CS489/689 Privacy, Cryptography, Network and Data Security

Secure Messaging

Winter 2023, Tuesday/Thursday 8:30-9:50am

Hello

- I’m Vecna
- My pronouns are they/them
- I love crypto and am excited to talk about it :)

Today

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

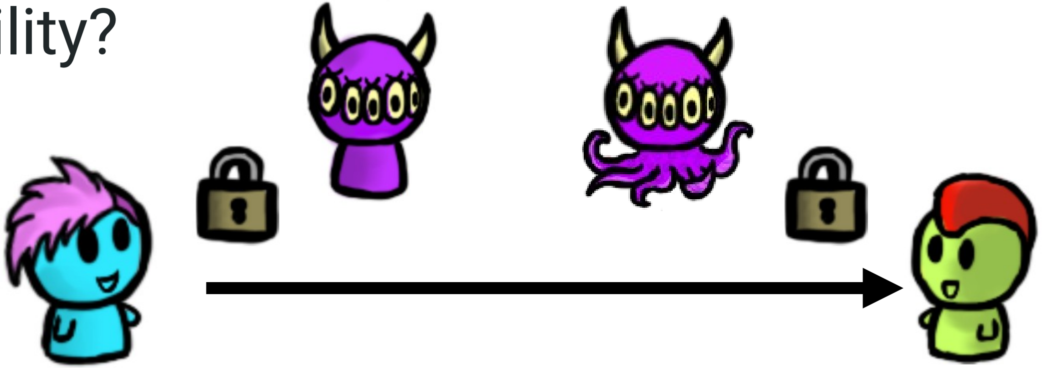
Secure Messaging Goals

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-repudiability?

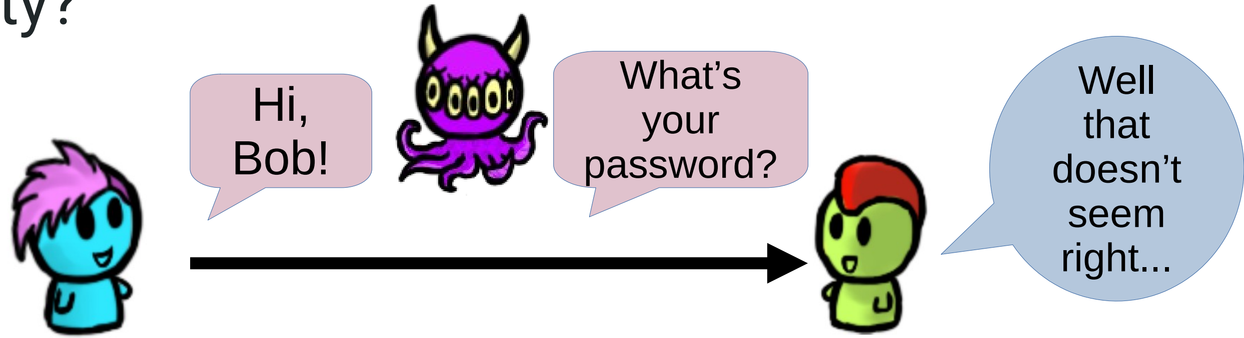
Secure Messaging Goals

- **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-repudiability?



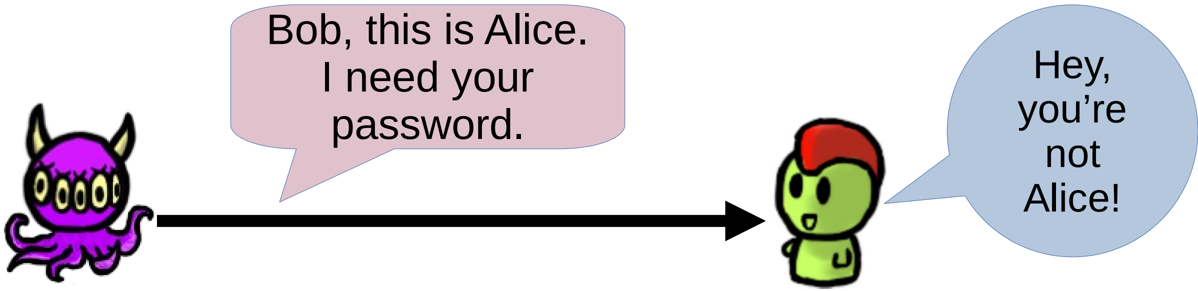
Secure Messaging Goals

- 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-repudiability?



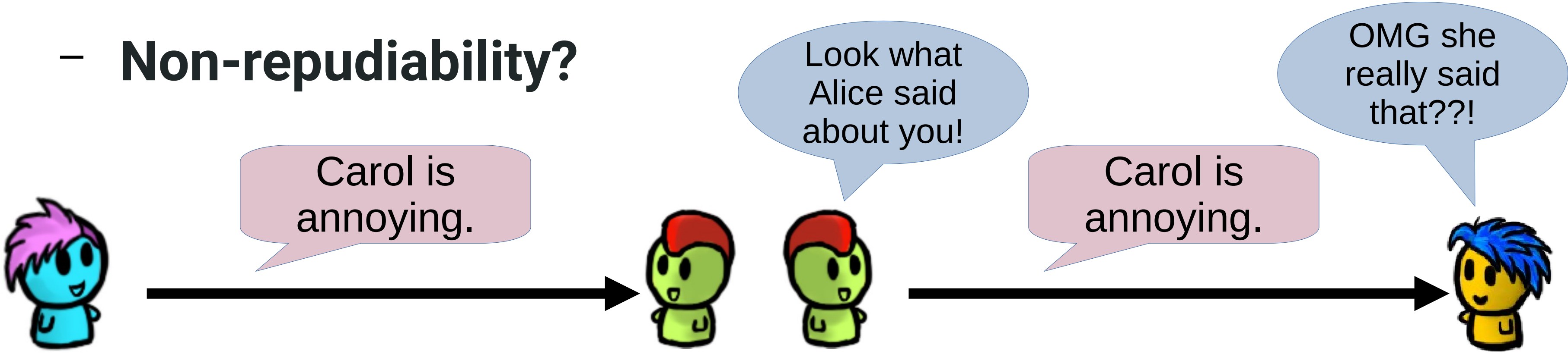
Secure Messaging Goals

- 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-repudiability?



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-repudiability?**



Pretty Good Privacy

PGP

- Public-key (actually hybrid) encryption tool
- Used for encrypted email (and other uses)
- Originally made by Phil Zimmermann in 1991
  - He got in a lot of trouble for it, since cryptography was highly controlled at the time

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

PGP

- PGP: Pretty Good Privacy (original program)
- OpenPGP: Open standard (RFC 4880)
- GPG/GnuPG: GNU Privacy Guard (a popular OpenPGP program)
- Many people just say “PGP” for all of the above
- Today, there are many programs which implement the OpenPGP standard
  - GNU Privacy Guard (gpg), Thunderbird, Evolution, Mailvelope, OpenKeychain, PGPro, Delta Chat, Proton Mail, ...

PGP

Message

PGP

Message

hash(Message) = h

PGP

Message

hash(Message) = h

sign(h , ) = sig

PGP

sig | Message

PGP

sig | Message

SK = secret key (random)

PGP

sig | Message

SK = secret key (random)

enc(sig | Message , SK)

(symmetric encryption)

PGP

enc(sig | Message , SK) = C1

(symmetric encryption)

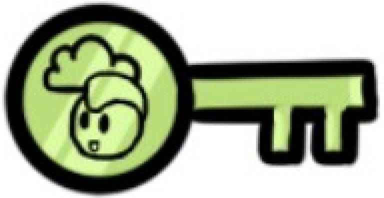
PGP

C1

SK

PGP

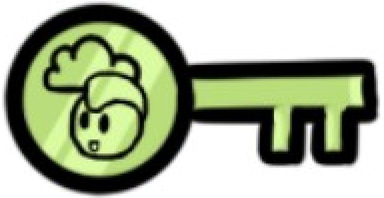
C1

enc(SK , ) = C2

(public key encryption)

