

# 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?

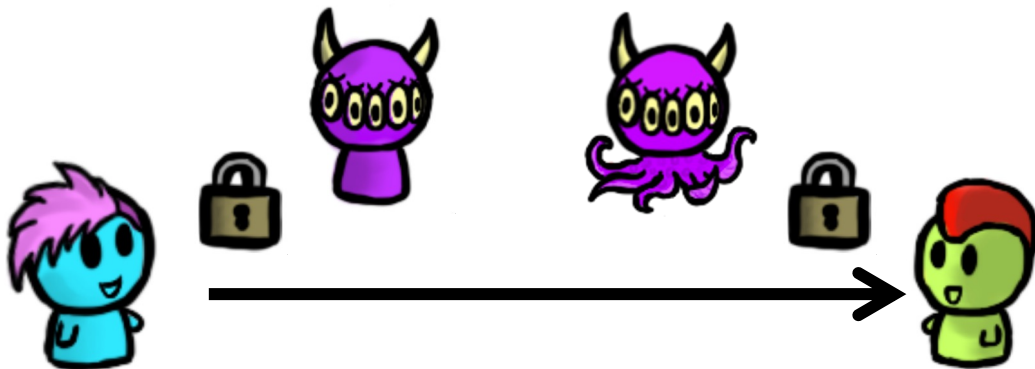


# Secure Messaging Goals

---

- **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?

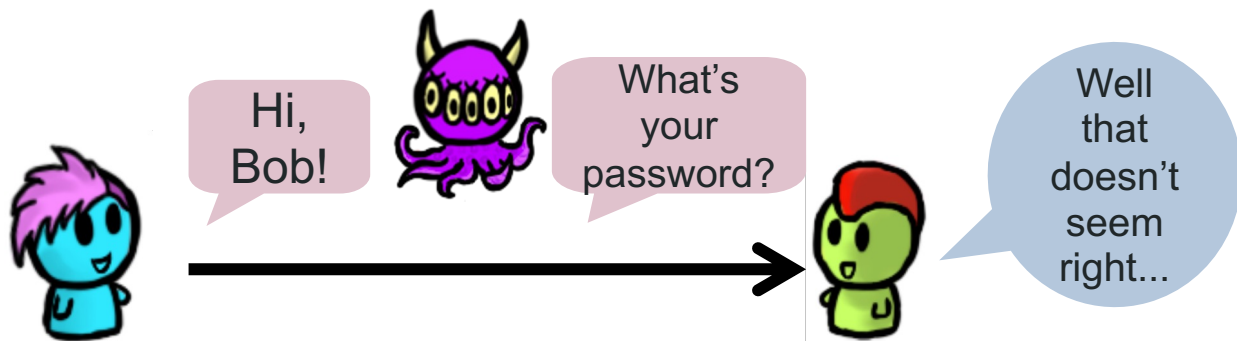


# Secure Messaging Goals

---

- **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

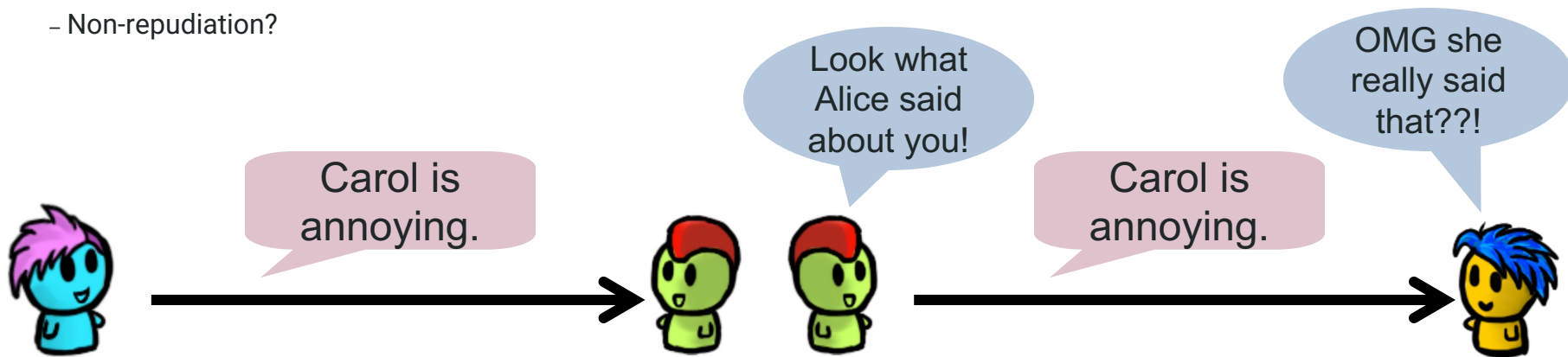
– Non-repudiation?



# 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

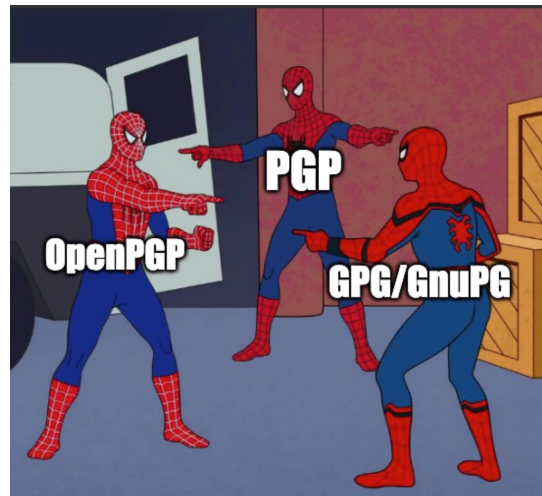
---

- 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!



Msg

1.  $\text{Sign}(\text{hash}(\text{Msg})) = \text{sig}$



2.  $\text{Enc}(\text{sig}, \text{Msg}) = \text{C1}$



3.  $\text{Enc}(\text{brown key}) = \text{C2}$



C2      C1

6.  $\text{Ver}(\text{sig}, \text{Msg}, \text{blue key}) = \checkmark$

5.  $\text{Dec}(\text{C1}) = \text{sig}, \text{Msg}$



4.  $\text{Dec}(\text{C2}) = \text{brown key}$




# How safe is all this?




Msg


1.  $\text{Sign}(\text{hash}(\text{Msg})) = \text{sig}$



2.  $\text{Enc}(\text{sig}, \text{Msg}) = \text{C1}$

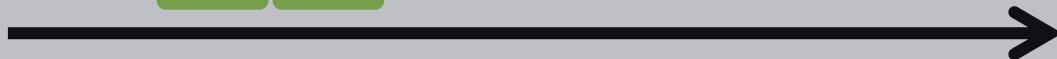


3.  $\text{Enc}(\text{brown key}) = \text{C2}$



C2

C1




Perhaps I can re-encrypt Alice's signed message and send it to Carol...


... Carol would take it as coming from Alice!



6.  $\text{Ver}(\text{sig}, \text{Msg}, \text{blue key}) = \checkmark$



5.  $\text{Dec}(\text{C1}) = \text{sig}, \text{Msg}$



4.  $\text{Dec}(\text{C2}) = \text{pink key}$



# How safe is all this?



Msg

1.  $\text{Sign}(\text{hash}(\text{Msg})) = \text{sig}$



2.  $\text{Enc}(\text{sig}, \text{Msg}) = \text{C1}$



3.  $\text{Enc}(\text{key}) = \text{C2}$



C2

C1

How can Alice prevent this?  
(think about it...)

Perhaps I can re-encrypt Alice's signed message and send it to Carol...

... Carol would take it as coming from Alice!

6.  $\text{Ver}(\text{sig}, \text{Msg}, \text{key}) = \checkmark$

5.  $\text{Dec}(\text{C1}) = \text{sig}, \text{Msg}$

4.  $\text{Dec}(\text{C2}) = \text{key}$



# 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-----

```
mQINBGPUBx4BEADa3jSmGX9GKriACgl1vokxOc8ltbHS7aYyMzu5UzgCxYy29n
7YDGDWlN23byl8Gf36HJ6mQzUjB7T54ed8pE1rtMWL+70oMNRNaFvxoST5
3pFn+GRYSaviGPKut8YdyrkaLixshjakYehmwwVvCVMBBgfrP3pR93dKWbET2EN
RMDSVBO6AzPrnjedZmGpUqp8UPeP8JotCn0xAv4ugIM6VE6xb/Cj15l/SPslhx
76LPqsPuWzRkQ9stP8YjTX+OI9+GNqLhtdmY5yXPDP9/NO+fhQVwUZOaj544a
KeFDQ/G9GKfJzTlzhQn9BdkZpf5Kjzun0+4HNk0msB5S8BdtPpuc3qs+rK6LW
aAnXUS9j7mB3Gf58fju+1gMP5dXG16nduB/W3SuH2/QSymjSm6PkuNcSMIOXEN
FCUH/aorJZQV/Xi5laQHg+cbETLRACdkaAHNNjxGDxkzbuYzjt3hPMNIBF897
PvihCO2w4pXBQ7rpxzn6OvU11awfrmdQZA2tRZOSN2Cpti3KJ00zKzfGT0VFRaVq
NfEy26ZtEPAZhgBJDo8SLUkshrMLhNnlobR/BLng1v/xSrjPTAVE/sK032GfqZ
uynR6zO+VcwAkZ3g/aK5kknPG/Or4KdEhsmOKuPgAT5duGo96t299dRqQARAQAB
tBtBGGJZSA8YyWxy2VZAZKhnbX8Z5Sj2b2H+QJXBBMBcABBFIE7YJufCQ7em
fk+0HNjVA8FMDQJfAmPUBx4CwMfCQAnjQAFcwkliBwClGfGQJcAsCBYCAwEC
HgcCF4AAc8qQHNNWA8FMDQV3LQ/8CnyOARm+seUp4ShUo5xqlEMP6GF+VbBE45G
XGIEr/PeMbdTJtkr00Qzsk0/tVYKJGILES9W/1Ta9zAkmsnyvHwFop3XZQGeqlt
U9mPpBQzAbfzW21++3CK48WcCb5mRh+O9Z7jwF0aEYDOKxO2og6a9132kUp66n
CctBy+h6ucBVMmTZS0Jfr5YHFZKa/lyQ6ODgkv+flwfPZm2N93jHejldrKSvtzi
Yb5tiXgQdwojSlxhVA6pX03CtENKqrpDP50tM70AdmVSmjQn7AR3UtBjn4JlMb
ic+/yKD2JIGLS1R5RKvov1BBQHU7fATcrFL4SORQ5o5iaEteMfLLbMmors23
oNus/wmeWkUOG76uvjQnuAr/Bc7DF4lhY/WpZGdAlayA9v9TVMUmxzDjMwmfkeJ+
OcljwJOBO6GbMBBNIr76ae+zWpleqZrjv757H+hb0i08nPBKrtXbGLM7wg/r9ii
qEm4pHT5Poi6WBR3Pyu/PoyEnPlKonxSv9kOIXGyJdcdV6jBA6c37mFF5Ofk8A
s/x3V85+0YK34RbDVdqm5+v42Lo5DP49KdBV1dp+O07nWRJDSoroFarbMcPCCWJ
i0P4+r9nU9Hx8k6mjstyzBzplmDhBnCo5hAaAytuOLU3wKwmqh8ONChKYRxo
+88+0P65Ag0EY9QHhgEQAOFF4x8GK5CjK5JUL87s0nm9QOGtpx8L4dm9Rftu
u6cP7XcJOngxf4HufcL6wNfPMF5knU6ezXUgMvOseFVT30VC6uf39OrqQJ26va/
LcYzKaiWFLKYuBvtLDuPudANhplQhH7s4FQlVTPUO+saCAqJdJtOsaQ/F/n+Gtzz
DxDnPbsTC5oEskgfhyednT9gZpCscx9Gd3mDyDDkMGyWaE4bWjdjX2NEj6tuezY
ijyqTYBHKf9eNsmPY9SEbV9HIMLgZa/RamrtZ+AMya2IuyBXI6oo+oEI571cefD
BFajeOKH0MhtPKQvagyett6l5Ta+6Ekqoy50c90s85UduIzKCaZsA8vrkhLNh
KvJ90U5iVuo+G6wppZzQHplumX+eRMSX1U4hBash5z+le3YUCn8rDwEfM5G2
EAMRDSFGQ7L5dM56Z3PRD4a4PzP71TijtpNubF3N3uOUI/T1ChghJLfm79DI
O9MSYRdOfPVlUmqWliw862zXor8dqwnlKB9uDWVMHnGkEftseCOWrsRaeMHDfc
7A/bNCoDrA8x18GielkVTmhuFmC77WMN43rJyL71W2VOKjN0NHcYsGohC4z
0aJcdLDvkt4ArixpmhSmMOWZsvblT9l5voY8GIEbltX5xppOUGZ+3vfq0UwER
ABEBAAGJAjwGAEIAcyWlQTViuUW6pxd6tZ+T7Qc0YDwUwNBQUCY9QHglbDAUJ
ACeNAAKCRACv0YDwUwNBROIEACI8LSN8YlnRkq/9JqJy6qkoLTrOr5Yvz7Fm/F
KRP7vDiCkGH3NwsrBE3+r7UB8MWWJjOrdtWld7a5AasEwtSTXKHrpzc/s8kn1m
POTr/sAlfbf6jXAOqK0ZWhod4YsRBY7Xe9EhOup5y6UeFbGMS80HvLrApju
lUvKJndpD+21U0Hu16JKAulhyKfpxVtjH3lxnagBI9UOILGoh49aMa4RwAmY0
Z4h9S2CqHMOoKtLdovHo5BvyDla91TpennGhm+AeE11VpdRfpaa104srGMUQX
kjtnHndMVHEzMsYsvwygJEIXMBpkfZC/COohqvqMrRQgh05TATa6xVRNym1241
PqMbzN7JYM2DflbMPD2qd9IT6rKfXUzltRQshwXpcvi+8Mgsb53JYkQlpidu0
z+VoQ7ObHuwwPC1oHJ8Q3FaKlynhACVOiDR8l89rZ3mvtlMLMKKyYeiJpB/
idbN3QtUuPjAluclN4883DwzMO5ZQ8CPc3/6yOQ0YUyTUpNo143XcQJ/OWc3Tmm
YsMnvZVhY6MoiQ7cXDJwVUOTU4lG6gkwmbE07zatGHXv/agSxpRulZlhZhem
fi1144fYlI2ZxWWVr2vQ6T9eLEtYcJtEgxaot0thOxxQ3dpXavuydG84ZzYmId
i96dv==
=tJAW
-----END PGP PUBLIC KEY BLOCK-----
```

