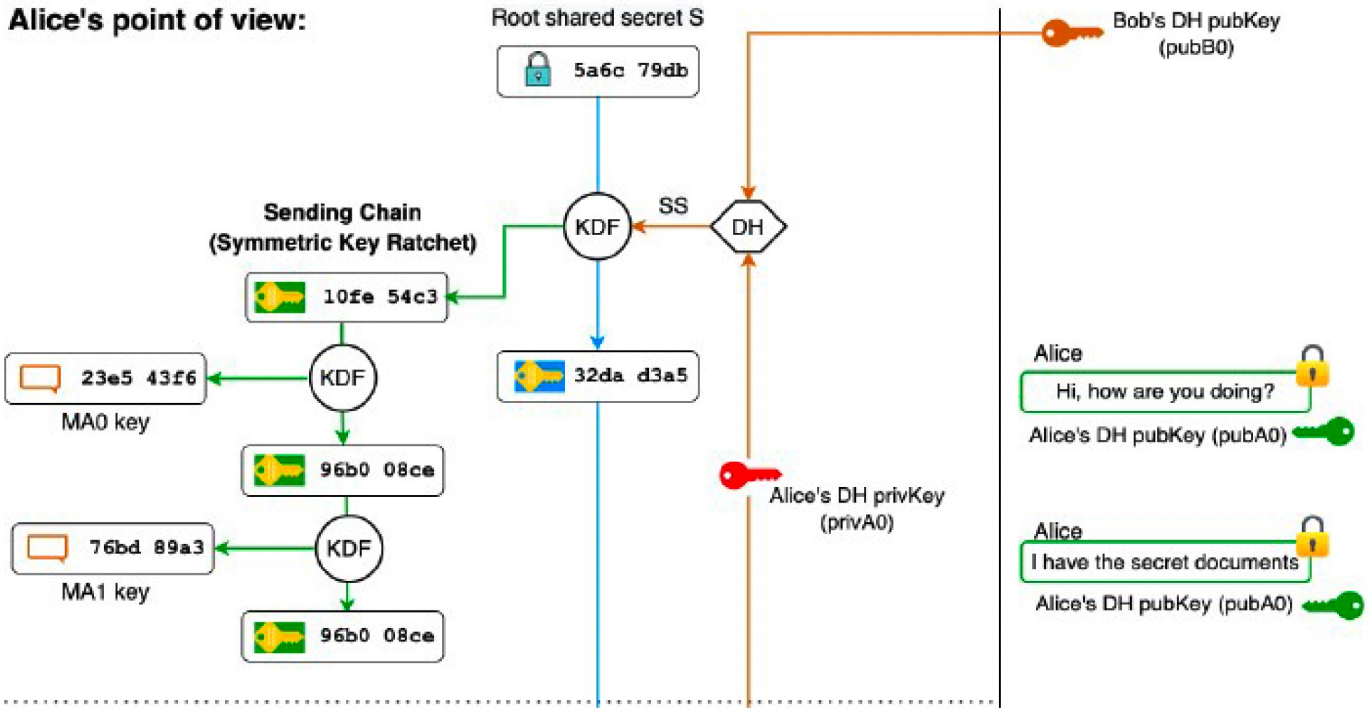
Note: There are two KDFs, and the OTR ratchet's receiving and sending chain keys are inputs to the KDF ratchet.

- Just like OTR. But now also:
- Output is used for generating receiving chain and sending chain keys
  - These are used as input for the KDF ratchet