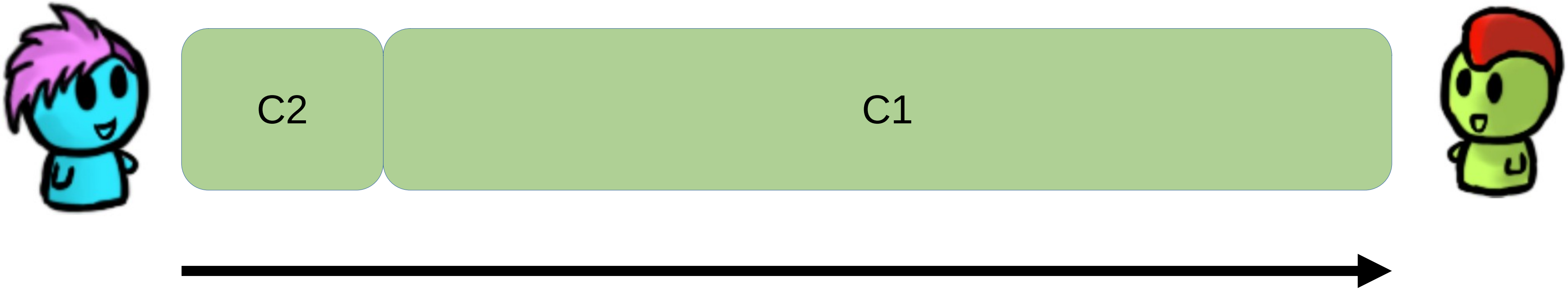
PGP

C1

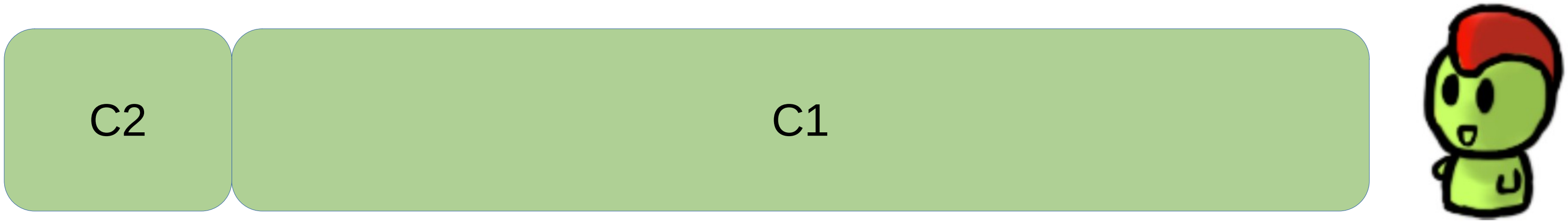
enc(SK , ) = C2

C2 C1

PGP

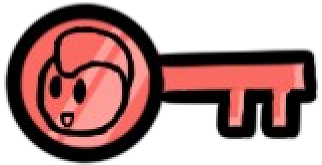


PGP



PGP

C2	C1

dec(C2 , ) = SK

(public key crypto)

PGP

C2 | C1

dec(C2 , ) = SK

(public key crypto)

C1 | SK

PGP

C1

SK

PGP

C1 SK

dec(C1 , SK)

(symmetric encryption)

PGP

C1

SK

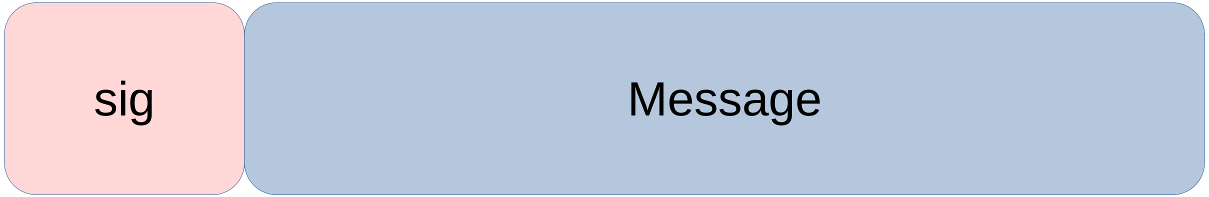
dec(C1 , SK)

(symmetric encryption)

=

sig | Message

PGP



PGP

sig | Message

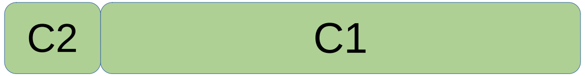
verify(sig , Message , )

Encrypted Messaging Goals and PGP

- Confidentiality
- Integrity
- Authentication
  - Non-repudiability?

Encrypted Messaging Goals and PGP

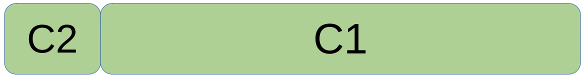
- Confidentiality



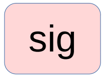
- Integrity
- Authentication
  - Non-repudiability?

Encrypted Messaging Goals and PGP

- Confidentiality



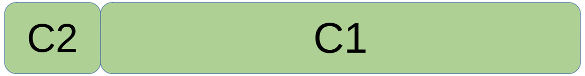
- Integrity



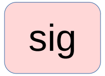
- Authentication
  - Non-repudiability?

Encrypted Messaging Goals and PGP

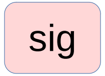
- Confidentiality



- Integrity



- Authentication



  - Non-repudiability?

Encrypted Messaging Goals and PGP

- Confidentiality

  C2 | C1

- Integrity

  sig

- Authentication

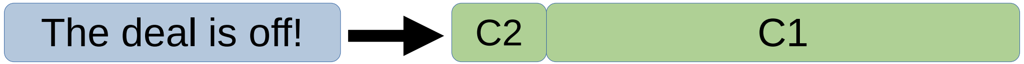
  sig

  - Non-repudiability?

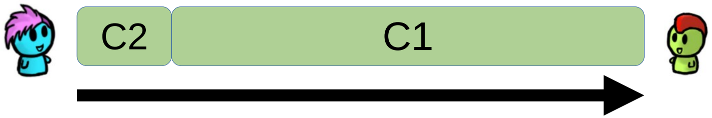
    sig

Act.

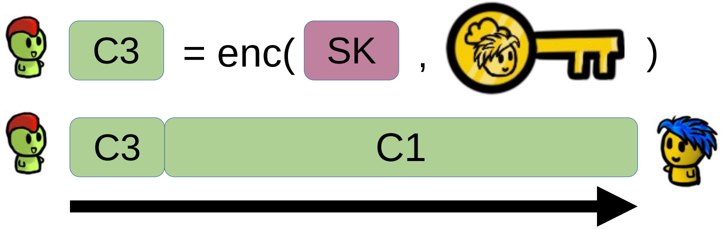
In-Class Activity

- Alice sends a message to Bob 

- Bob decrypts it, re-encrypts it for Carol, and sends it to Carol
- Does Carol believe that Alice intended this message for her? (Assume Carol ONLY relies on the cryptography.)
- If so, how can Alice prevent this attack **without changing the protocol**?