# 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-----

```
mQINBGPUBx4BEADa3jSMGX9GKriACgl1vokxOc8ltbHS7aYyMzu5UzgCxYy29n
7YDGDWn23byI8GF36hNl6mQzUjB7T54ed8pE1rtMWL+70oMNRNaFv6ostS
3pFn+GRYSaviGPKut8YdyrkaLixshjakYehmwwVvCVMBBgFrP3pR93dKWbET2EN
RMDSVBO6AzPrnjedZmGpUq8UPeP8JotCn0xAv4ugIM6VE6xb/Cj15I/SPslhx
76LPqsPuWzRkQ9stP8YjTX+OI9+GNqLhtdmY5yXPDPF/NO+hqVwUZOaj544a
KeFDQ/G9GKfJfzThnQ9BdkZpff5Kjzun0+4HNk0msB588BtdPpuc3qs+rKl6W
aAnXUS9j7mB3Gf58fjJu+1gMP5dXG16nduB/W3SuH2/XSympjSm6PkNcSMIOXEN
FCUH/aorJZQV/Xi5laQHhg+cbETLRACdkaAHNNjxGDkXzjvYzjt3hPMNIBF897
PvihCO2w4pXBQ7rpxzn6OvU1jwfrmdZQA2tRZOSN2Cpti3KJ00zKzfGT0VFRaVq
Nfey26ZtEPAZhgBJDo8SLUkshRMLhNnlobR/BLng1v/xSrjPTAVE/sK032Gfqz
uynR6zO+VcwAkZ3g/aK5kknPG/Or4KdEhsmOKuPgATsduGo96t299dRqQARQAQAB
t8lBBGjZSABYwXpy2VAZKhnbX8Z5j2b0+HjQXBMBBCABFFIE7YLiufCQ7em
fk+0HNjVW8FMDQJfAmpUBx4CwMfCQAnjQAFcwkIbWClGfGQcAAsCBYCAwEC
HgcCF4AAcGQHNNW8FMDQV3LQ/8CnyOARm+seUp4ShUoXqIEMPG6F+vB8E45G
XGIEr/PeMbdTjtKr00Qzsk0/tVYKjGILES9W/1Ta9zAkmsyvhFowp3XZQGeqlt
U9mPpBQzAfzwW21++3CK48WcCb5mRh+O9Z7jwF0aEYDOKxO2og6a9132kUp66n
CctBy+h6ucBVMmTZ0Jfr5YHFZKa/lyQ6ODgkv+flwPZm2N93jHejldrKSvtzi
Yb5tiXgQDwojSlxhVAp6X03CtENKqrpDP0StM70AdmVSmjQn7AR3UtBjnAJMb
ic+/yKD2JIGLS1R5RKvov1BBQHU7FAtrCrFL4SORQ5o5iaEteMfLLbMmors23
oNus/wmeVkkUOG76uvjQnuAr/Bc7DF4lhY/WpZGdAlayA9v9TVMUmxzDjMwmfkeJ+
OcljwJBO6G6MBBNlr76ae+zWpleZrjv757H+hb0i8n0PBKrtXbGLM7wg/r9ii
qEm4pHTSPoi6Wb3Pyu/PoyEnPlKonX5v9kOIXGjYDcdV6jBA6c37mFF5Ofk8A
s/x3V85+0YK34RbDVdqm5+v42Lo5DP49KdBV1dp+O07nWRJDSOrofArbMcPCCWJ
i0P4+r9nU9H8k6mjstyZBglpmDh8nCo5hAaAyuOLTU3wkmqh8ONChKYRxo
+88+0P65Ag0EY9QHhgEQAOFF4x8GK5CjK5JkU187s0nm9OGxtpx8L4dm9rFtu
u6cP7Xc0Jngx4HufcL6wNfPMF5knU6ezXUgMvOseFvT30VC6uf39OrqQJ26va/
LcYzKaiVFLKYuBdtLduPudANhplQhH7s4FQivTpuO+saAQjDitQsQ/F/n+Gttz
DxDnPBstCS0eSkghfyednT9gZpCsc9Gd3mDyDDkMgyWaE4bWjdjZNEj6tueZy
ijytYBHkF9eNsmPY9SEbv9HIMLgZa/RamrtZ+AMya2IuyBXi6oo+oEi571cefD
BFajeOKH0MhtPKQvkgayeti6i5Ta+6Ekqoy50c90s85UduIIZKCaZsA8vrkLnh
KvJ90U5iVuo+G6wppZzQHplumX+eRMSX1U4hBhB5z+flE3YUcn8rDwEfM5G2
EAMRDSFGQ7L5dM5GZ3PRD4a4ZpZf7TijytPNUbF3N3uOUI/T1ChghJLfm79DI
O9MSYRdOfPVilumqWliw862zXOr8dqwnlKB9uDWMMhGnEktfseCOWrbsRaeMhDFc
7A/bNcDrA8x18GielkVTMhuFmC77WIMn43rJyL17W2VOKjN0NHcYsGohC4z
0aJDDJLvdkt4vixpmhSnmMOWZsvblR7915voY8GIEbltQ5xppOUgZ+3vfgU0wER
ABEBAAAGIAjwGAEIACYWtViuUW6pxd6Z+T7Qc0YDwUwNBQUCY9QHglbDAUJ
ACeNAAAKCRAC0YDwUwNBQUCY9QHglbDAUJACeNAAAKCRAC0YDwUwNBQUCY9QHglbDAUJ
KR7vDiCoiGH3NwsrBE3+r7UB8MWWjOrdtWld7a5AasEwT5KXhprzSC/s8kn1m
POTR/Vsalfb6gXACQK0ZhWhoD4YsRBY7Xe9HeOup5YeU6rBGM580HvLrApju
IUVKJndpD+21U0Hu1.6jKAulhyKfPvXvjH3lxnagBI9UOILG0h49aMa4RwAmY0
Z4h9S2CqHm0OkelQdovHoS5ByDla91TpenngHm+AeE11VpdRfpaal1O4srGMUCXQ
kjtnHndMVHEzMsYvwygIEXMBpkfzF/CCOqhqvM+RQghO5TAt6xVRNym1241
PqMbzN7JYIM20fBMMPD2qd9IT6rKfXUzLtrQswhXpcvI+8Mgsb53jYQJpigiidu0
z+VQ70bHuwWwC1oh8Q3SfAKlyfhACV0iDR8l89r13mVt1MLkMKYKijpB/
idbN3QtUuPYAln4883DwzMO5ZQ8CP3/6yOQOuyTUpNo143XcQJ/OWc3Tmm
YsMnvZVhY6MoiQ7cXDJvwRUTU4lIG6gkwmbE07zatGHXv/agSxpRulzhZHem
fi1144FYI2ZxWWVr2vQ6T9eElTyCjTEGaot0thOxxQ3dpXavYdG84ZzYmId
i96dvg==
=tJAW
-----END PGP PUBLIC KEY BLOCK-----
```

# Verifying Fingerprints

---

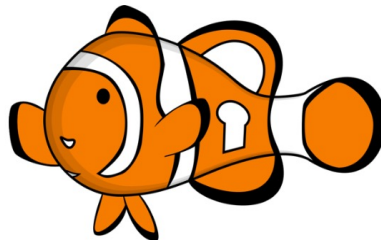
- Hash the key to get the key **fingerprint**, and compare key fingerprints instead!
- Much **shorter strings** to compare:
  - EF22 E516 EA9C 43B7 A67E 4FB4 1CD2 5603 C14C 0D05
- With a good hash function, no two key fingerprints should collide
  - Q: What if you only use part of the fingerprint? 

# Schemes for Manual Fingerprint Verification

- QR Codes & Safety Numbers



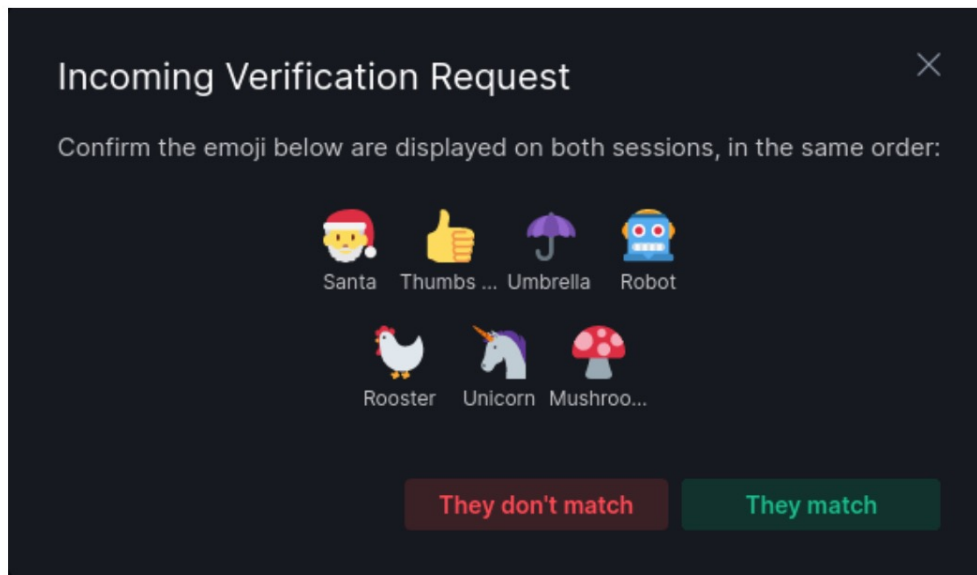
Alice and Bob's  
safety numbers  
are combined in  
a single number



# Schemes for Manual Fingerprint Verification

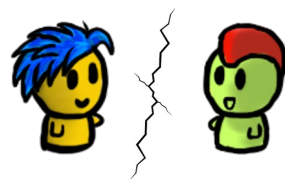
- Emoji

[**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

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<sup>1</sup>  
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  
shengxi@cmu.edu

Levi Broderick  
Electrical and Computer Engineering  
Carnegie Mellon University  
ljb@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

One month after we updated the current usability situation of PGP 5.0 in comparison to designed a pilot study to find the following areas: create a key pair, get an email, sign an email, signature, and save a backup of

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 show 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

In our study of 20 participants, groups of participants who attempted to exchange one pair was able to successfully complete the assigned task in the one hour. This demonstrates that encrypting email in Mailvelope, is still unusable.

Our results also shed light on several tools could be improved. First, it would be helpful in assisting first time users should be doing at any given point in time. A more complete description of public key infrastructure is needed to help users correctly manage their own keys.

## 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.*”



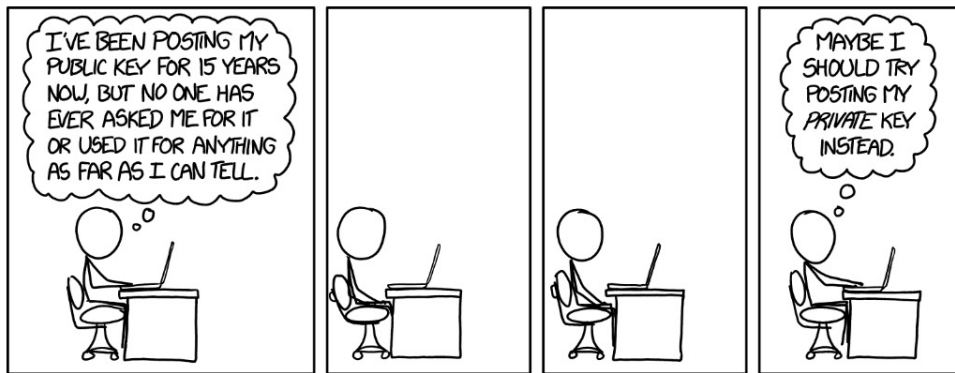
<https://xkcd.com/1181/>

# 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

---

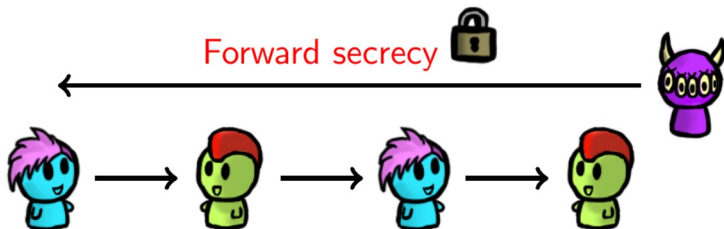
- Messaging (XMPP) extension for encryption with:
  - Forward secrecy
  - Post-compromise security
  - Deniability

Let's see  
what these  
are...