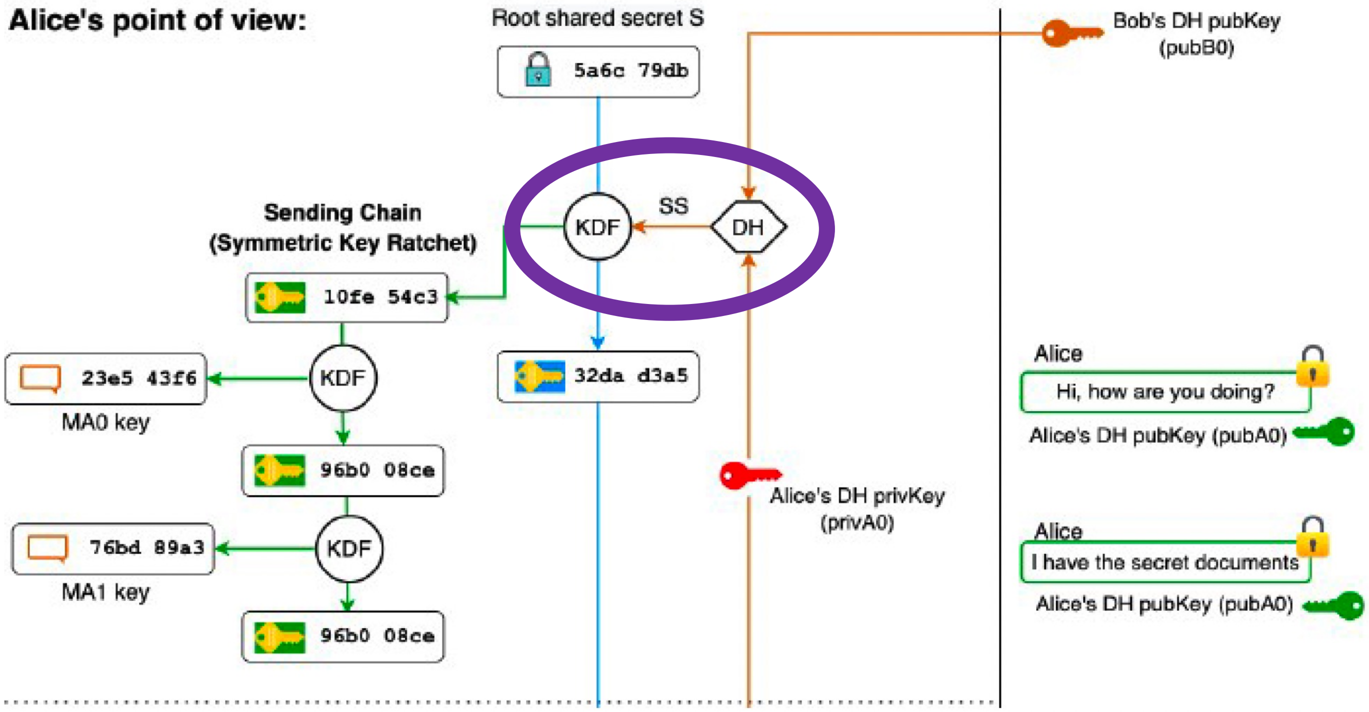
Double Ratchet Algorithm

- 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



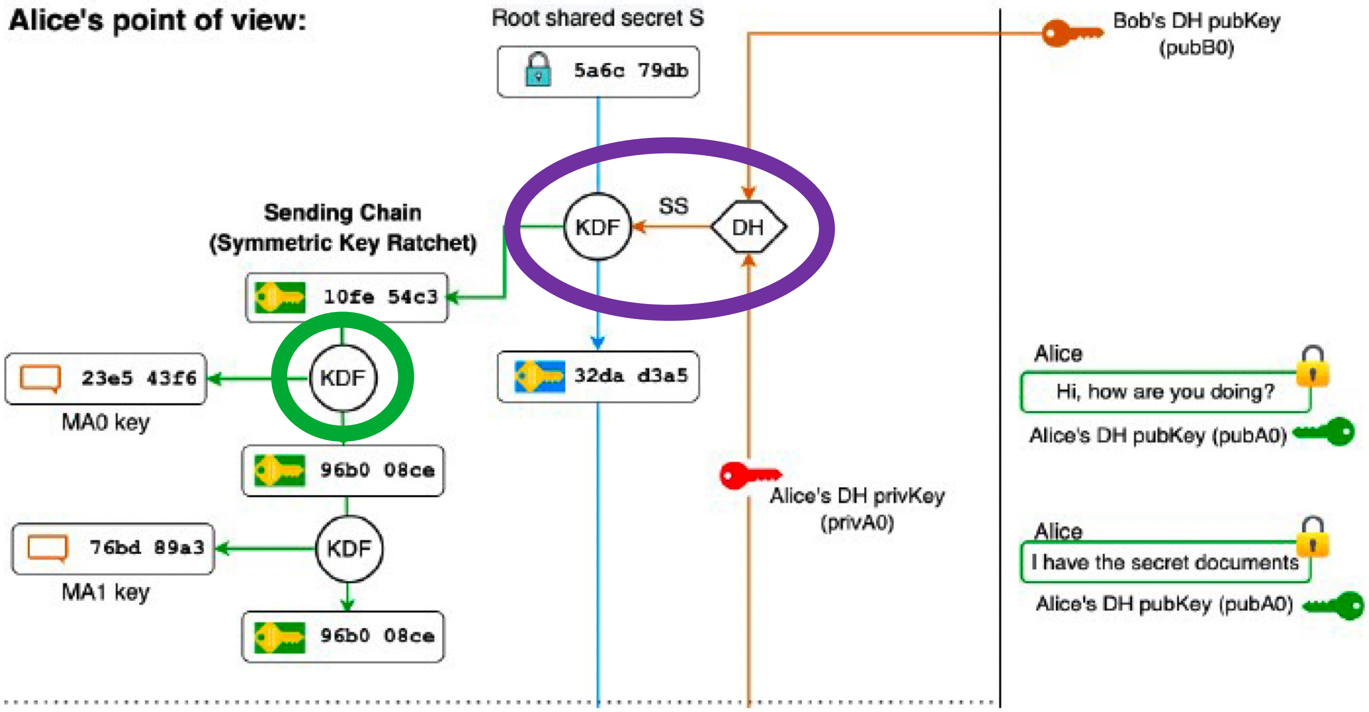
Double Ratchet Algorithm

- 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