- Submit to Learn:
  - Ideas for how to prevent this attack



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 (in-person, through a secure channel, etc.)
- 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-----
```

Fingerprints

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

Verifying Public Keys

- Alice and Bob have verified each other. Great!
- But verifying is hard
  - Inconvenient if possible at all
  - Bob and Carol may not know each other well
- What if Bob and Carol can't verify each other?
- (Would it help if Carol has verified Alice?)

Signing Keys

- Once Alice has verified Bob's key, she uses her certification key to sign Bob's key
  - (By default, certification key == signature key)
- This is effectively the same as Alice signing a message saying "I have verified that the key with [Bob's fingerprint] belongs to Bob"
- Bob can attach Alice's signature to the key he has published somewhere
- (Are there any issues with doing this?)

Web of Trust

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

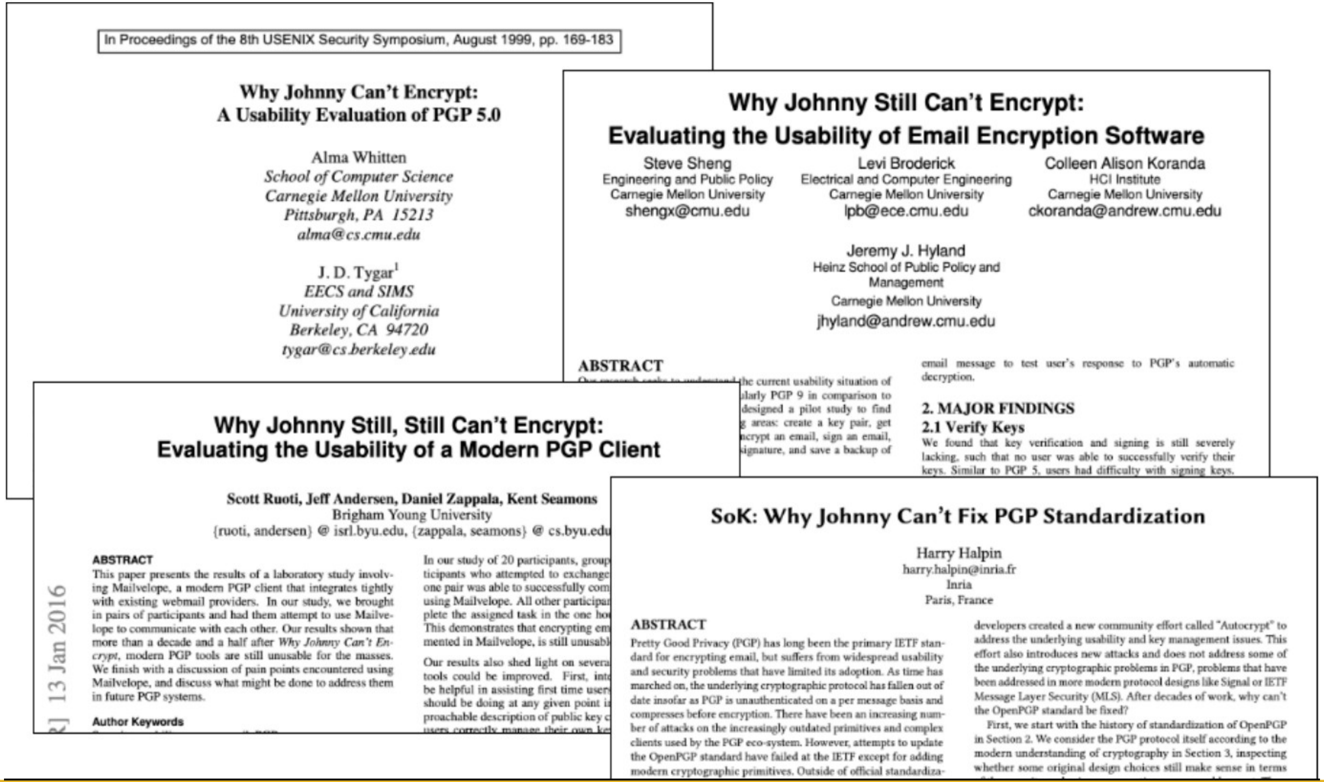
Awesome!

- If Alice and Bob want to have a private conversation by email:
  - They create their keys
  - They exchange their keys (possibly relying on the WoT)
  - They send signed and encrypted messages back and forth