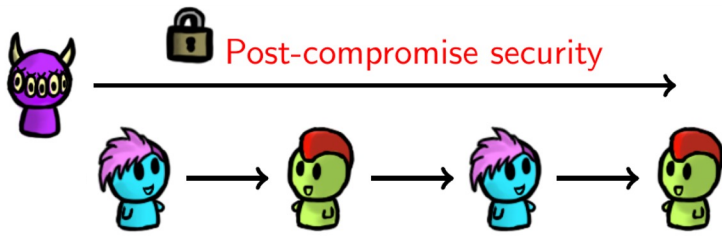
# Goals of Off-The-Record Messaging

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



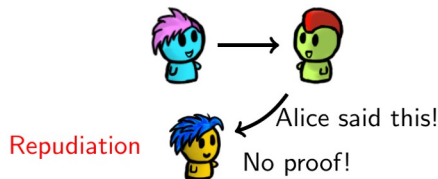
# Goals of Off-The-Record Messaging

- **(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



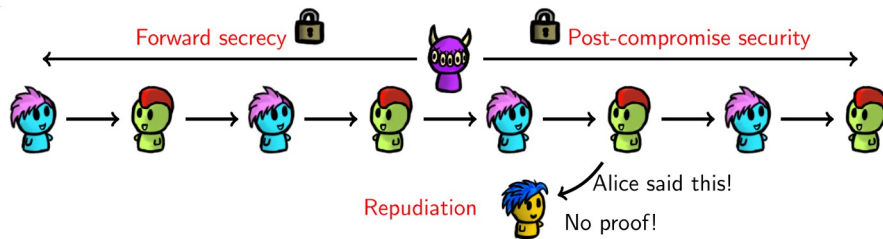
# Goals of Off-The-Record Messaging

- **(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

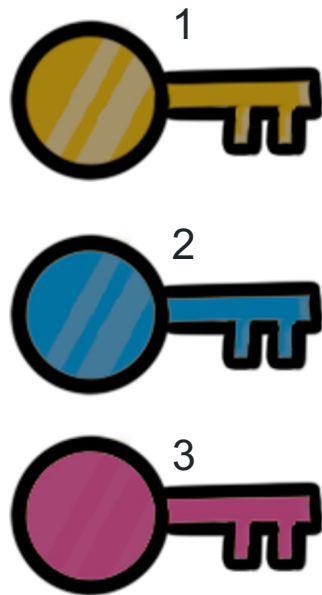
---

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

Q: How can we accomplish that?

Change the key!

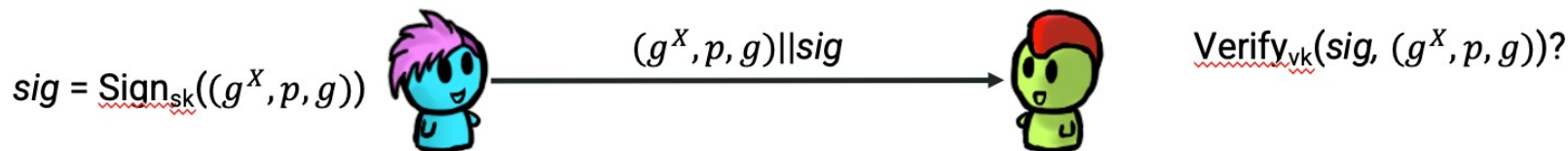
Old keys must be securely deleted





# Forward Secrecy (one approach)

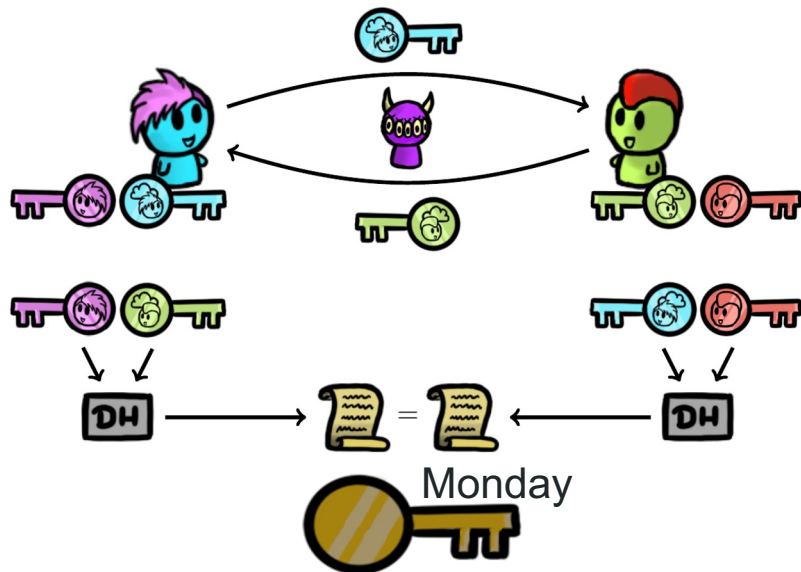
- Recall Authenticated Diffie-Hellman...



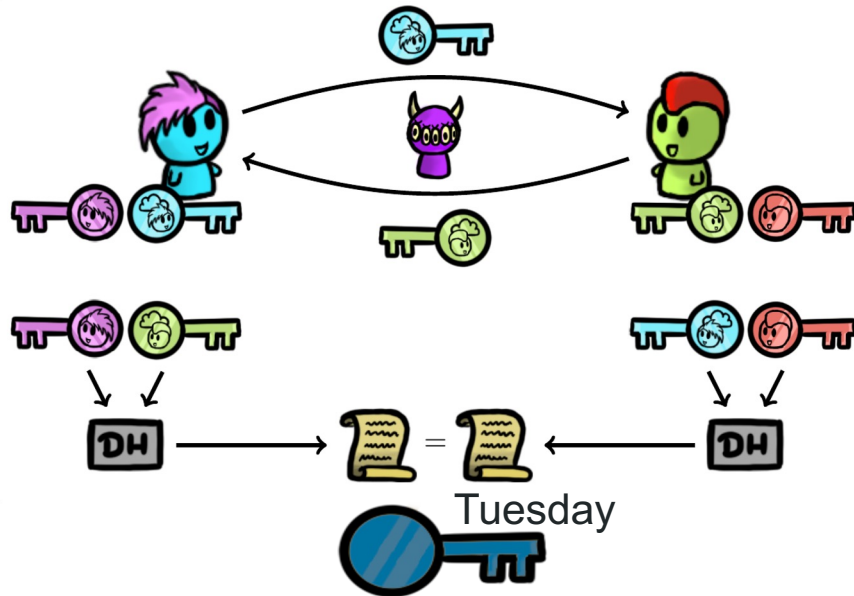
- Alice and Bob find a **shared secret** used to create a symmetric key
- DH keys can be used for **ephemeral** (temporary) communication “sessions”
  - Alice and Bob can always make new keys later
  - Call these “session keys”

# Forward Secrecy (one approach)

- Alice and Bob talk on Monday...



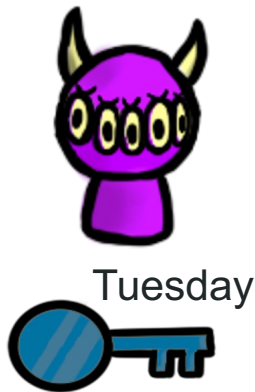
- Alice and Bob talk on Tuesday...



# 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...)




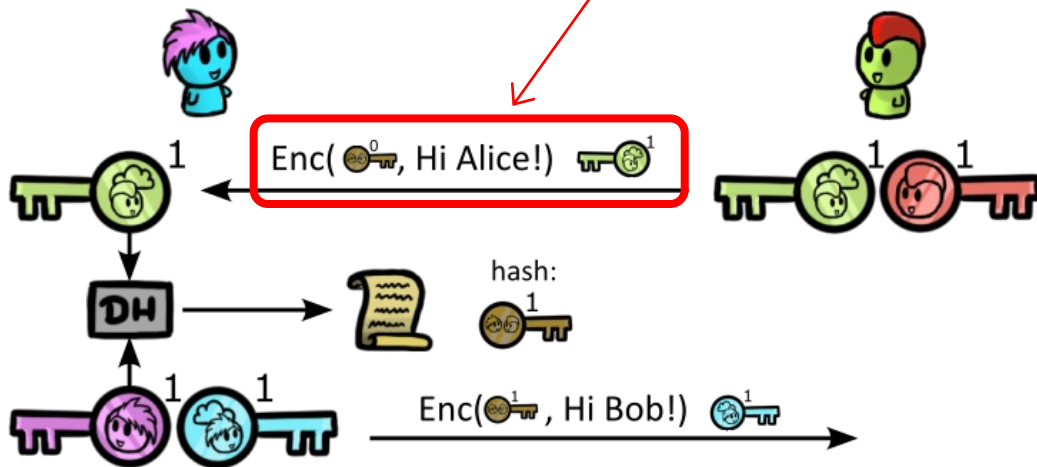
# Forward Secrecy in OTR

---


- **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?

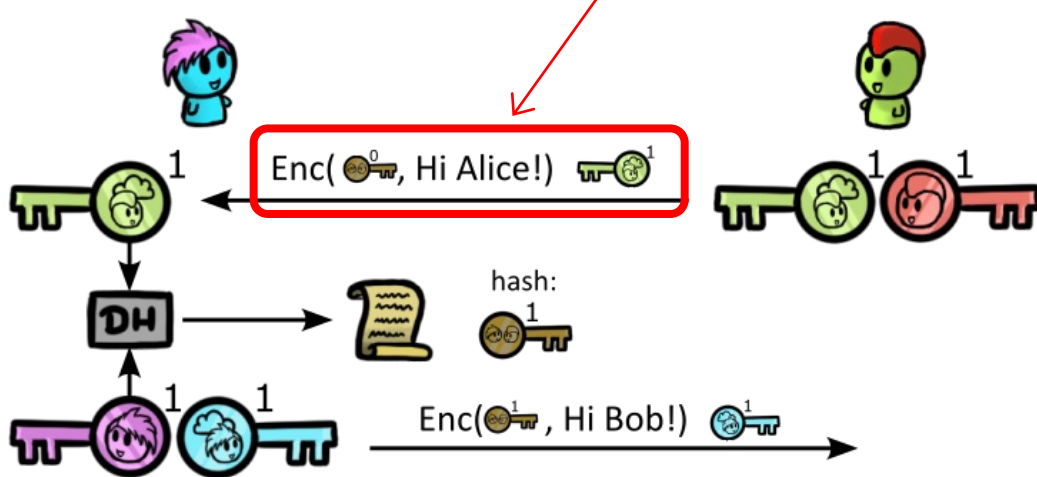
# OTR's DH Ratchet (incorrect)

- 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  $\text{Enc}(\text{Key}, \text{Message})$  for Encryption+MAC with a Key.




# OTR's DH Ratchet (incorrect)

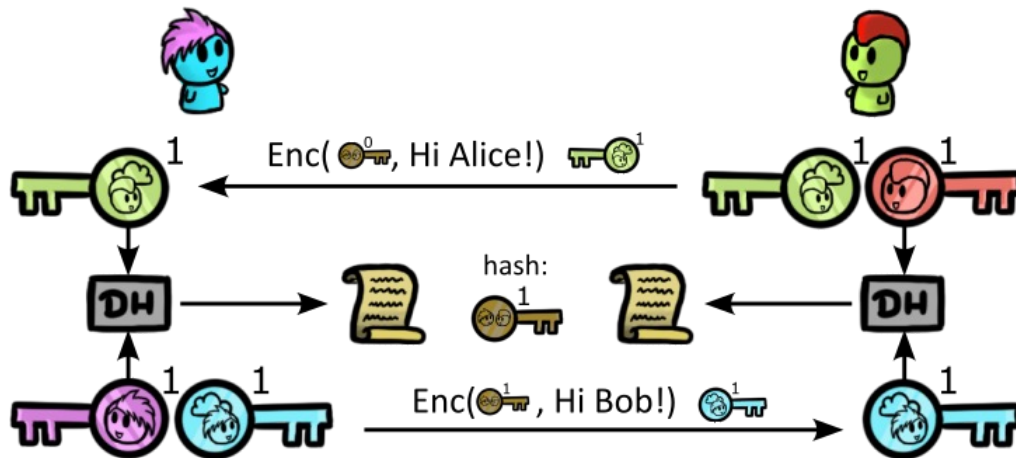
- Assume Alice and Bob have a pre-shared key 
- Assume Alice and Bob can have each other's long-term verification keys and there is a way to “**magically**” authenticate the **first message** Bob sends (for simplicity)...
- In these slides, we use the notation  $\text{Enc}(\text{Key}, \text{Message})$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




How does Bob recover the message?

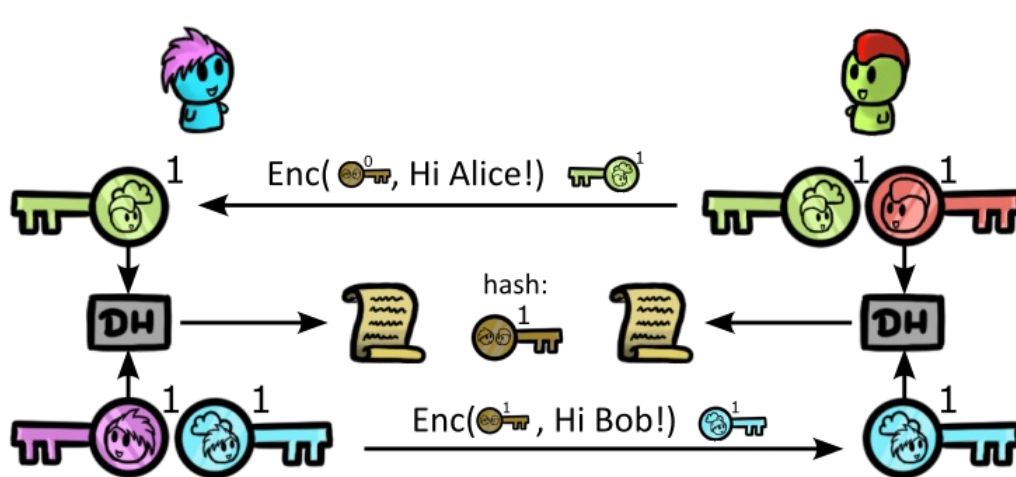
# OTR's DH Ratchet (incorrect)

- Assume Alice and Bob have a pre-shared key 
- 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  $\text{Enc}(\text{Key}, \text{Message})$  for Encryption+MAC with a Key.