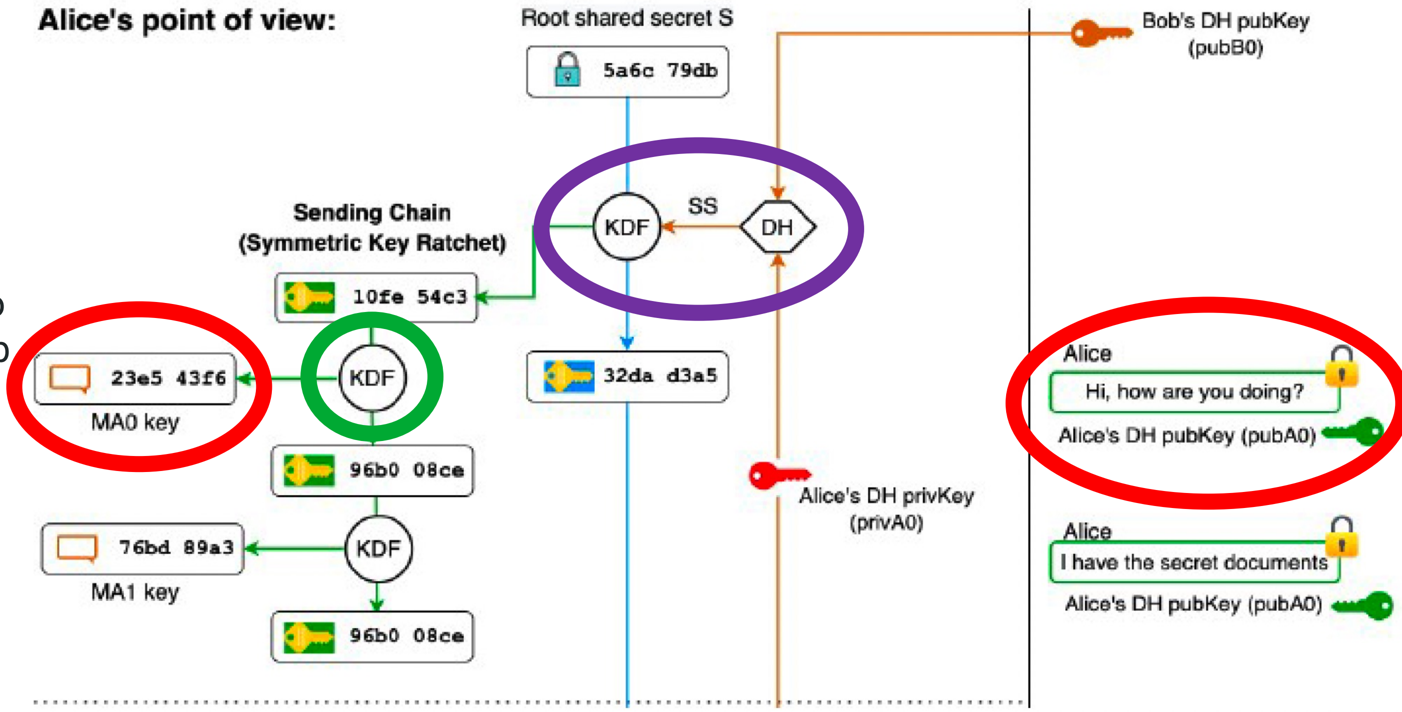
Double Ratchet Algorithm

- 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



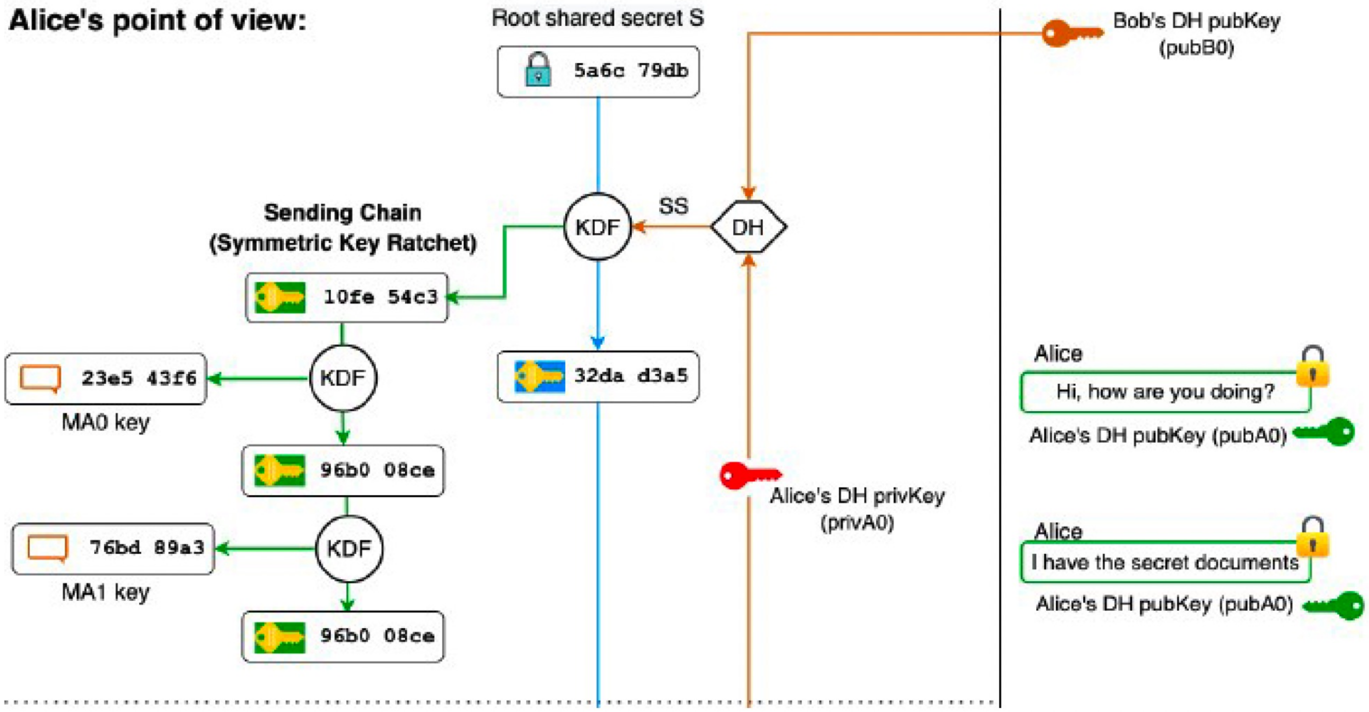
Double Ratchet Algorithm