- 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[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

...the current usability situation of ...larly PGP 9 in comparison to ...designed a pilot study to find ...areas: create a key pair, get ...crypt 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 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

In our study of 20 participants, group... ticipants who attempted to exchange... one pair was able to successfully com... using Mailvelope. All other participa... plete the assigned task in the one ho... This demonstrates that encrypting em... mented in Mailvelope, is still unusabl...

Our results also shed light on severa... tools could be improved. First, int... be helpful in assisting first time user... should be doing at any given point i... proachable description of public key c...

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 standardization 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 in insecure ways

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-repudiability

- 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

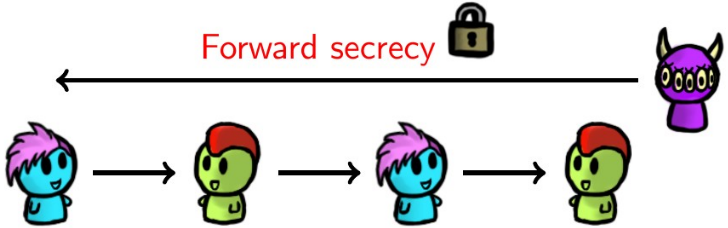
Off-The-Record (OTR) Messaging

OTR

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

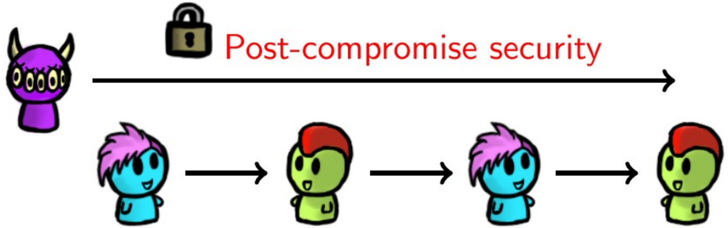