# OTR's DH Ratchet (incorrect)

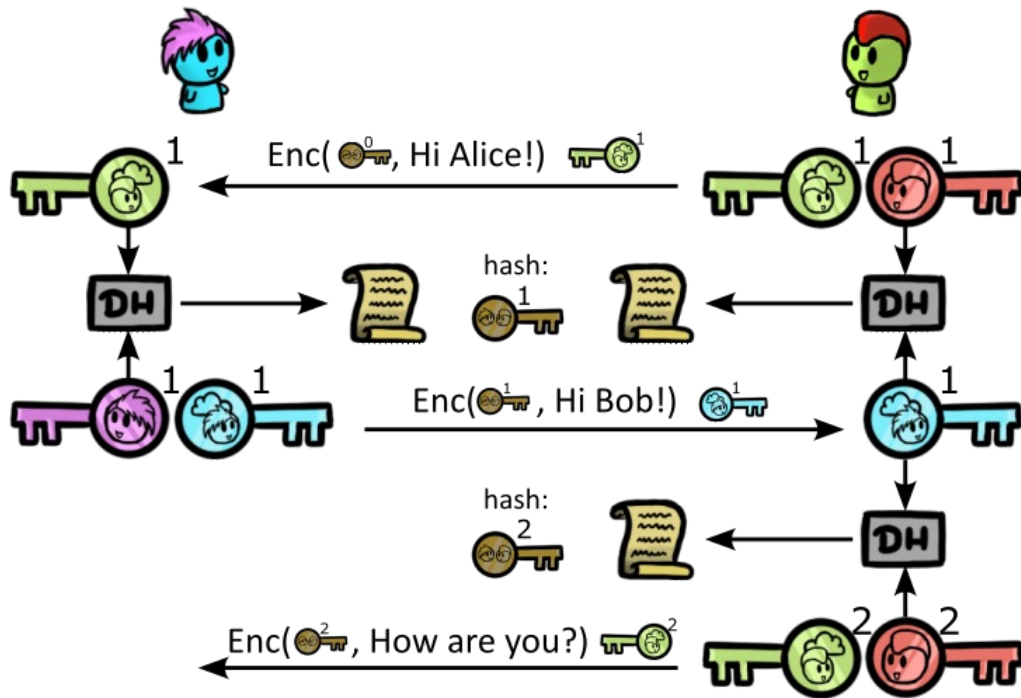
- Assume Alice and Bob have a pre-shared key 
- Assume Alice and Bob can have each other's long-term verification keys and there is a way to “**magically**” authenticate the **first message** Bob sends (for simplicity)...
- In these slides, we use the notation  $\text{Enc}(\text{Key}, \text{Message})$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



Following this logic, how does Bob reply to Alice?



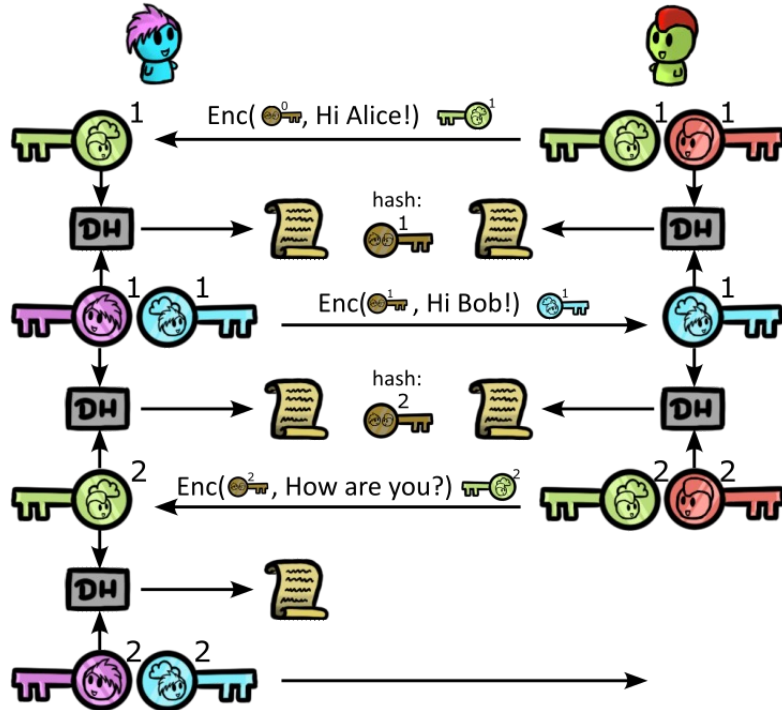
# OTR's DH Ratchet (incorrect)



- 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



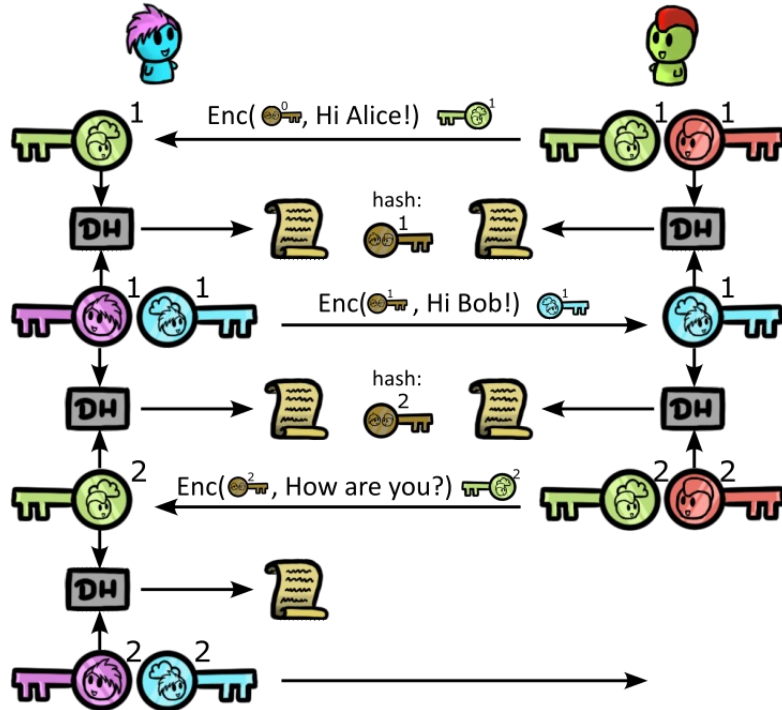
# OTR's DH Ratchet (incorrect)



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



# OTR's DH Ratchet (incorrect)

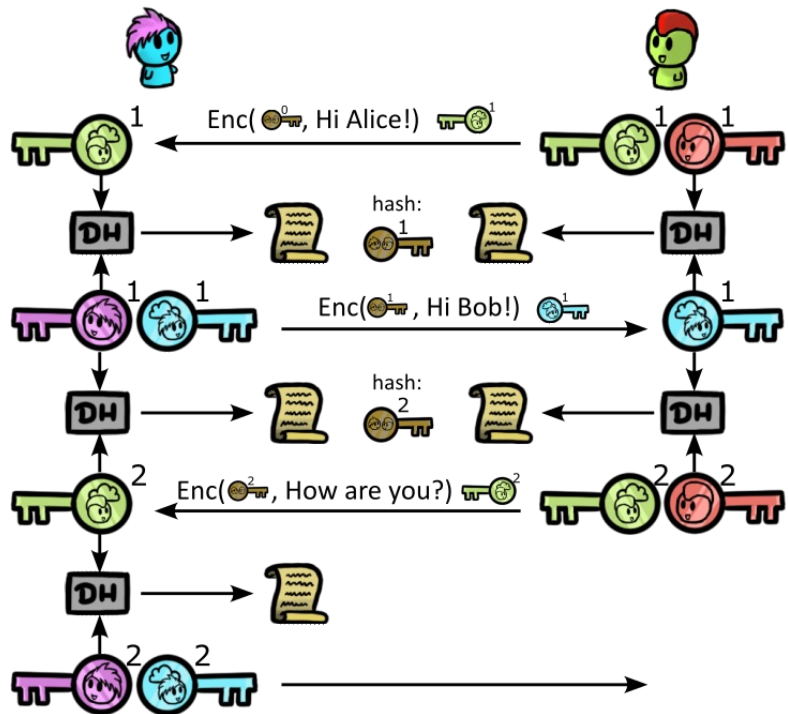


What happens if Eve learns a private key?  
E.g., Eve learns:



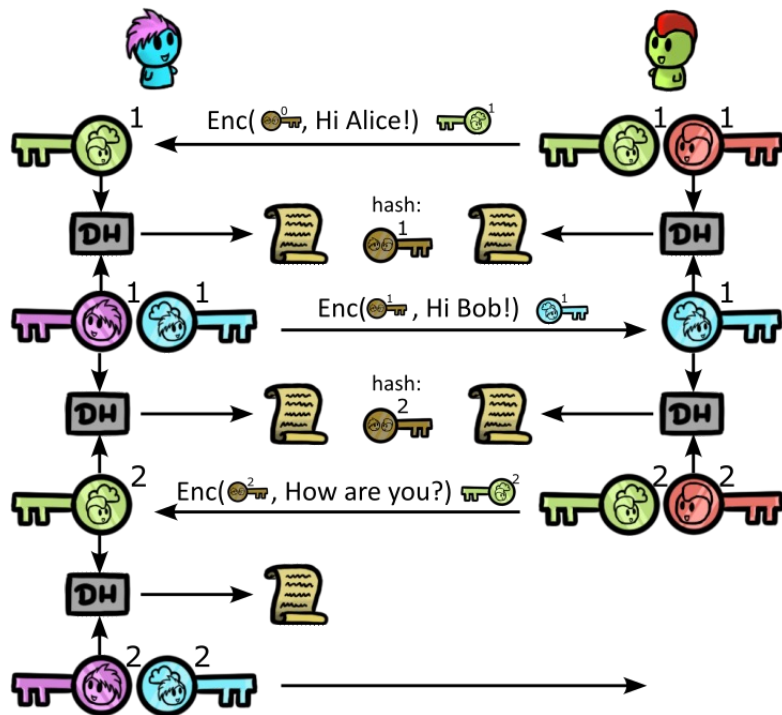
She can decrypt "Hi Bob!" and "How are you?"

# 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

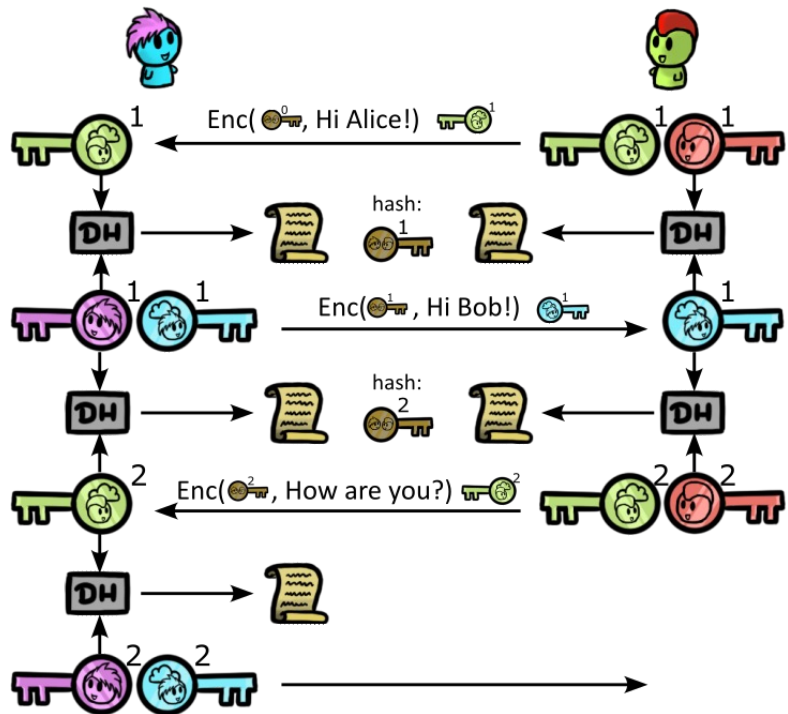
**Q:** Why are MACs deniable?