- 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



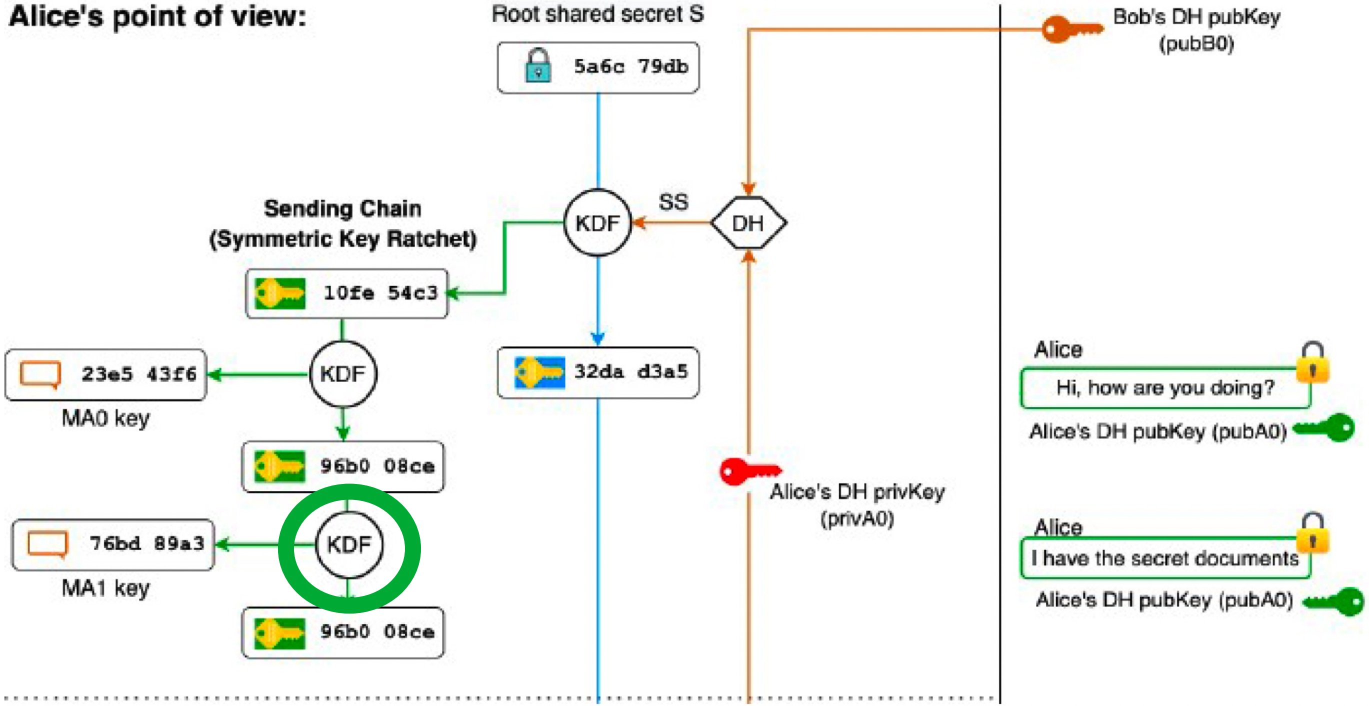
Double Ratchet Algorithm

- 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



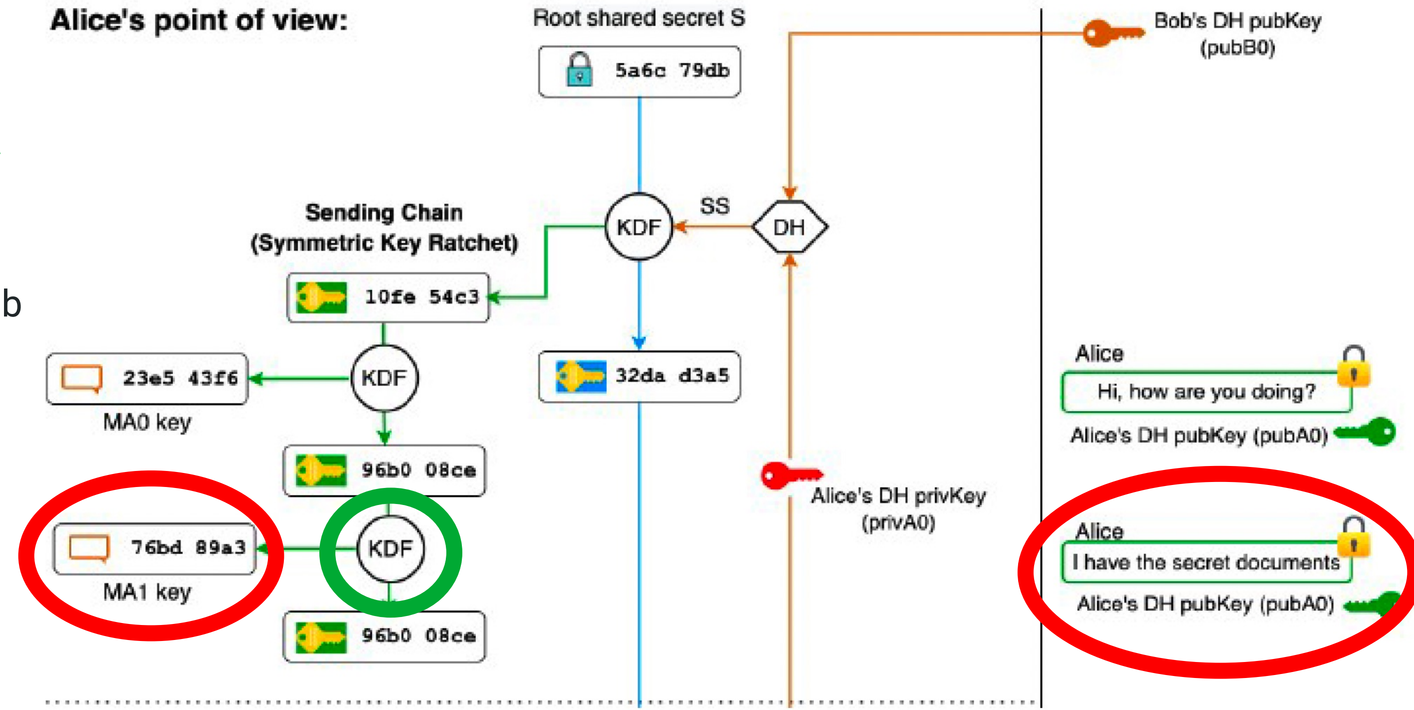