Goals of Off-The-Record Messaging

- **(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



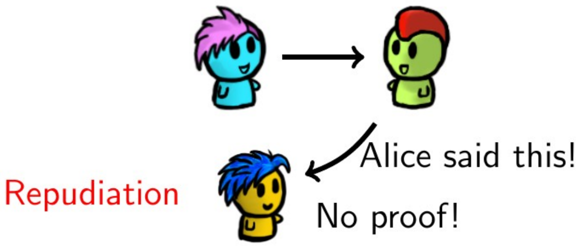
Goals of Off-The-Record Messaging

- (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



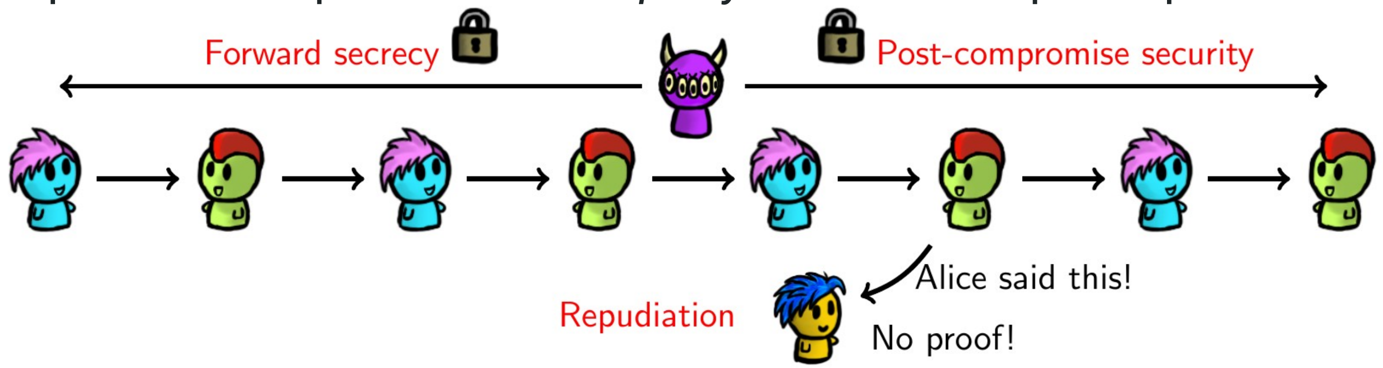
Goals of Off-The-Record Messaging

- (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

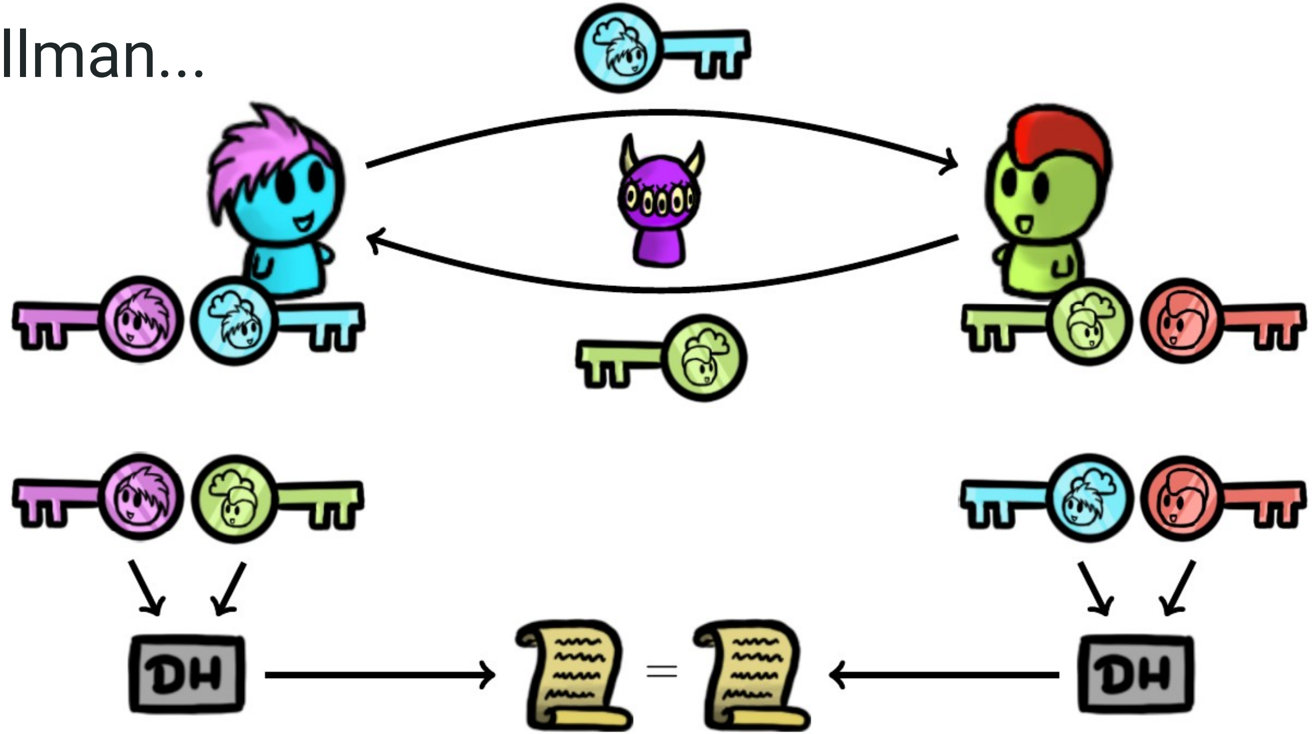


Forward Secrecy

- Forward secrecy: Key compromise doesn't reveal past messages
- How can we accomplish that?
- Change the key!
- Old keys must be securely deleted

Forward Secrecy (one approach)

- Recall Diffie-Hellman...

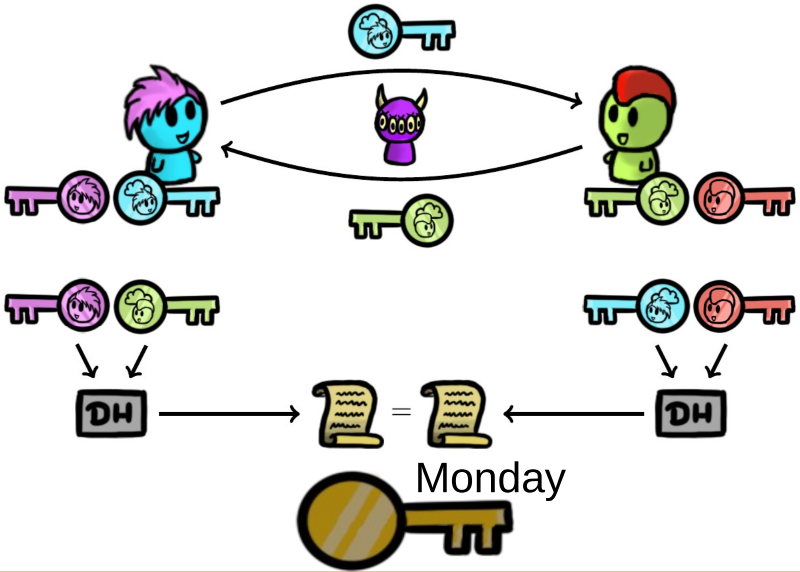


Forward Secrecy (one approach)

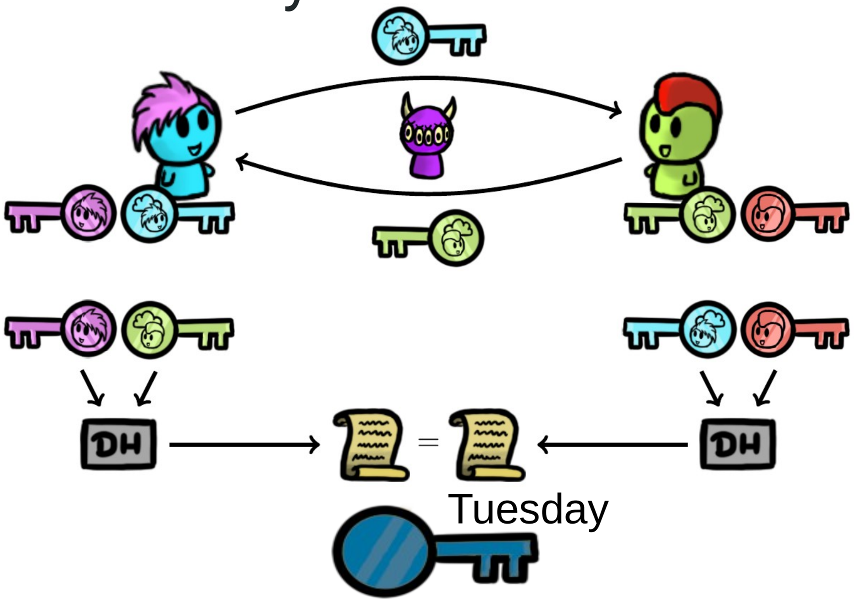
- Alice and Bob have ephemeral (temporary) “sessions”
- Alice produces ephemeral DH keys (a, g^a)
  - She signs the public key with her long-term key A
- Bob produces ephemeral DH keys (b, g^b)
  - He signs the public key with his long-term key B
- Alice and Bob use shared secret g^{ab}
- They make new keys later

Forward Secrecy (one approach)

- Alice and Bob talk on Monday...



- Alice and Bob talk on Tuesday...



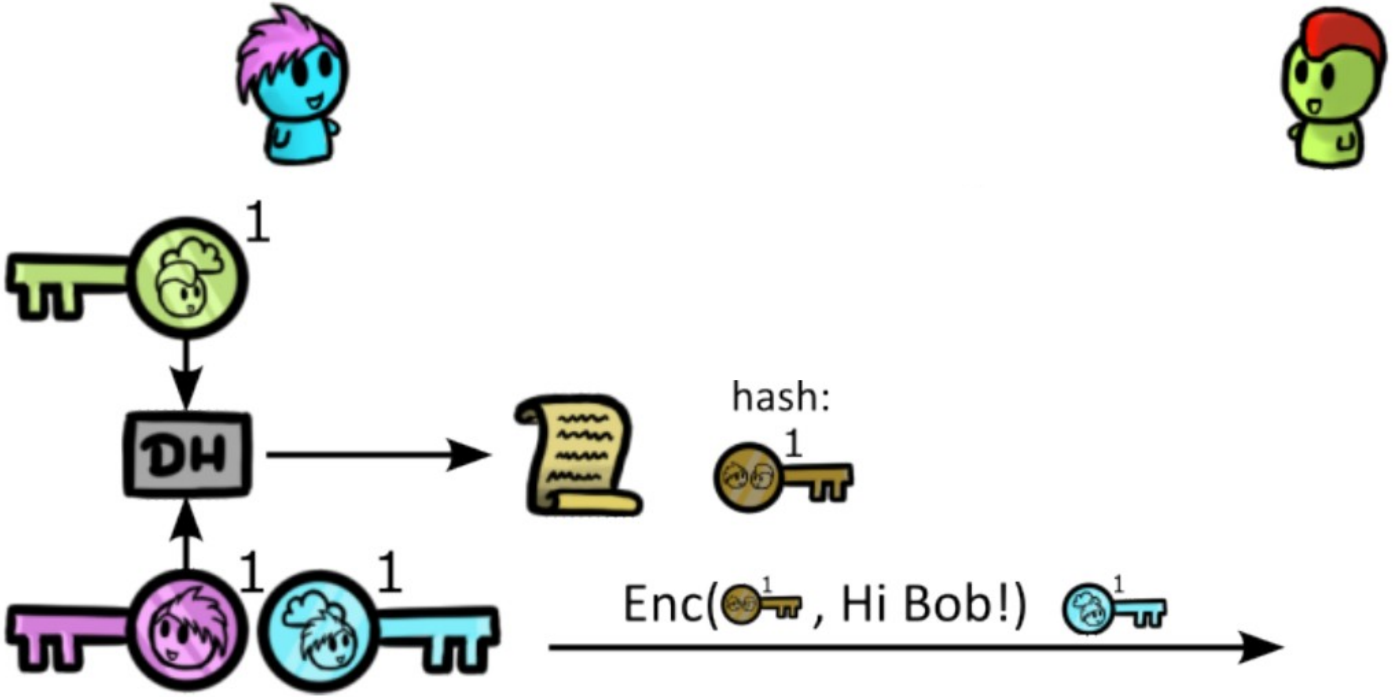
Forward Secrecy (one approach)

- Eve can compromise a session but not everything