**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)



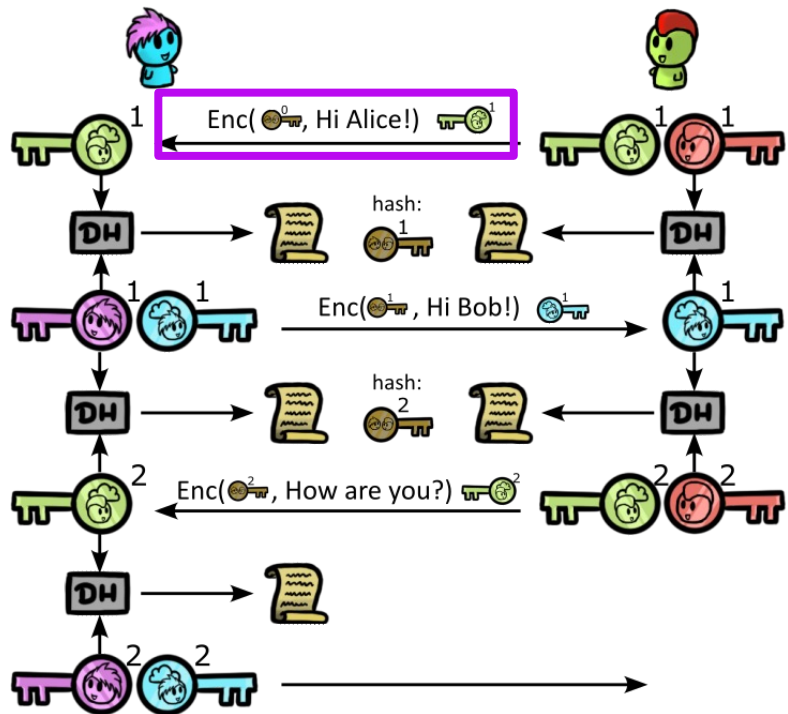
Remember, we assume Alice and Bob have 

- 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)

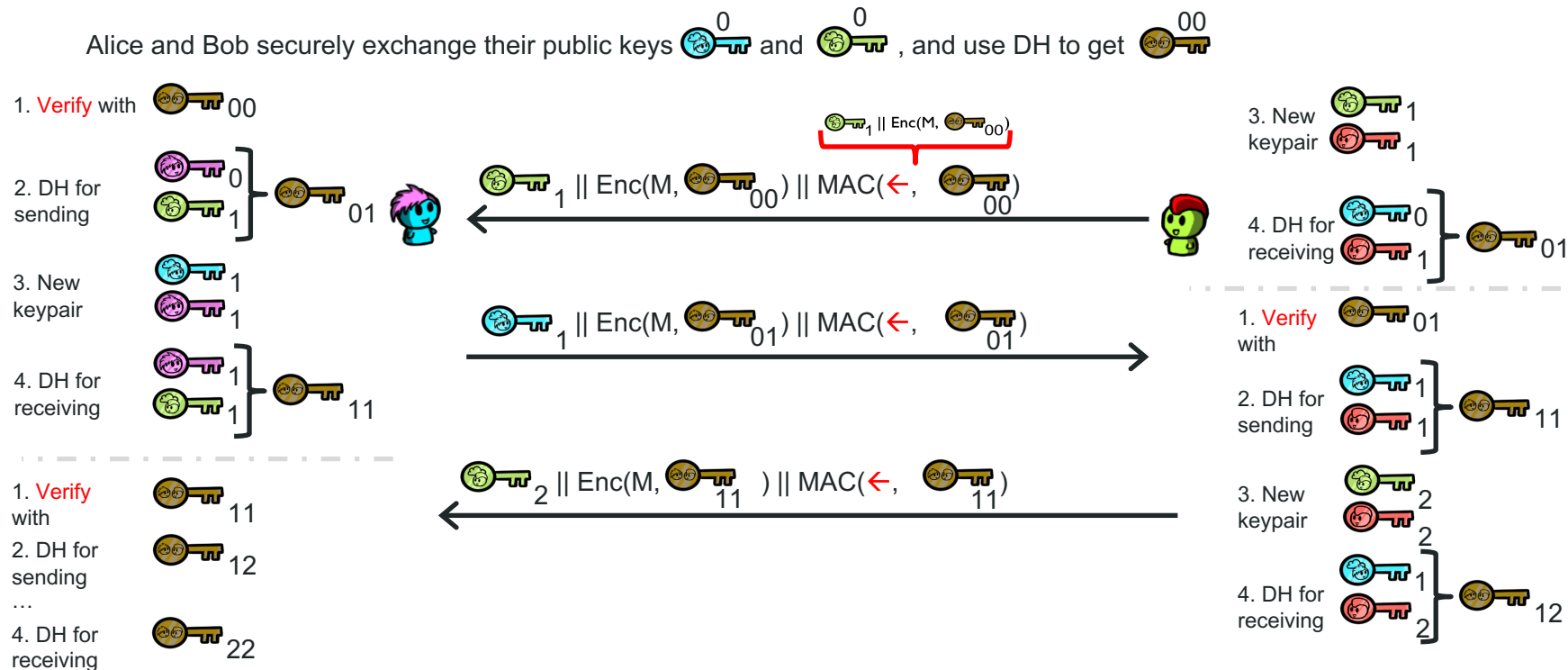


Remember, we assume Alice and Bob have 

- 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

---

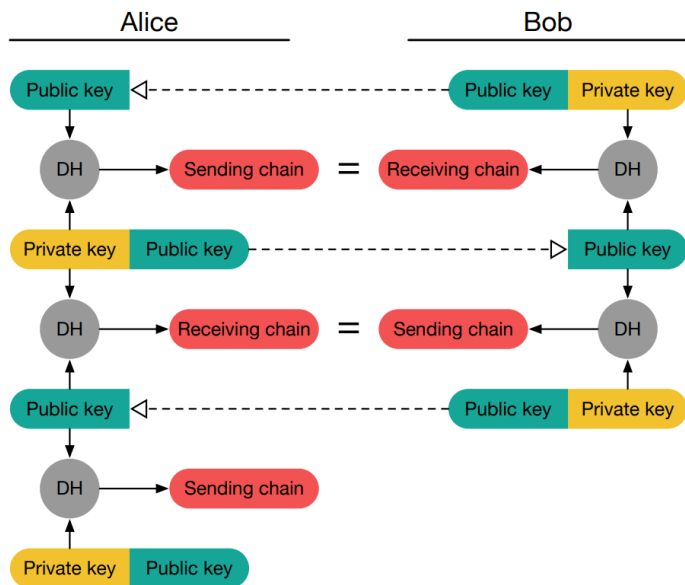
- Mobile app with companion desktop (Electron) client
  - OTR was less mobile-friendly
- Encryption protocol based on OTR
  - Double Ratchet Algorithm builds on OTR DH ratchet
  - Deniability ideas from OTR
- Protocol also used in other apps like WhatsApp, OMEMO extension for XMPP, etc.

# Signal

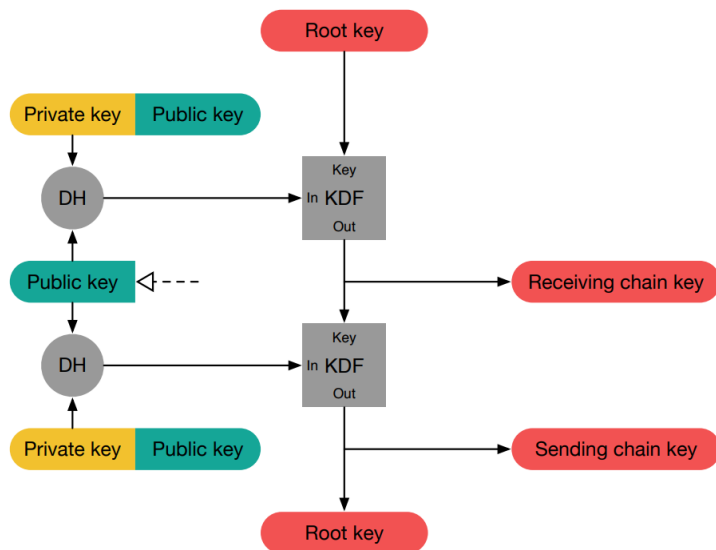
- Provides **forward secrecy**
  - Similar to OTR, it uses a “ratchet” technique to constantly rotate session keys.
- Provides **post-compromise security**
  - A leak of past or long-term keys will be healed by introducing new DH ratchet keys.
- Provides **improved deniability**
  - It uses a “Triple 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

## DH ratchet (asymmetric, like in OTR)



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



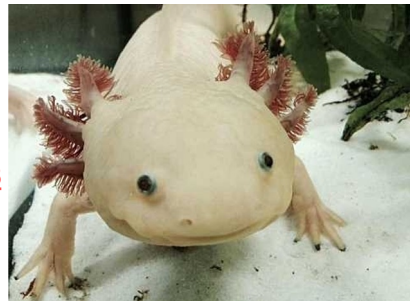
# The double ratchet

---

- Originally called **Axolotl ratchet** for its “self-healing” property (from the DH ratchet)
- It is very well explained on the [Signal website](#).

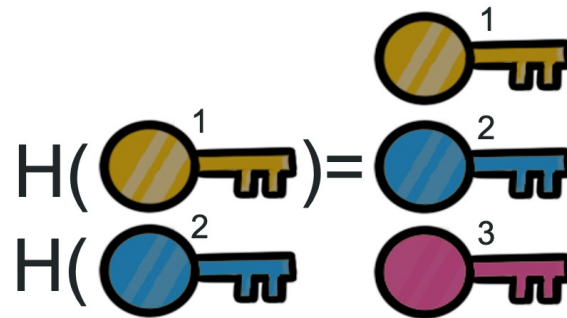
Photo: [th1098](#)

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



# Rationale for the KDF Ratchet

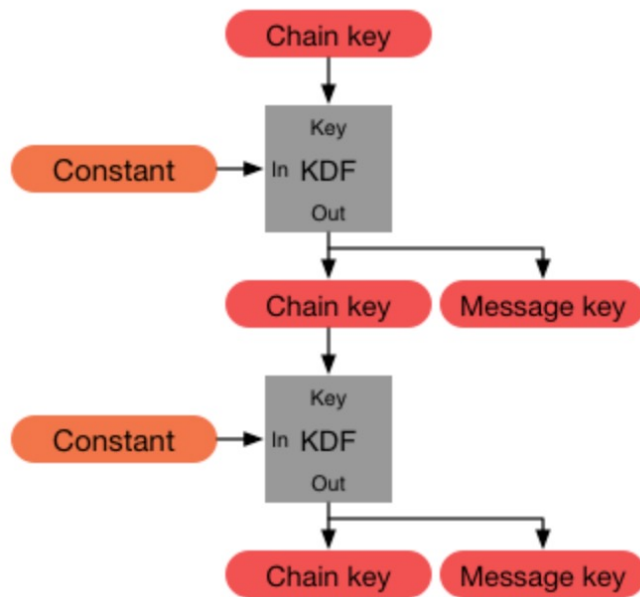
- What if instead of session keys, we had a new key for *each message*?
- We can do this deterministically
  - Simplified ratchet:  $K_{n+1} = H(K_n)$



- 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?

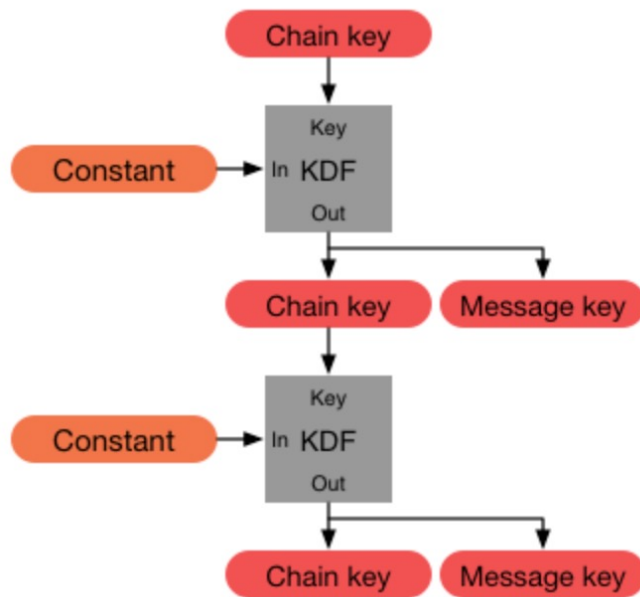




# KDF Ratchet

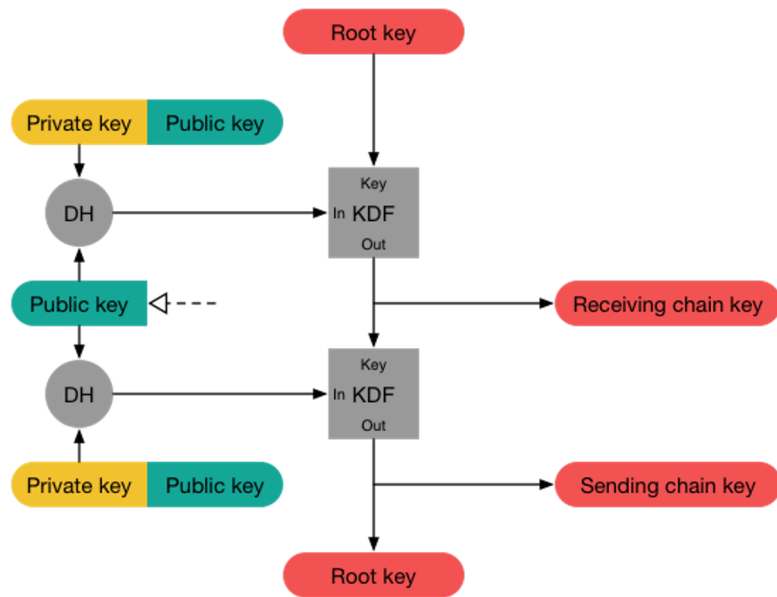
**Takeaway:** Message keys cannot be used to generate other chain or message keys, so it's safe to store old message keys for not-yet-delivered messages.