Double Ratchet Algorithm

- 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



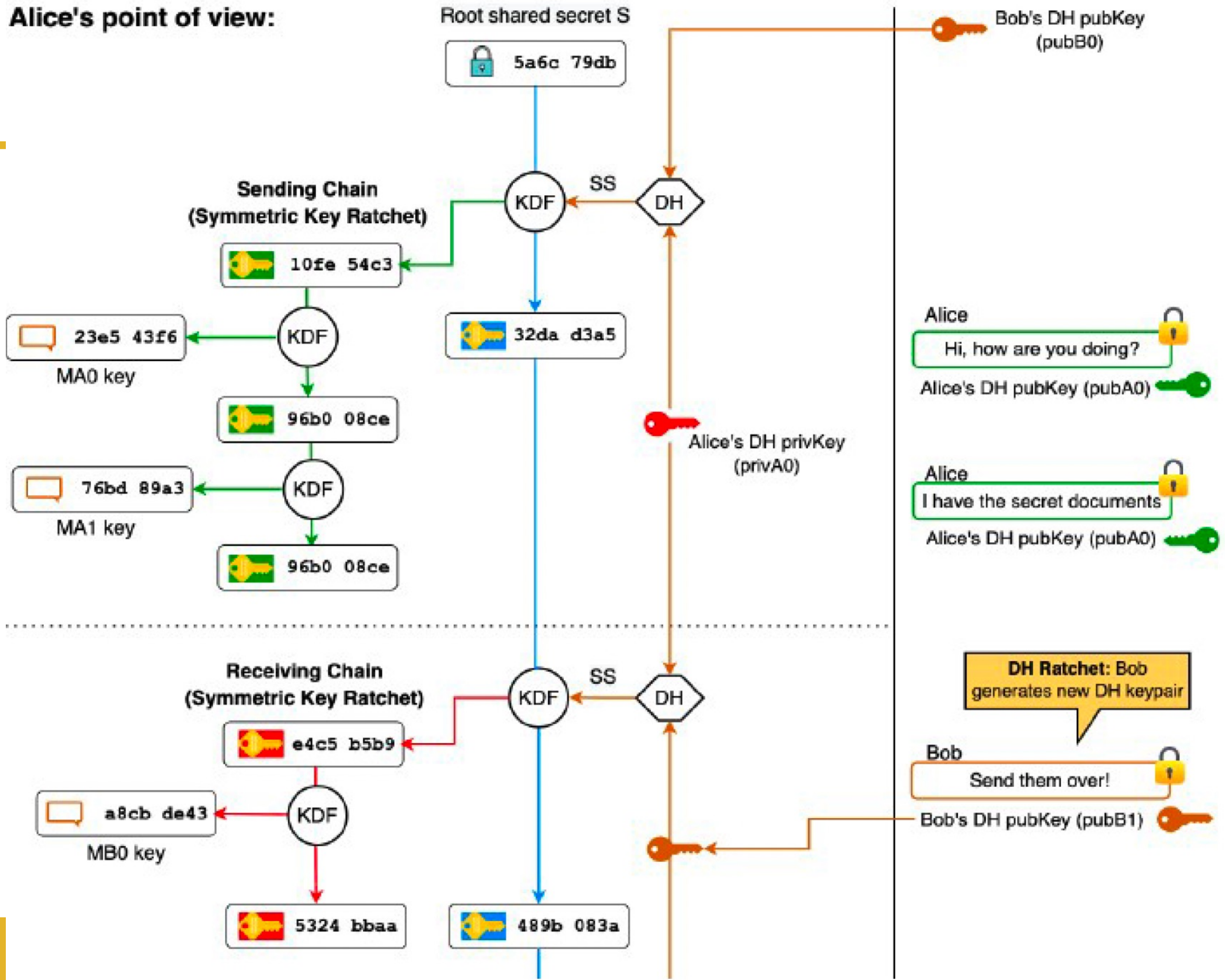
Double Ratchet Algorithm

- 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



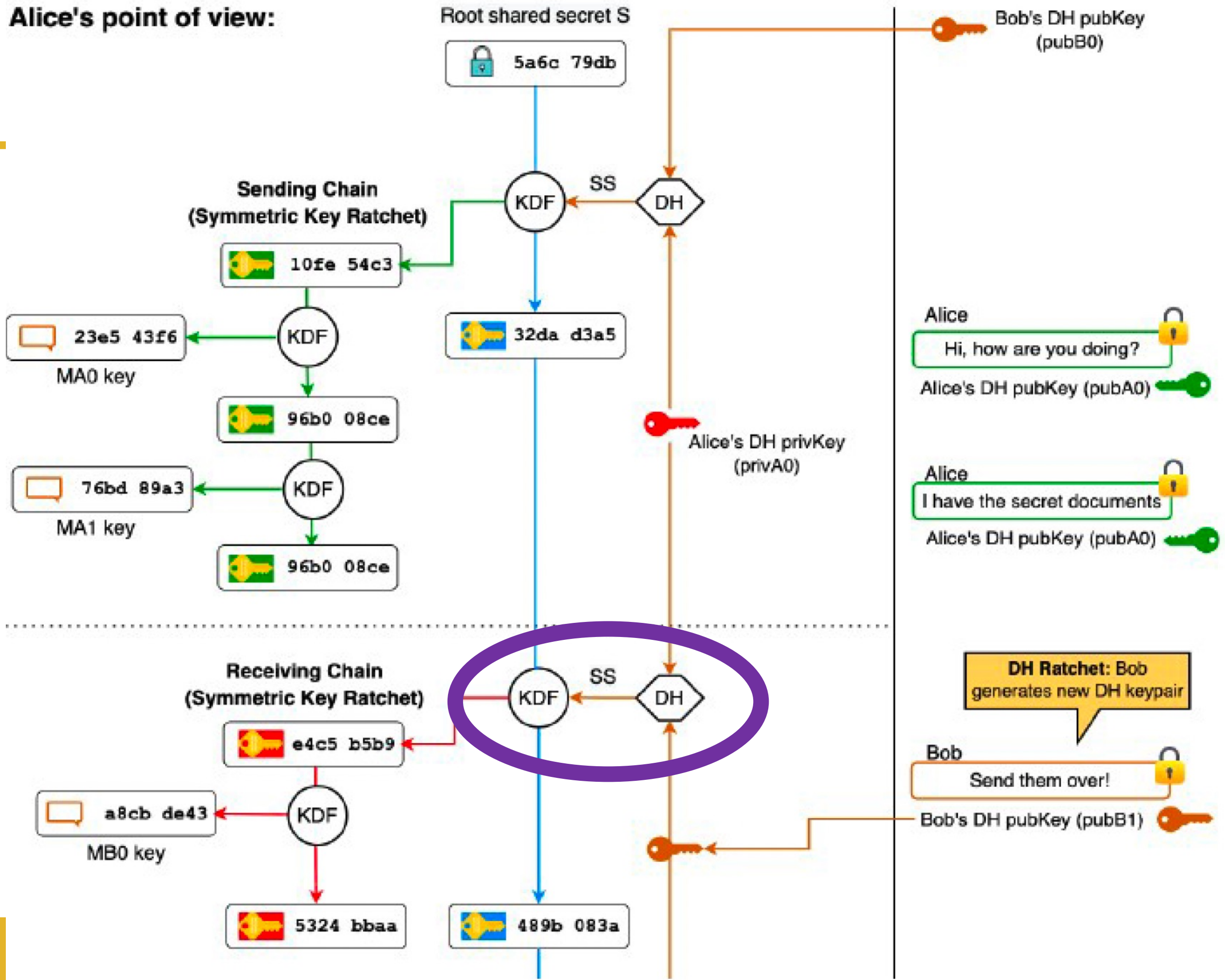