- Problems?
  - Alice can't start a session unless Bob is online
  - Eve can still compromise a whole session
  - We'll see other ideas later

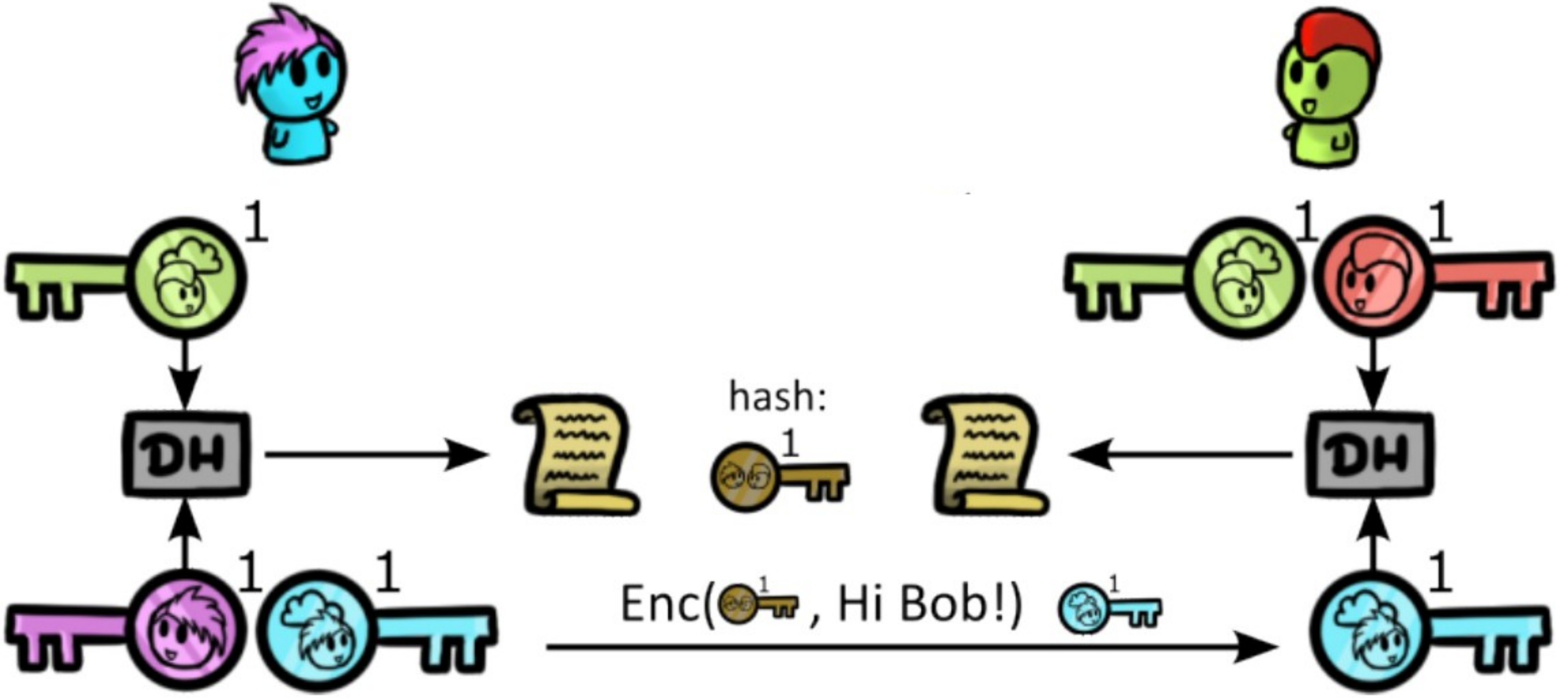
Forward Secrecy in OTR

- What if we make the sessions as short as possible?
- What if new sessions don't have to be negotiated interactively?

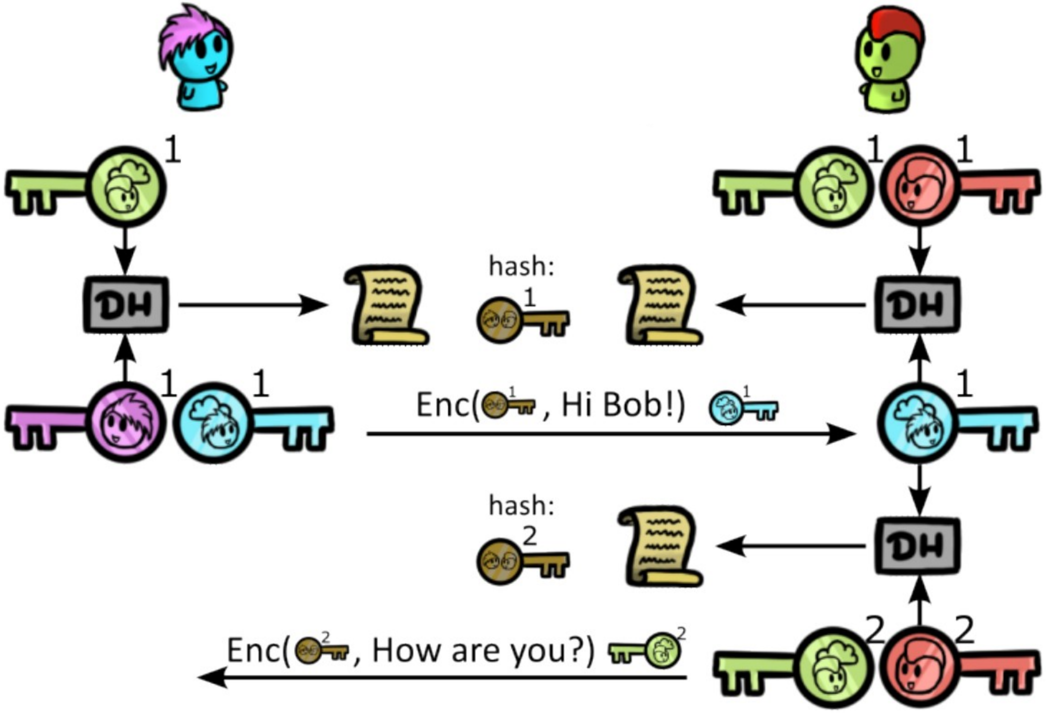
Forward Secrecy in OTR



Forward Secrecy in OTR

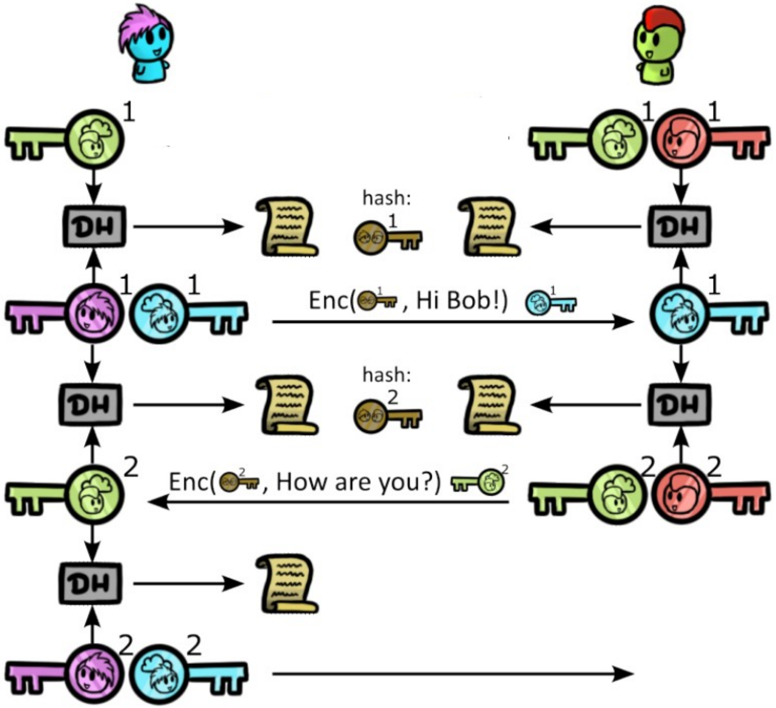


Forward Secrecy in OTR



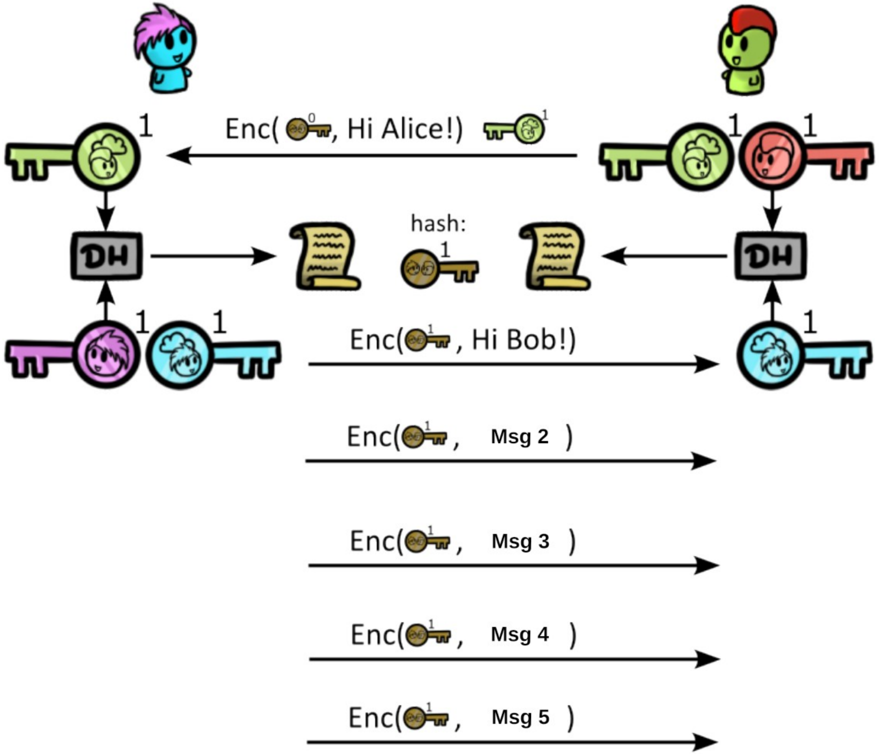
Forward Secrecy in OTR

- 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



Forward Secrecy in OTR

- One problem...
  - Session keys only roll forward when sender changes
  - What if Alice sends Bob many messages in a row?
  - (We'll see Signal improve upon this later)



Deniable Authentication in OTR

- PGP uses signatures for authentication...
- ...but they also provide non-repudiability
- How can we get authentication without non-repudiability?

Deniable Authentication in OTR

- PGP uses signatures for authentication...
- ...but they also provide non-repudiability
- How can we get authentication without non-repudiability?
- With a MAC!
  - Alice and Bob similarly negotiate DH authentication key

Recall...

- Why are MACs deniable?
- Only Alice and Bob know K