- 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

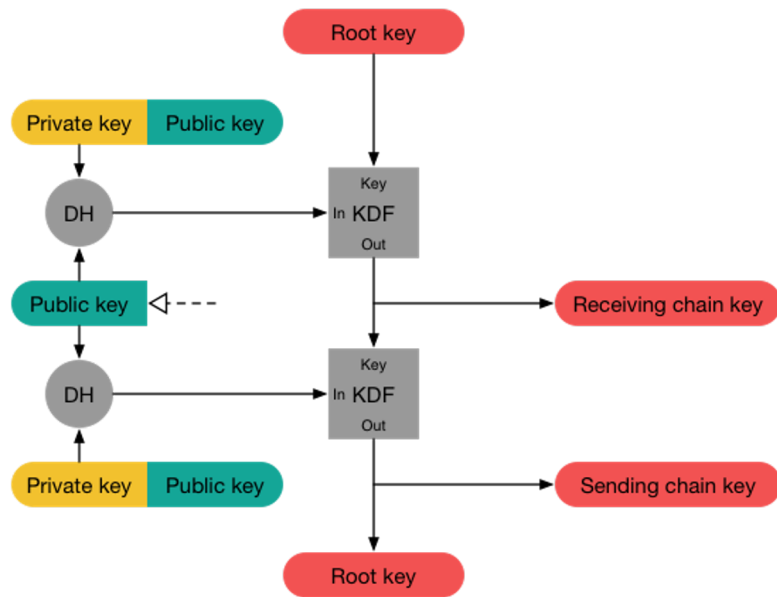
- Just like OTR. But now also:
- Output is used for generating **receiving chain** and **sending chain** keys
  - These are used as input for the KDF ratchet



# DH Ratchet

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

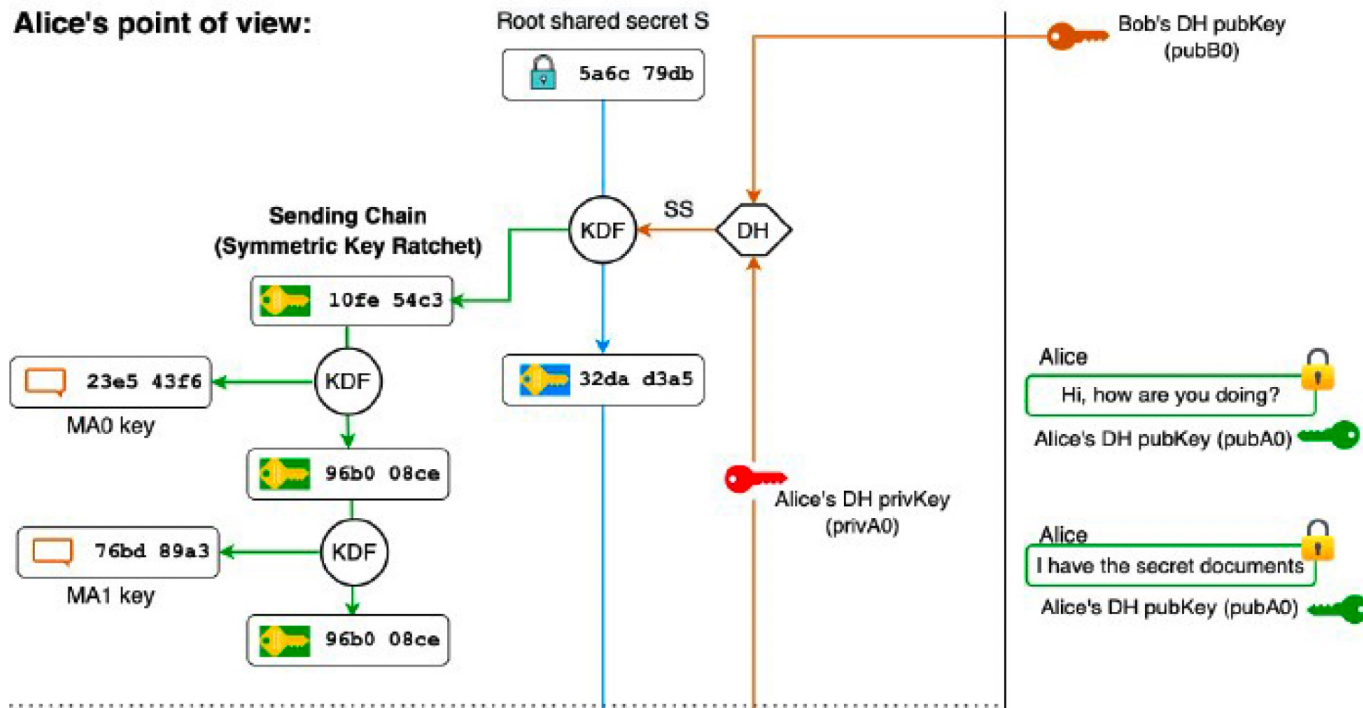
- Just like OTR. But now also:
- Output is used for generating **receiving chain** and **sending chain** keys
  - These are used as input for the KDF ratchet



# 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

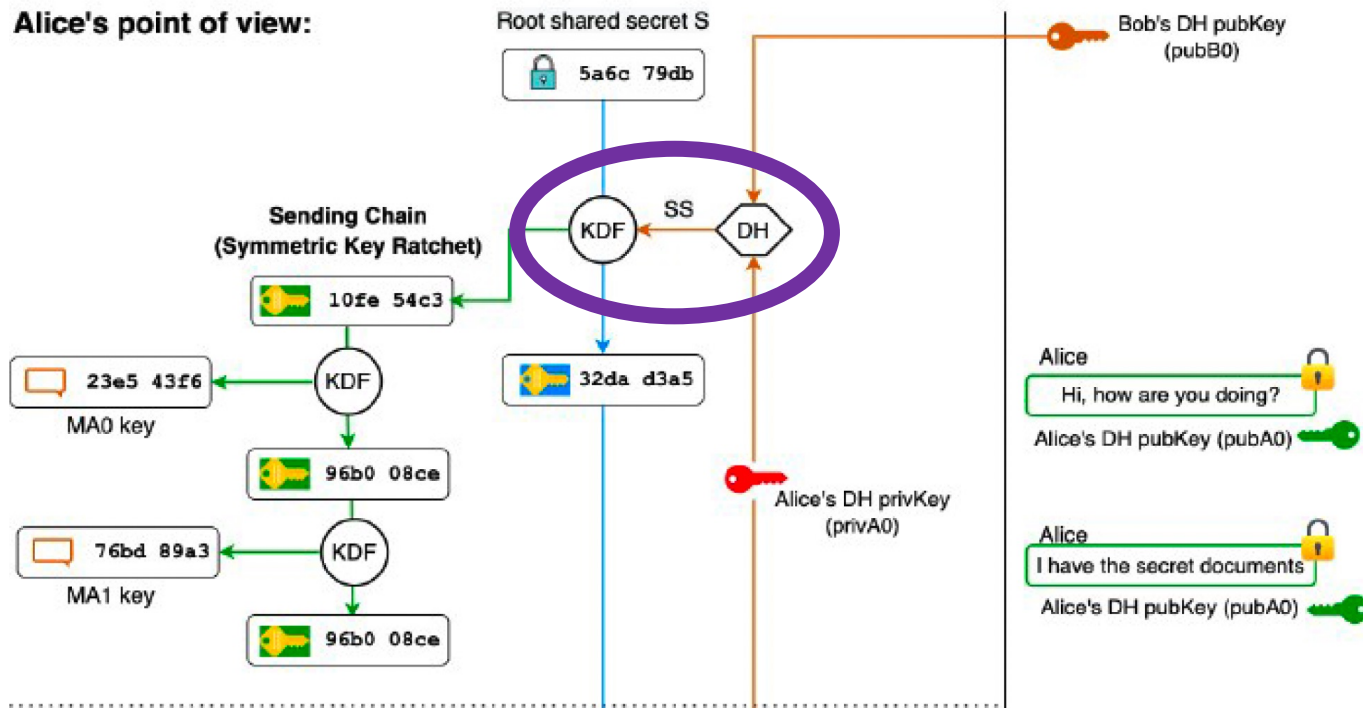
Alice's point of view:



# 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

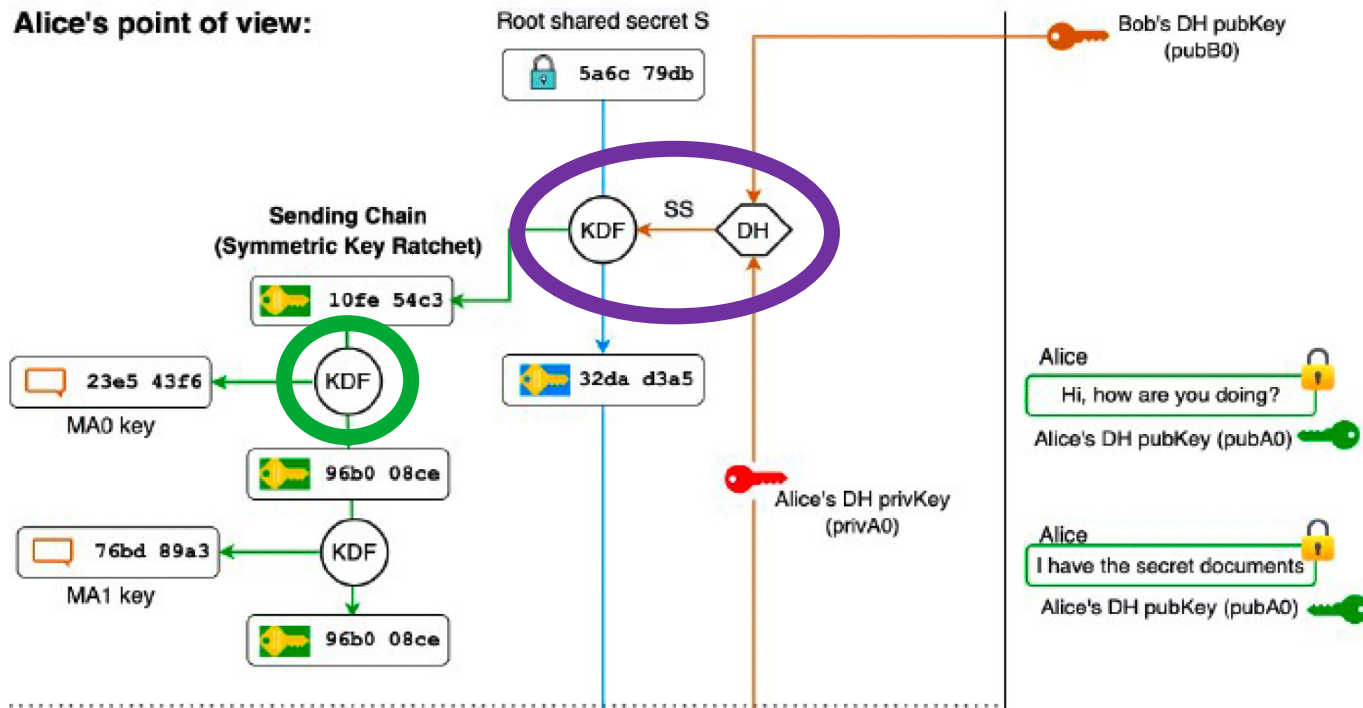
Alice's point of view:



# Double Ratchet Algorithm

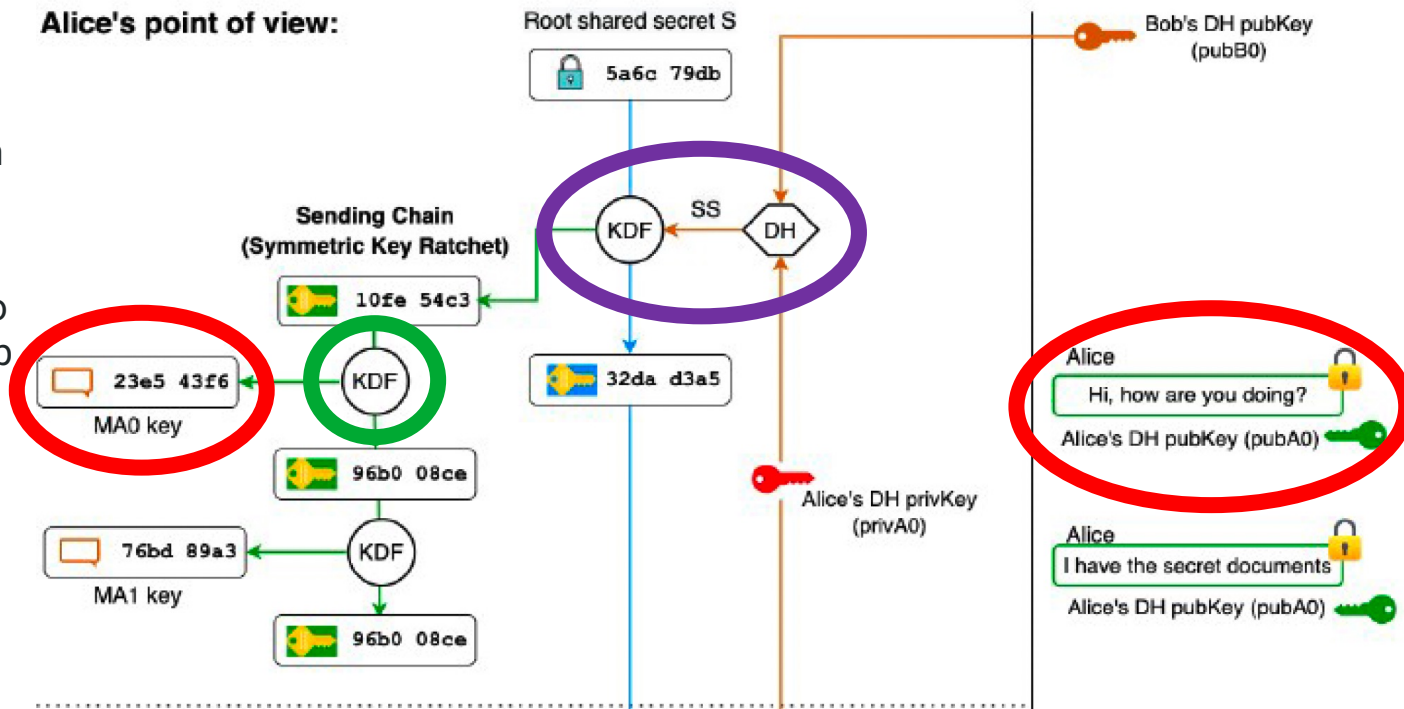
- Alice → Bob
- Alice and Bob do **DH** and get Alice's sending chain/Bob's receiving chain
- Alice **derives a key** with her sending chain
- Alice uses this **MAO** key to encrypt her message to Bob