Double Ratchet

- 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



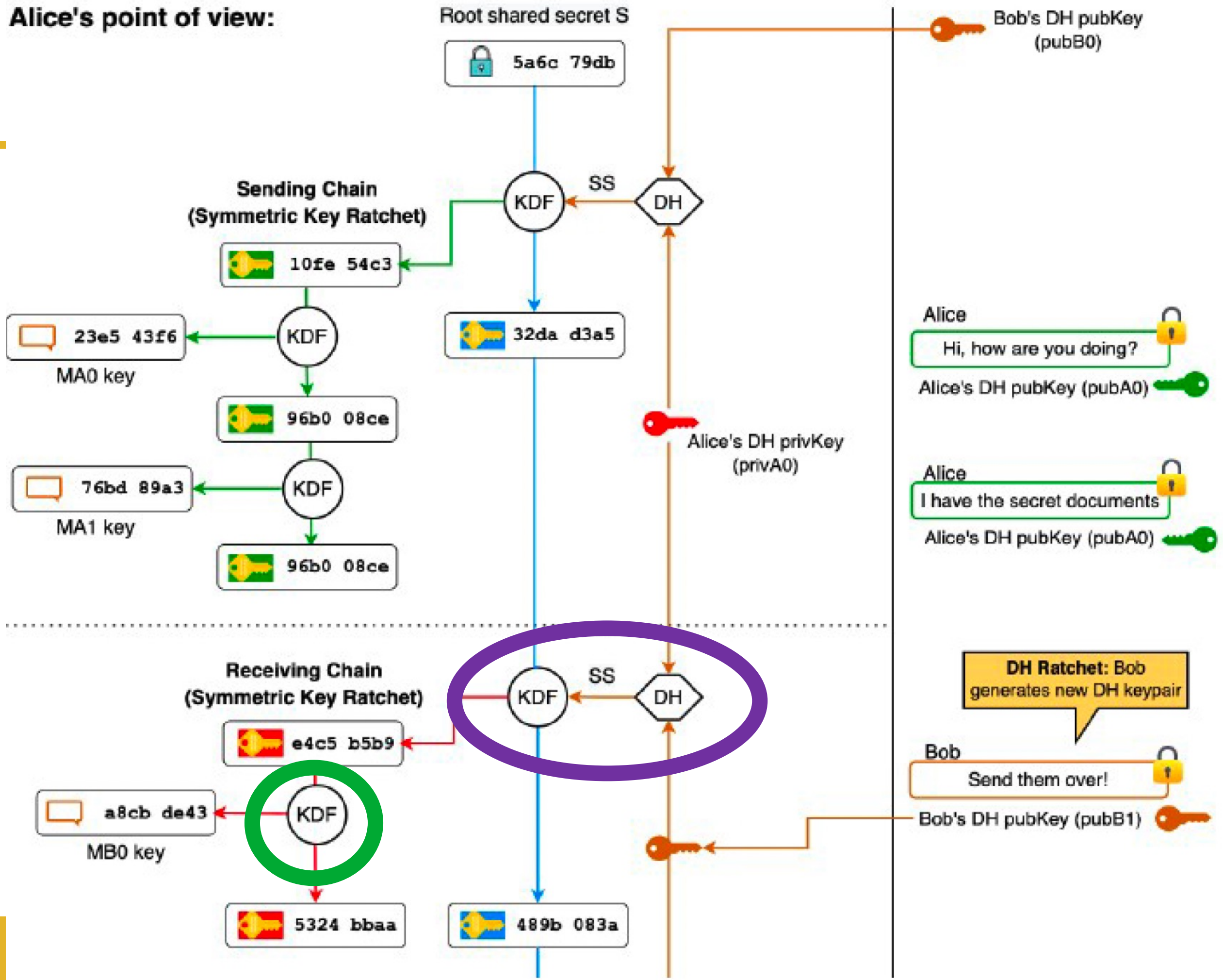
Double Ratchet

- 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