- Alice sends Bob a message MACed with K
- Bob knows it was Alice because:
  - Only Alice or Bob could have produced this MAC
  - Bob did not produce the MAC
- Why doesn't this argument work for Carol?

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.

Double Ratchet Algorithm

- Uses two ratchets:
  - KDF chain
  - Diffie-Hellman sessions (like OTR)
- Originally called Axolotl ratchet for its "self-healing" property (from the DH ratchet)

Photo: th1098

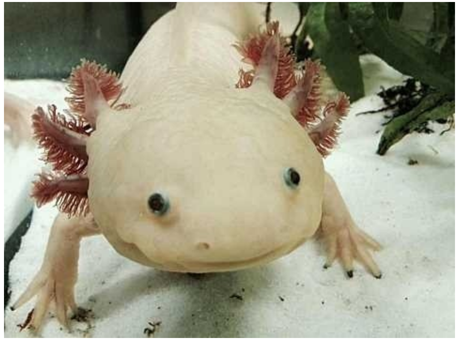


Illustration: ArmandoAre1

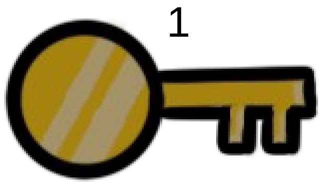
"Axolotl" is a Nahuatl word. (pronunciation)

Forward Secrecy (another approach)

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

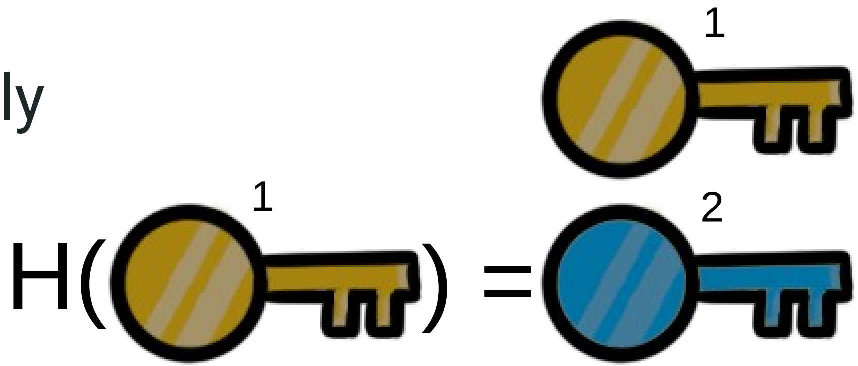
Forward Secrecy (another approach)

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



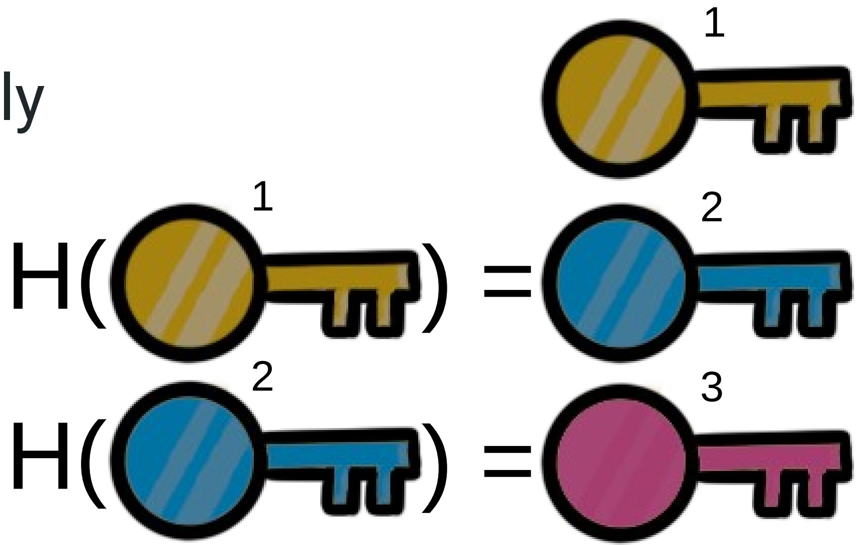
Forward Secrecy (another approach)

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



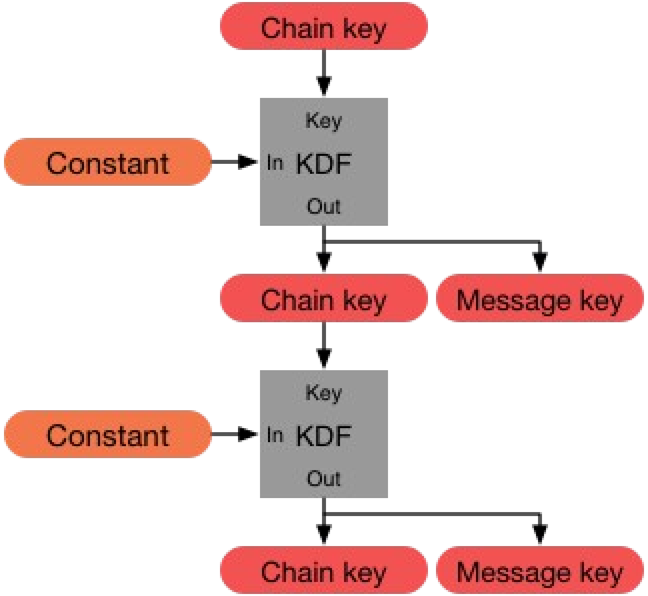
Forward Secrecy (another approach)

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



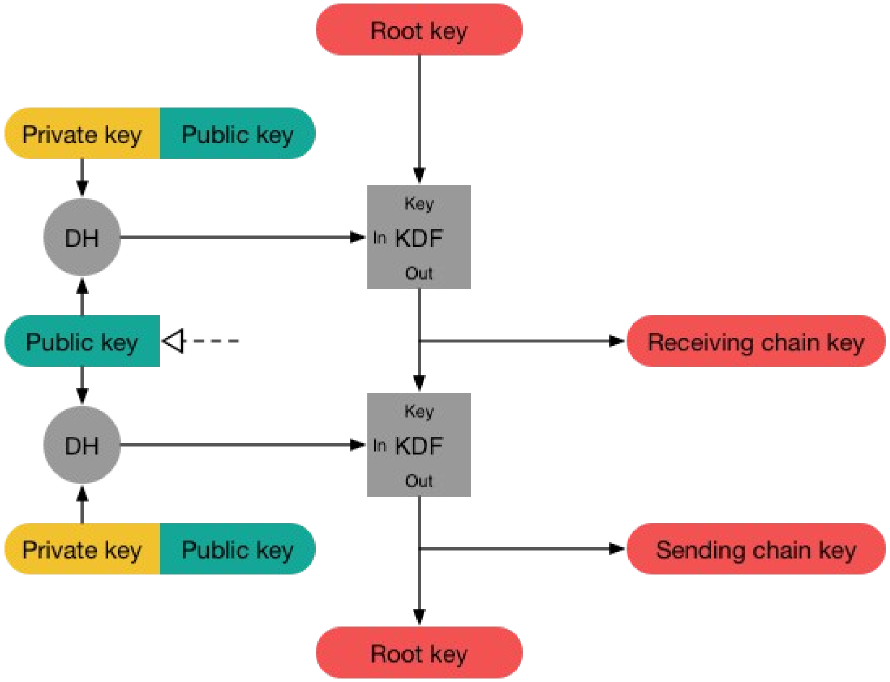
KDF Ratchet

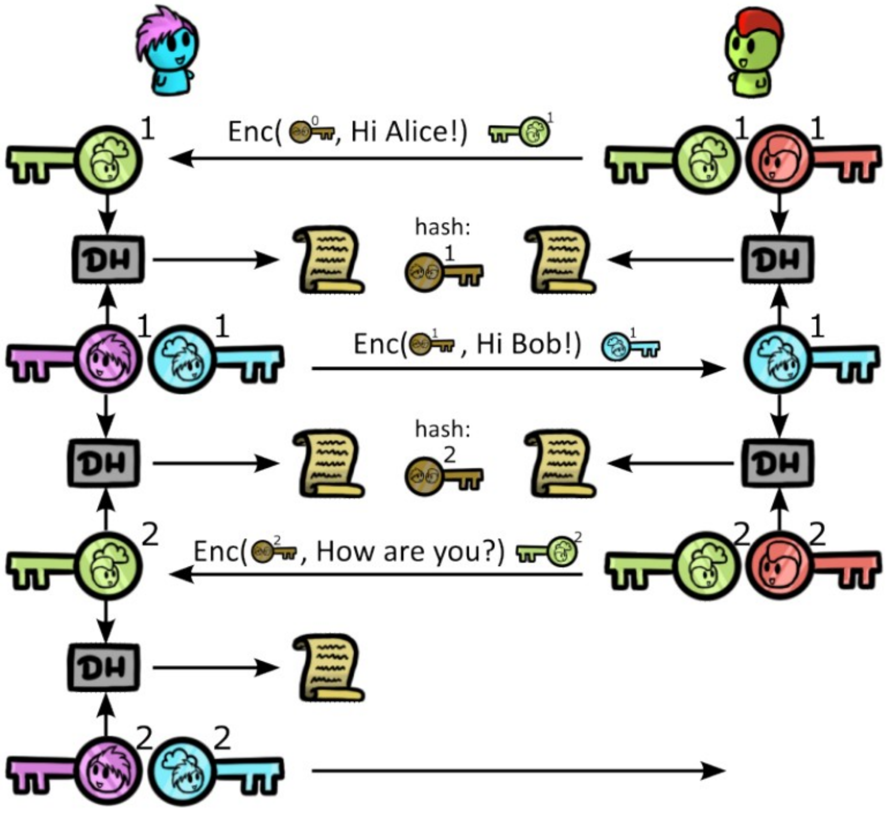
- KDF = Key Derivation Function (think hashing – it only goes one way)
- Outputs message key