**Alice's point of view:**



# Double Ratchet Algorithm

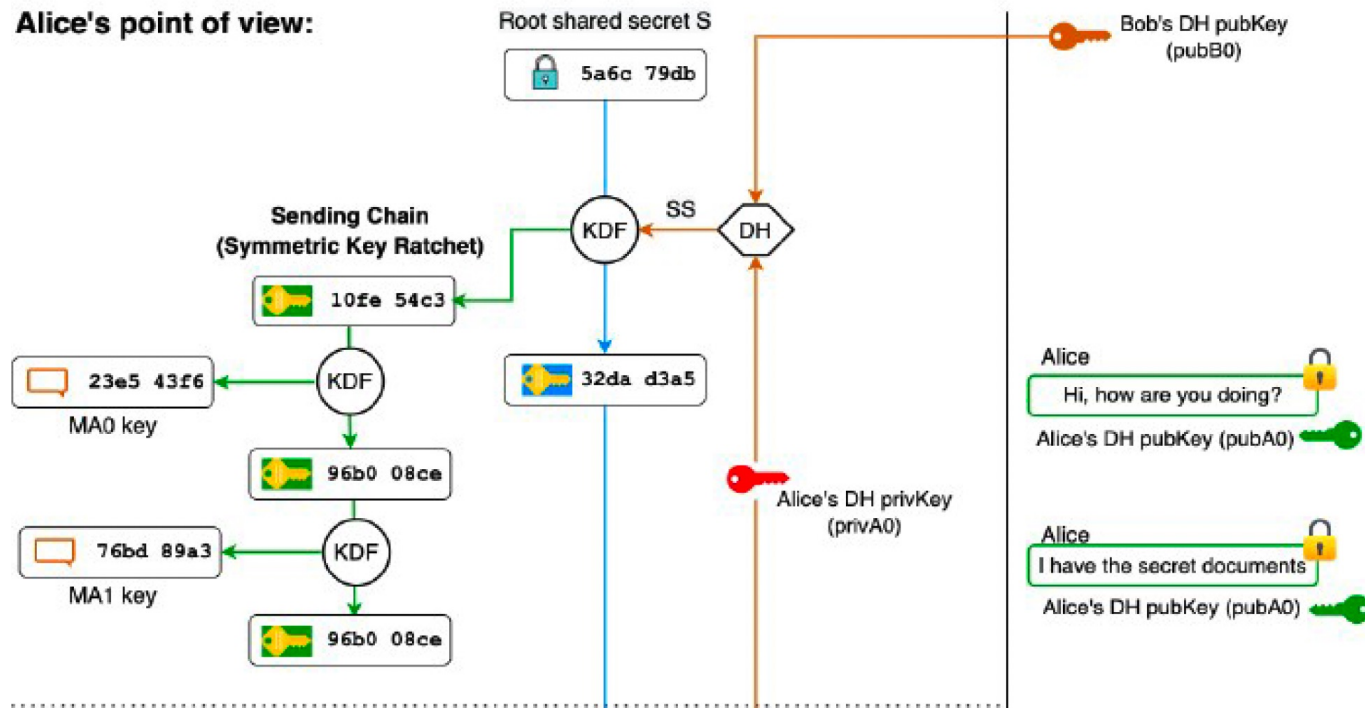
- Alice -> Bob
- Alice and Bob do **DH** and get Alice's sending chain/Bob's receiving chain
- Alice **derives a key** with her sending chain
- Alice uses this **MA0** key to encrypt her message to Bob



# 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

Alice's point of view:

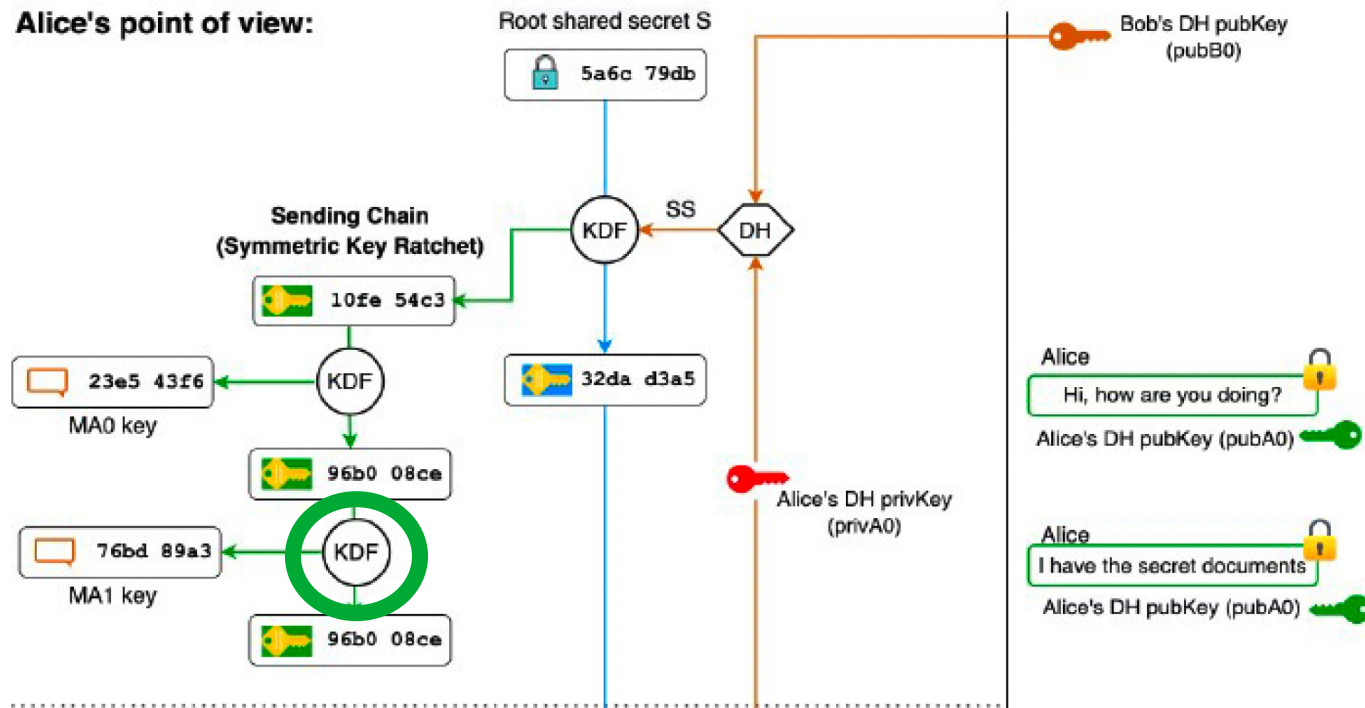




# Double Ratchet Algorithm

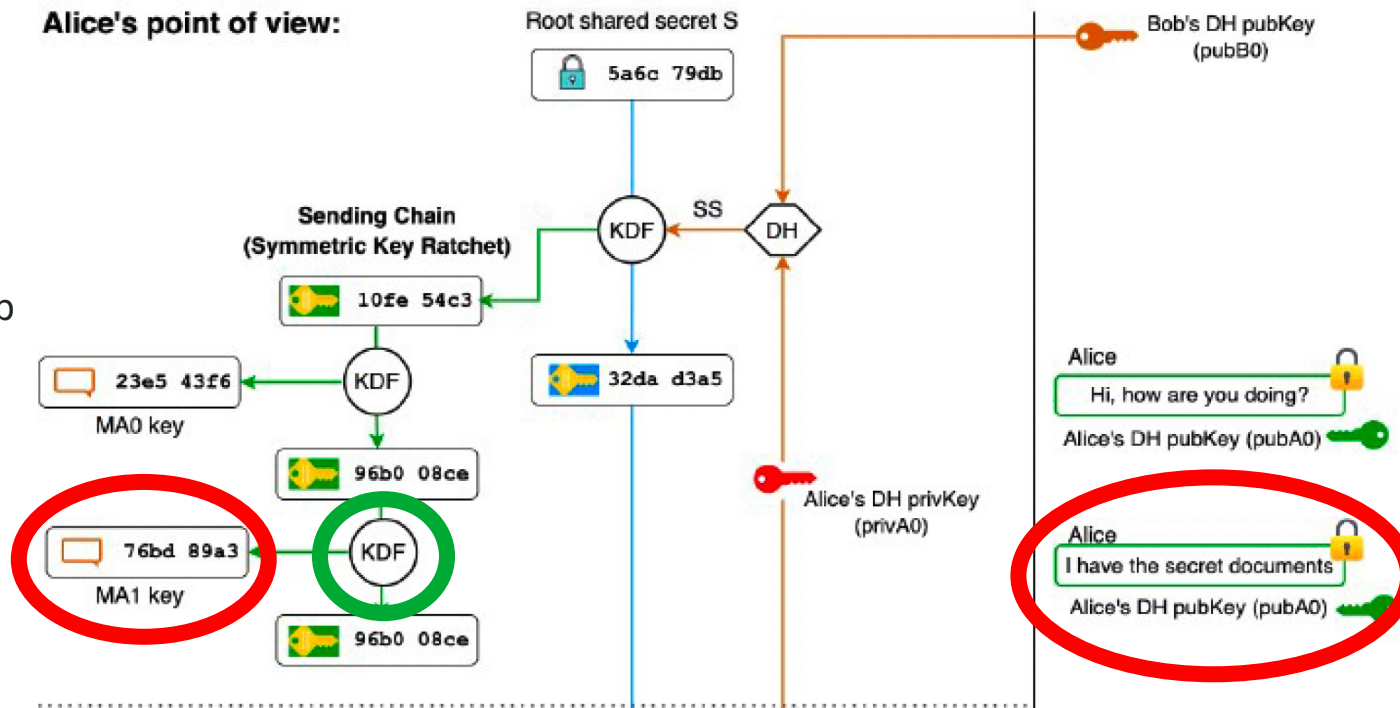
- Alice -> Bob (again)
- No new DH until Bob replies
- Alice **derives another key** with her sending chain
- Alice uses **MA1** key to encrypt her message to Bob

Alice's point of view:



# Double Ratchet Algorithm

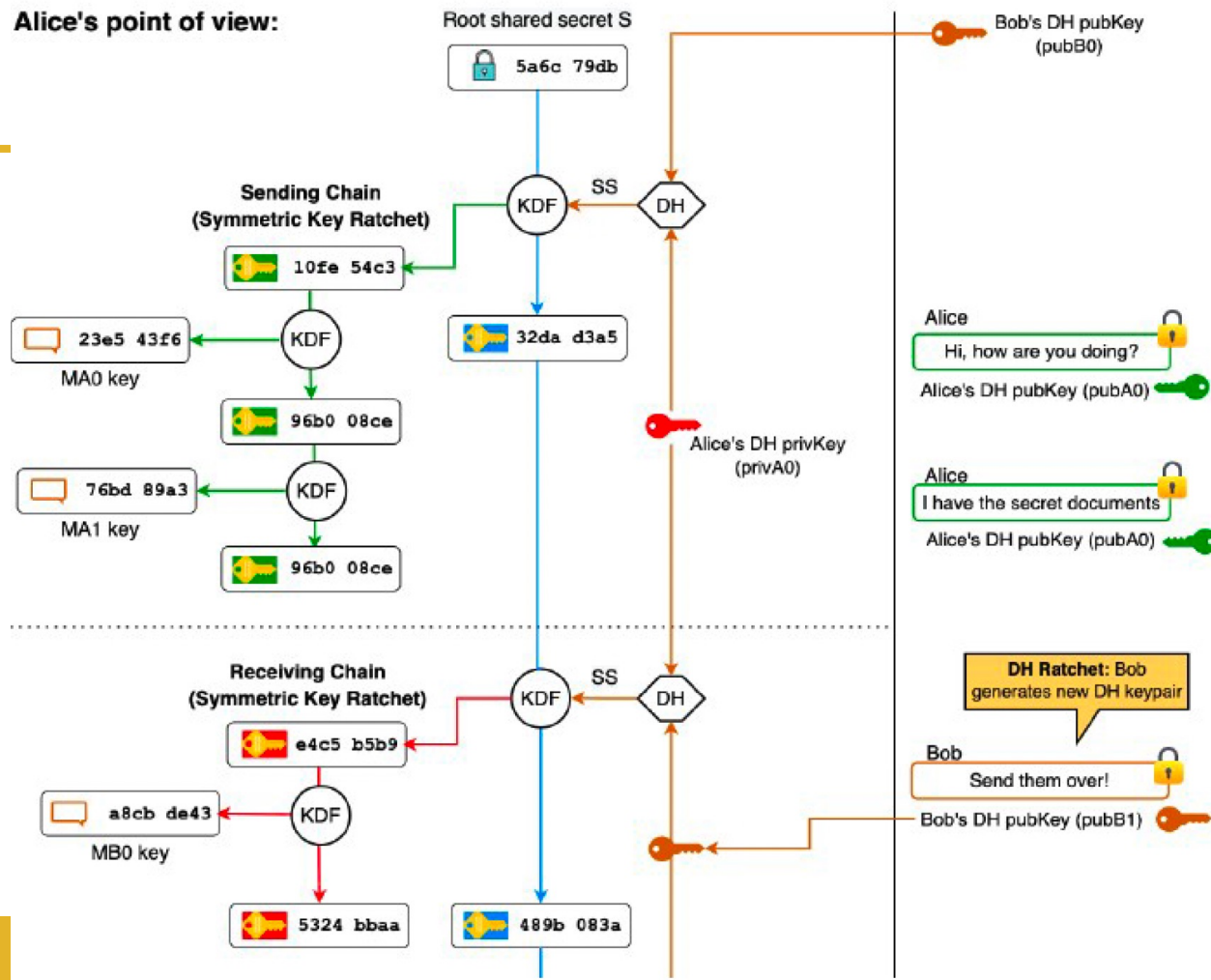
- Alice -> Bob (again)
- No new DH until Bob replies
- Alice **derives another key** with her sending chain
- Alice uses **MA1** key to encrypt her message to Bob