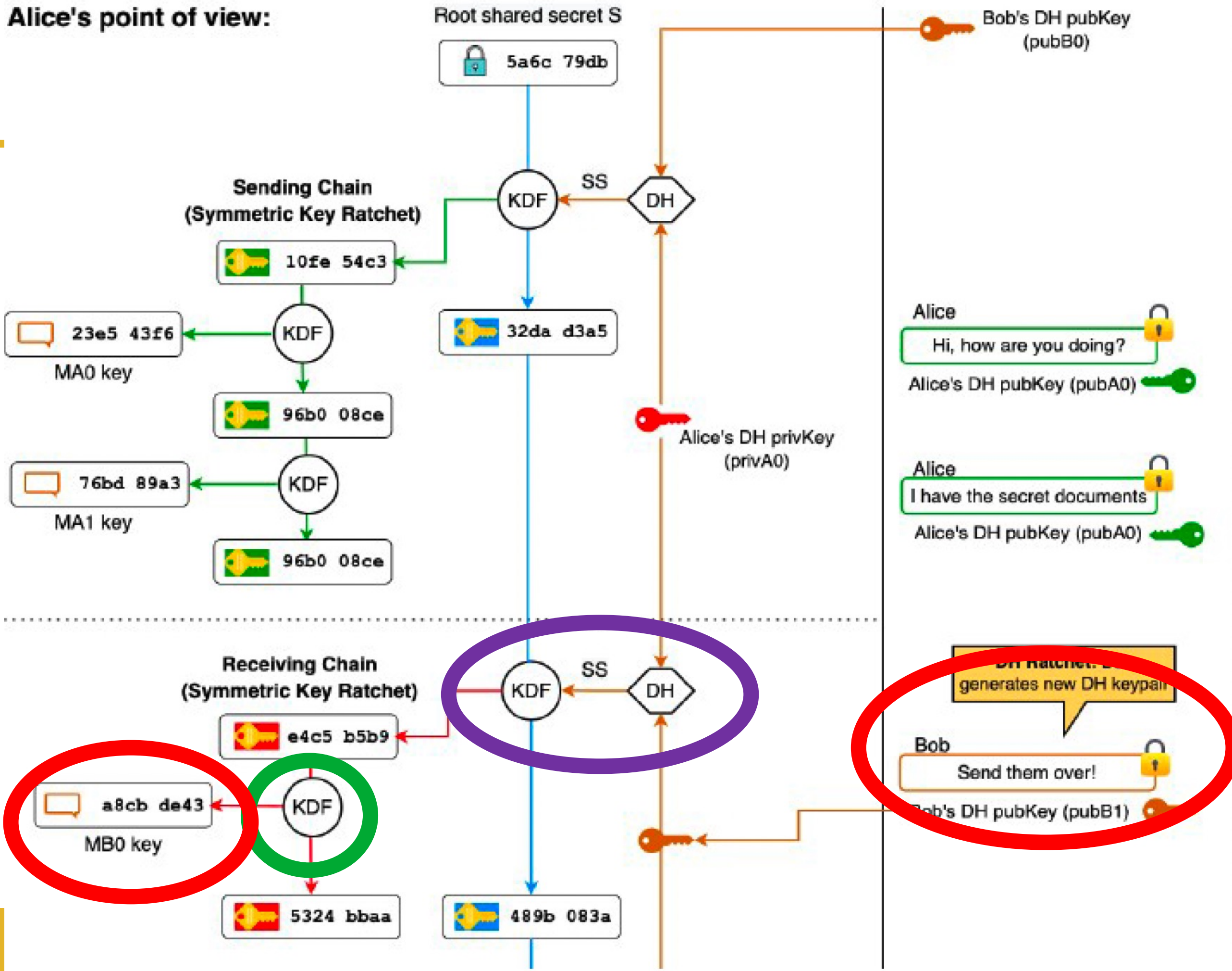
Double Ratchet

- 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



Quick Recap

- **PGP**
  - No forward secrecy
  - Non-repudiable (not off-the-record)
- **OTR**
  - Forward secrecy *and* post-compromise security through DH ratchet ☺
  - Deniable ☺
- **Signal**
  - Forward secrecy *and* post-compromise security through DH ratchet ☺
  - KDF ratchet provides only forward secrecy, but for *every message* ☺
  - Deniable ☺