# Double Ratchet

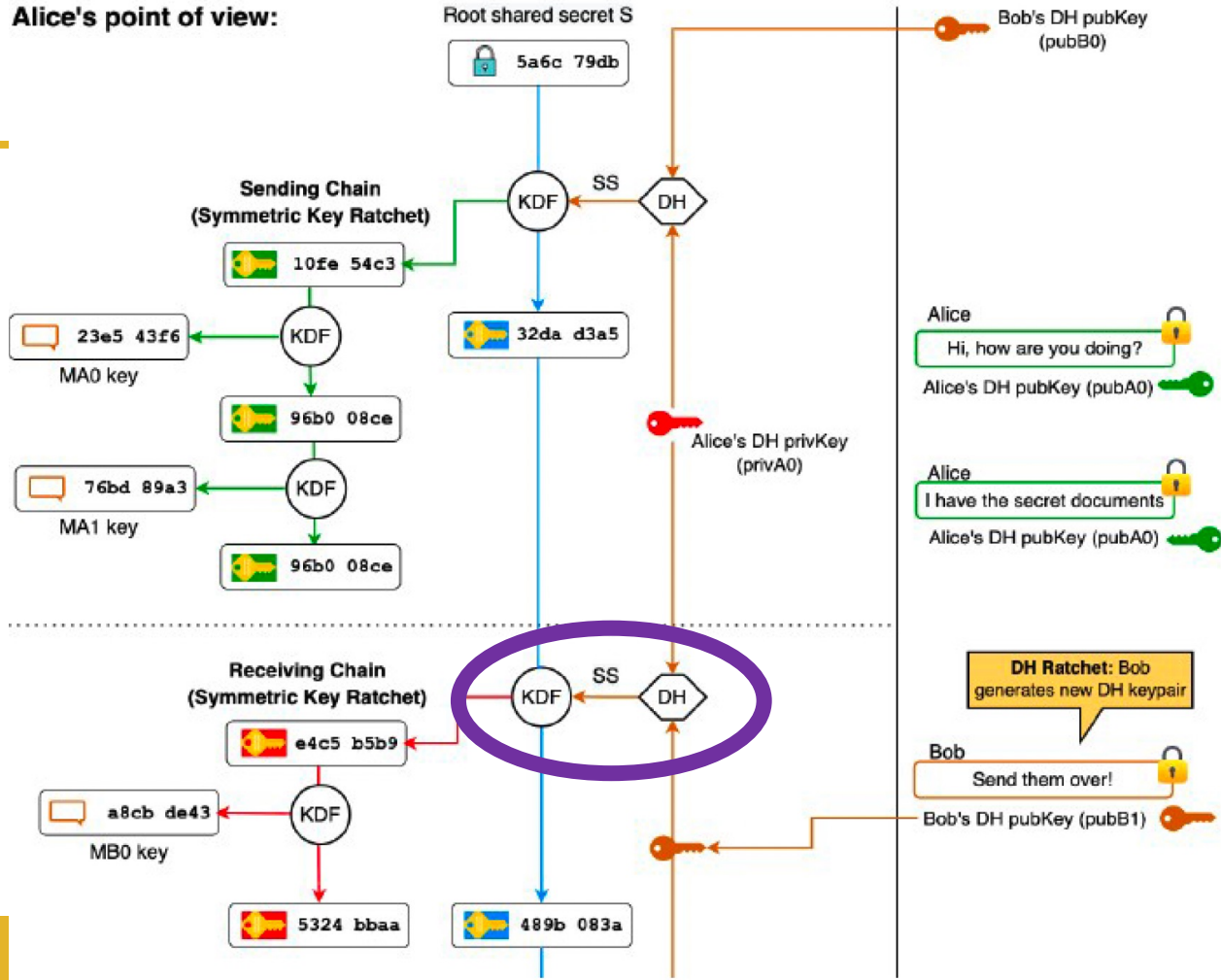
- Bob -> Alice
- Alice and Bob do DH and get Alice's receiving chain/Bob's sending chain
- Alice **derives a key** with her receiving chain
- Alice uses **MB0** key to decrypt a message from Bob

Alice's point of view:



# Double Ratchet

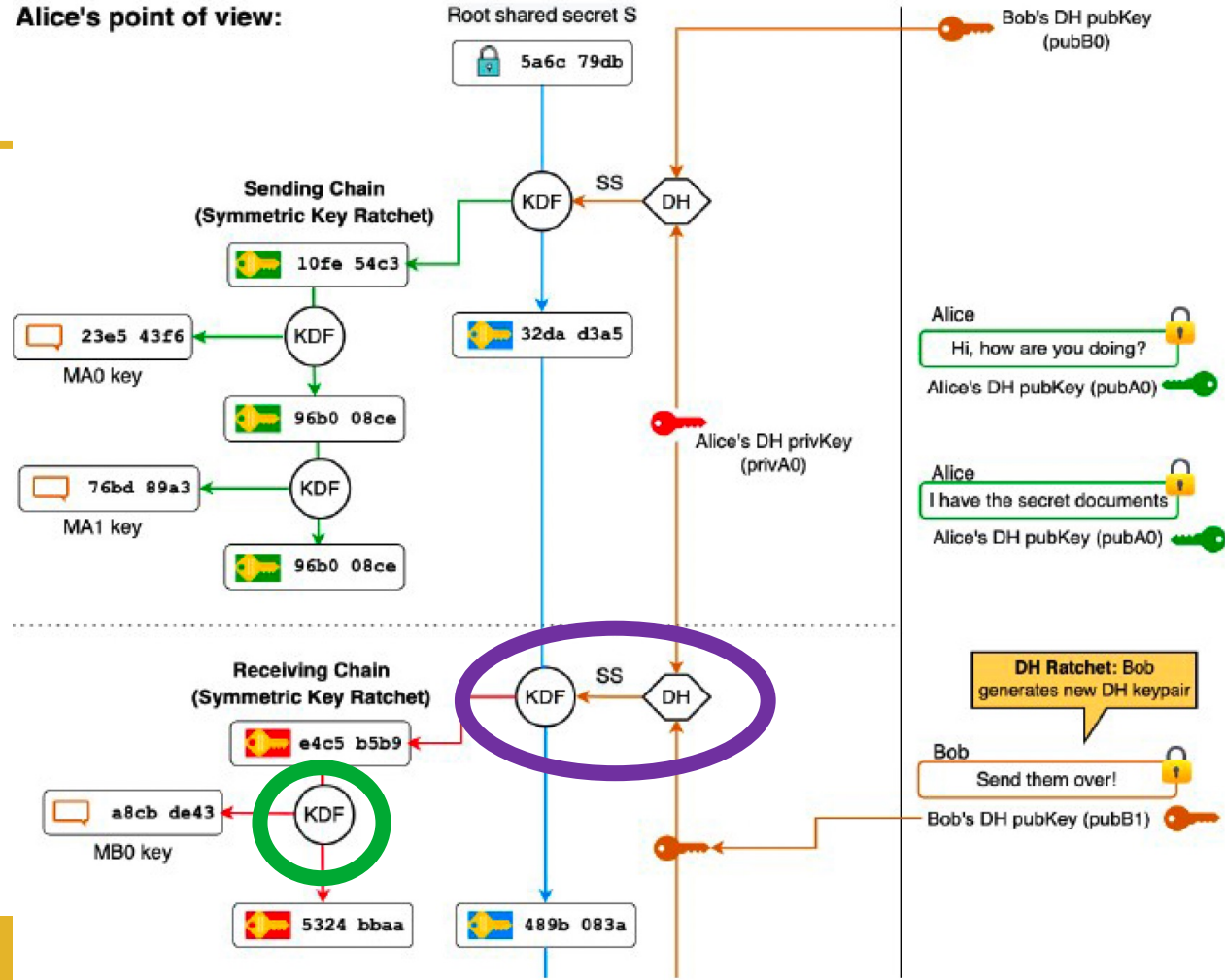
- Bob -> Alice
- Alice and Bob do DH and get Alice's receiving chain/Bob's sending chain
- Alice **derives a key** with her receiving chain
- Alice uses **MB0** key to decrypt a message from Bob



# Double Ratchet

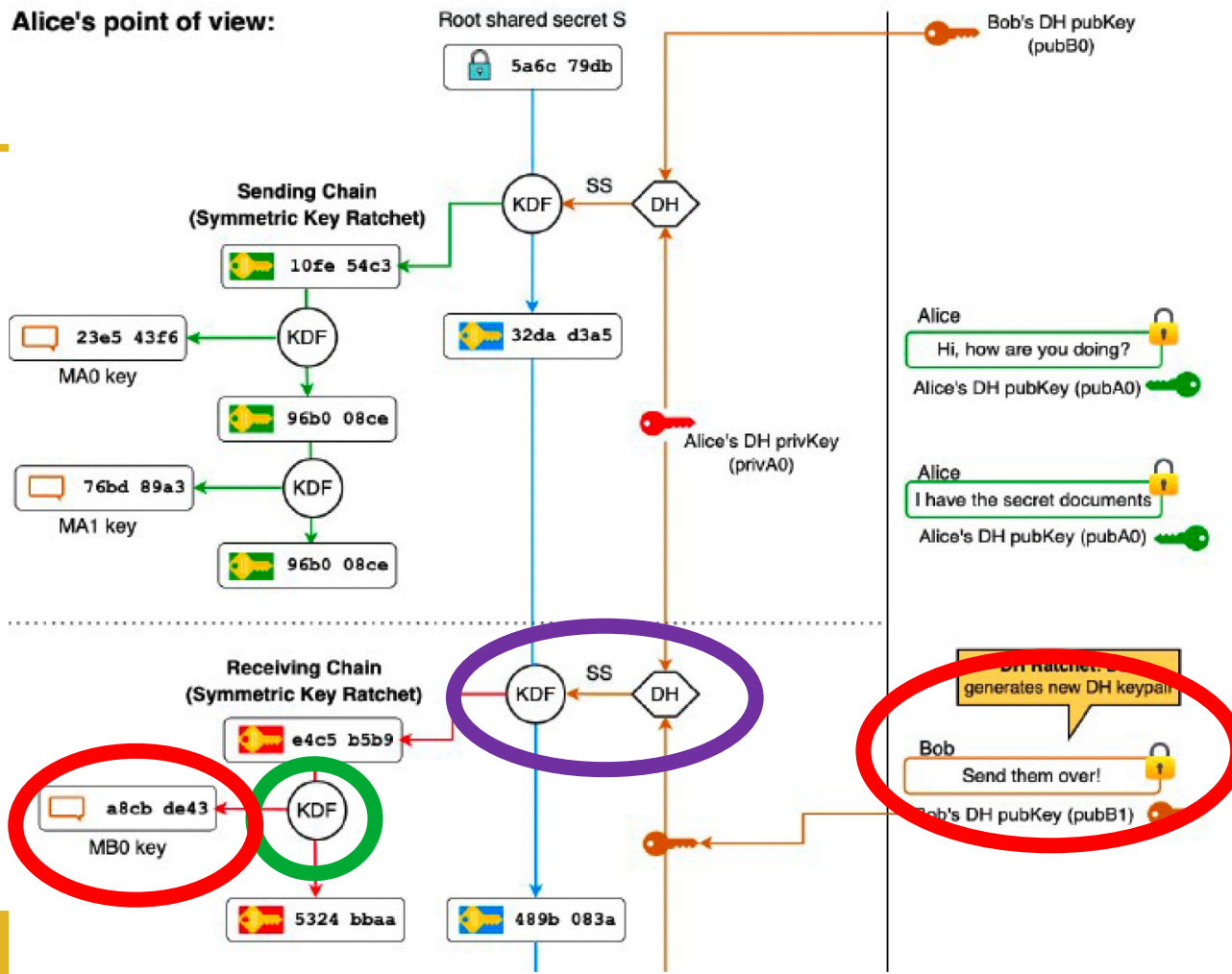
- Bob -> Alice
- Alice and Bob do DH and get Alice's receiving chain/Bob's sending chain
- Alice **derives a key** with her receiving chain
- Alice uses **MB0** key to decrypt a message from Bob

Alice's point of view:



# 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 😊