  - **Used to encrypt a single message**
- Outputs chain key
  - Used to derive future keys
- Why separate chain & message keys?
  - What if messages are out-of-order?

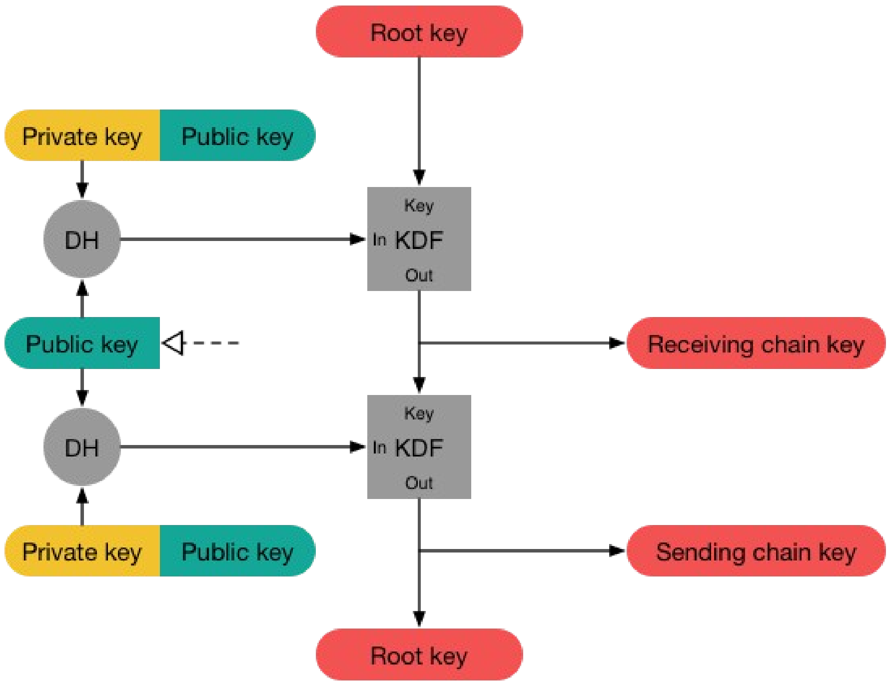


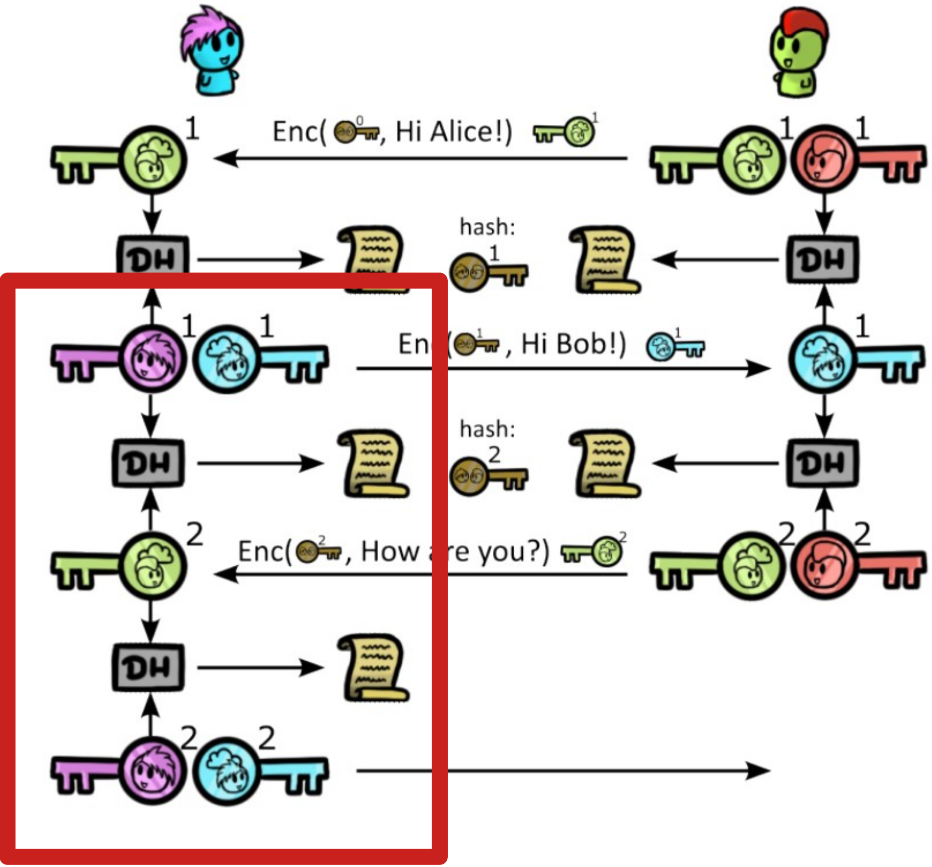
DH Ratchet

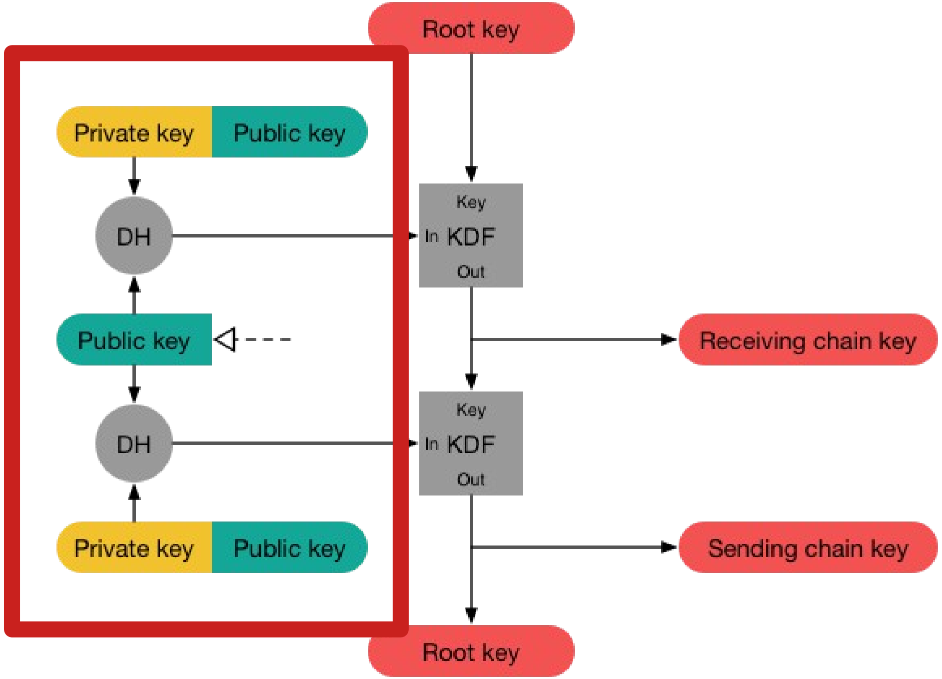
- Like OTR
- Outputs Receiving and Sending chain keys
  - These are used for KDF ratchet (previous slide)




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


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


1
Enc(0, Hi Alice!) 1
1 1
hash: 1
1 1
Enc(1, Hi Bob!) 1
1
hash: 2
2
Enc(2, How are you?) 2
2 2
2 2


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

Brace Yourselves!!!

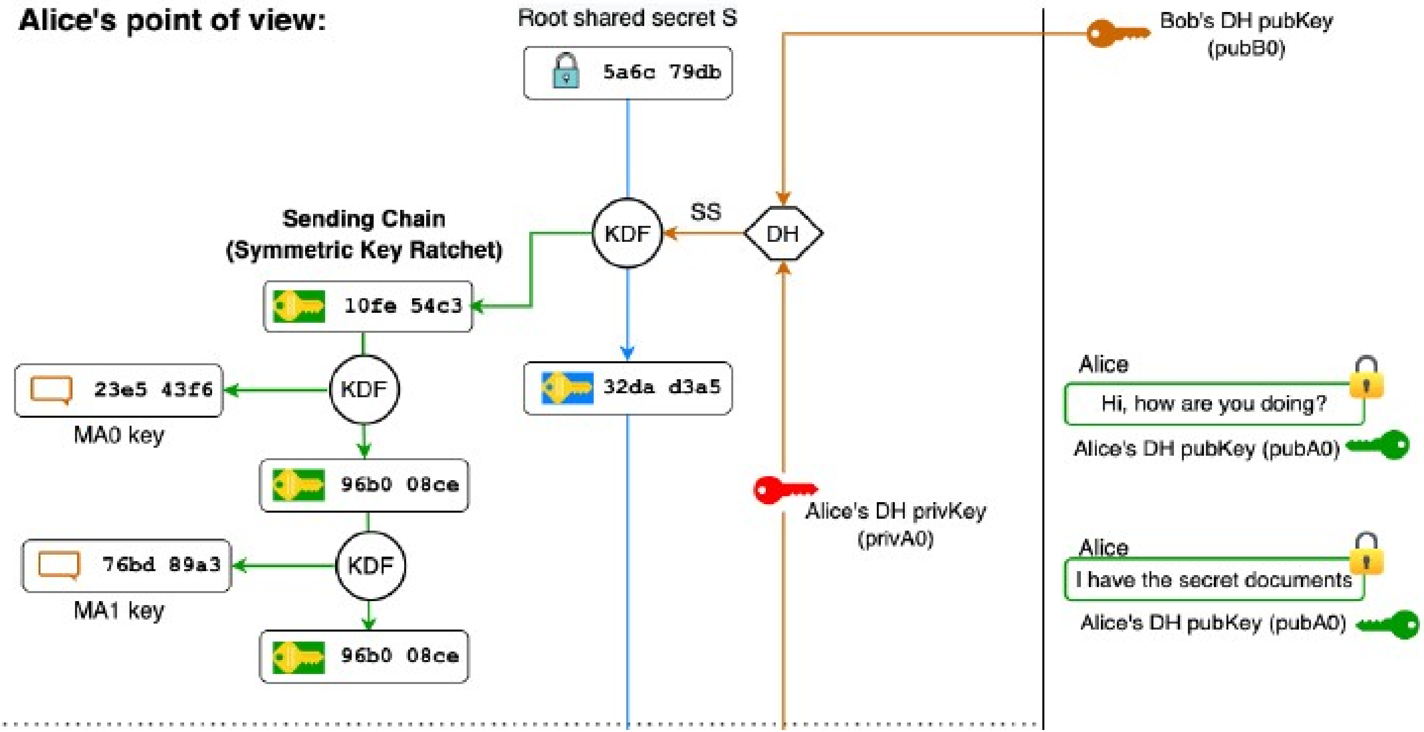
- We're about to put the two ratchets together
- It's going to be complicated
- But it will be okay

Photo: ZeWrestler

Photo: David J. Stang

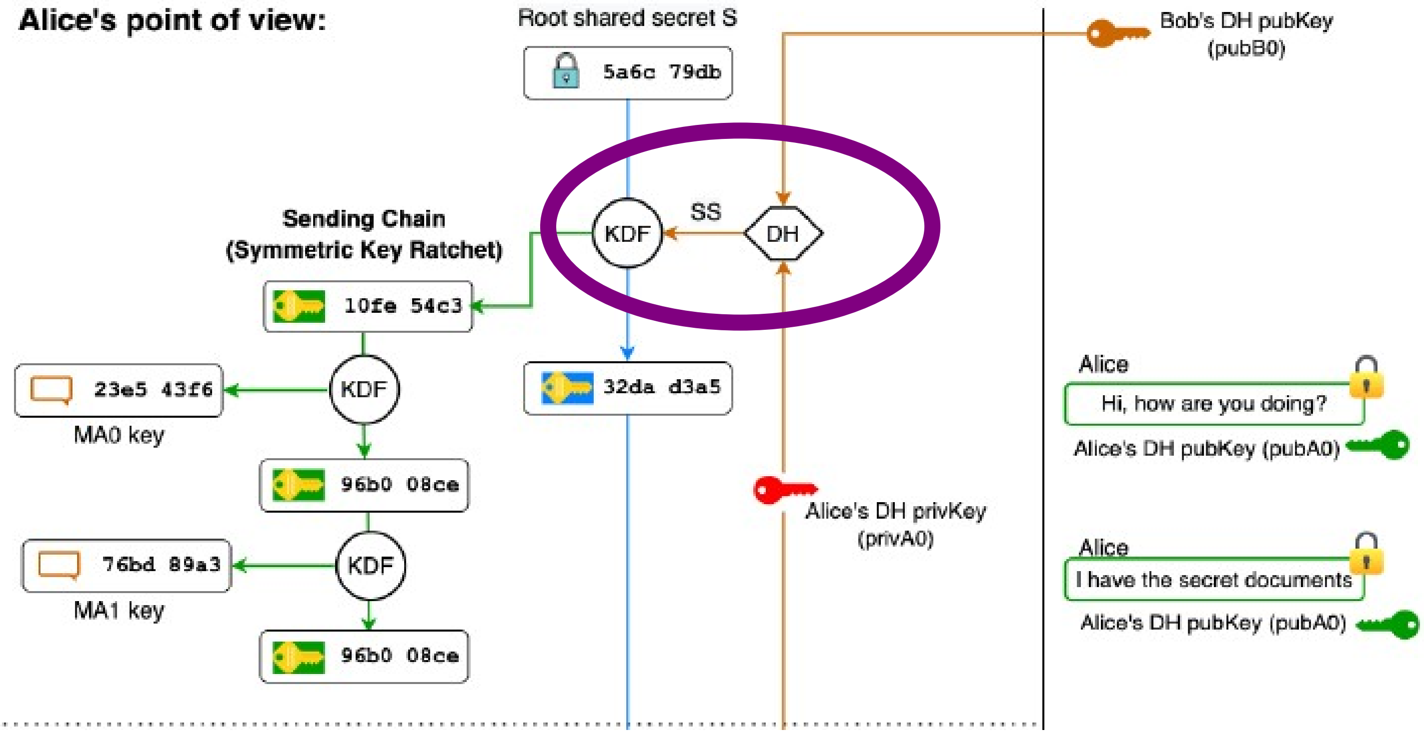
Double Ratchet Algorithm

- 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 for Bob



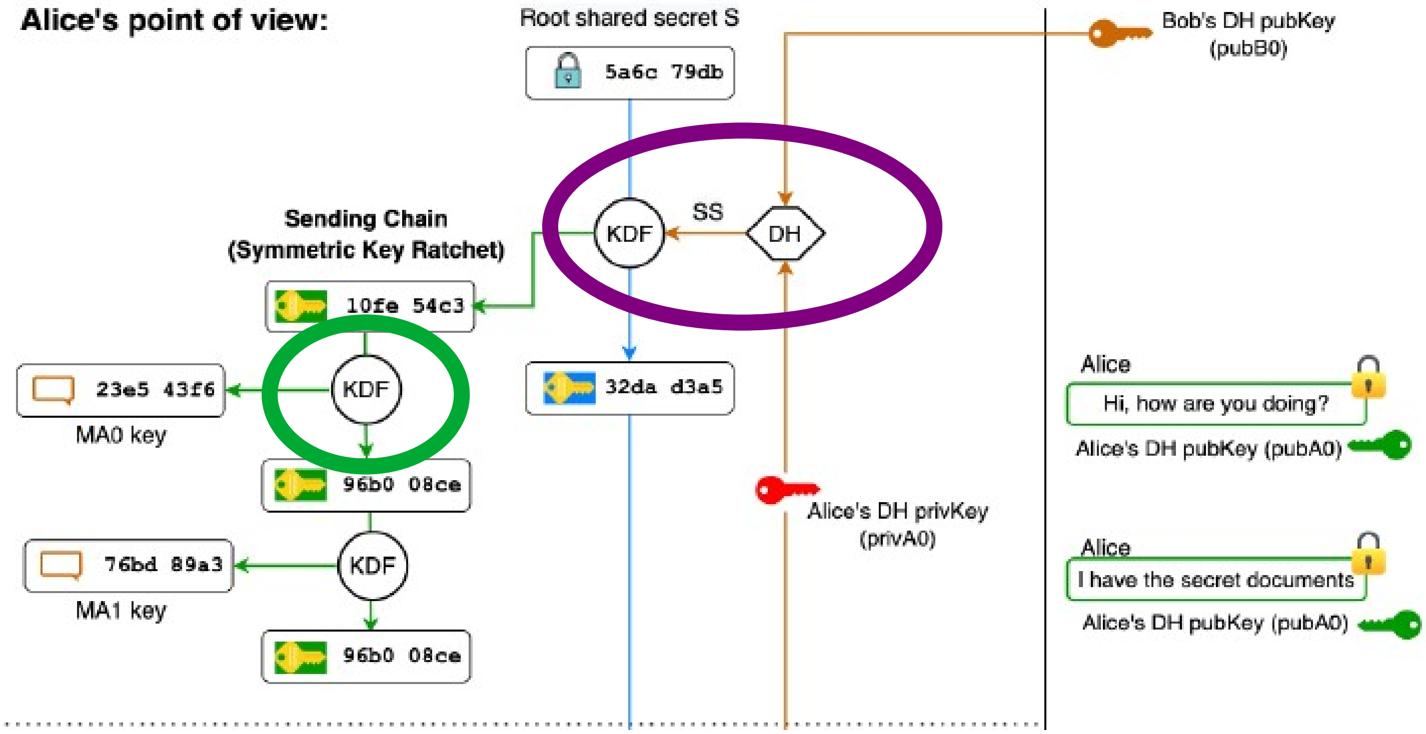
Double Ratchet Algorithm

- 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 for Bob



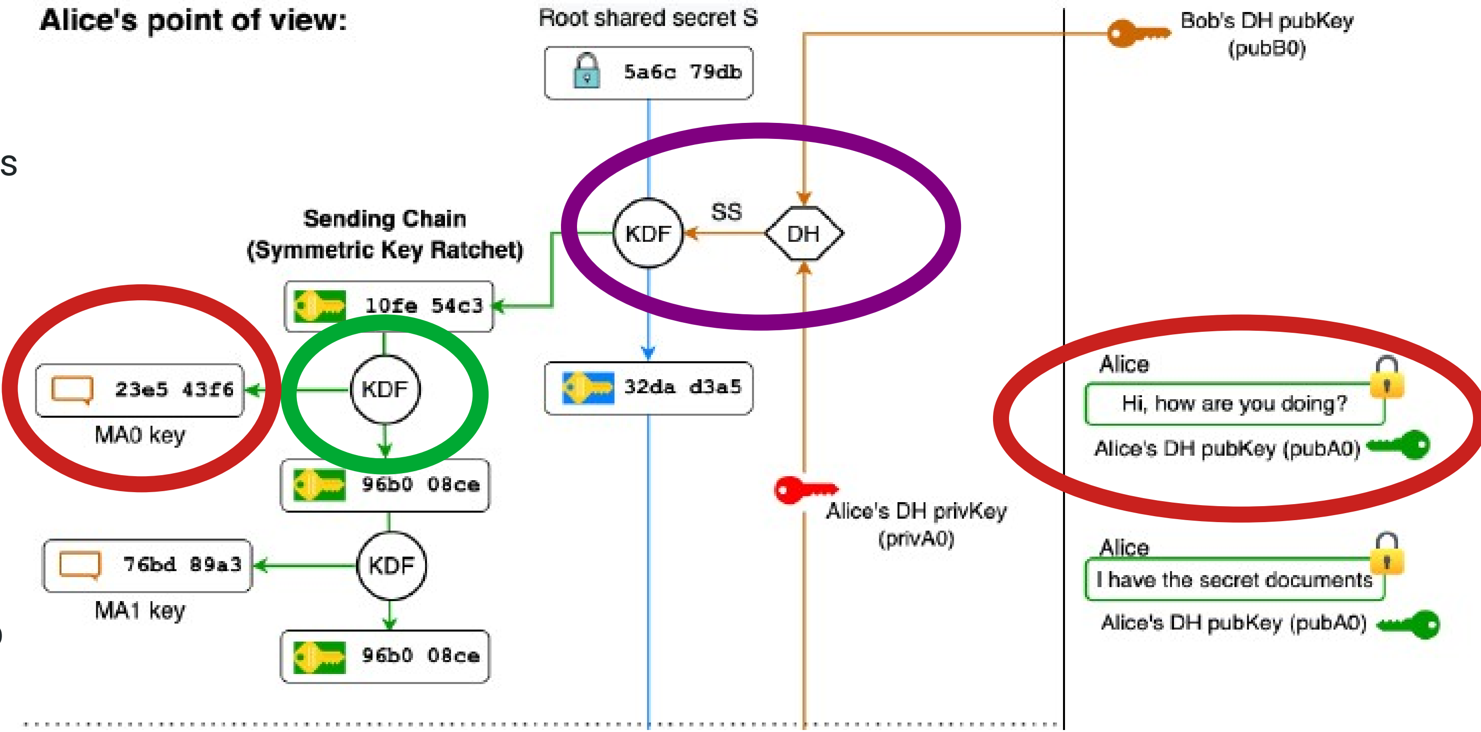
Double Ratchet Algorithm

- 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 for Bob



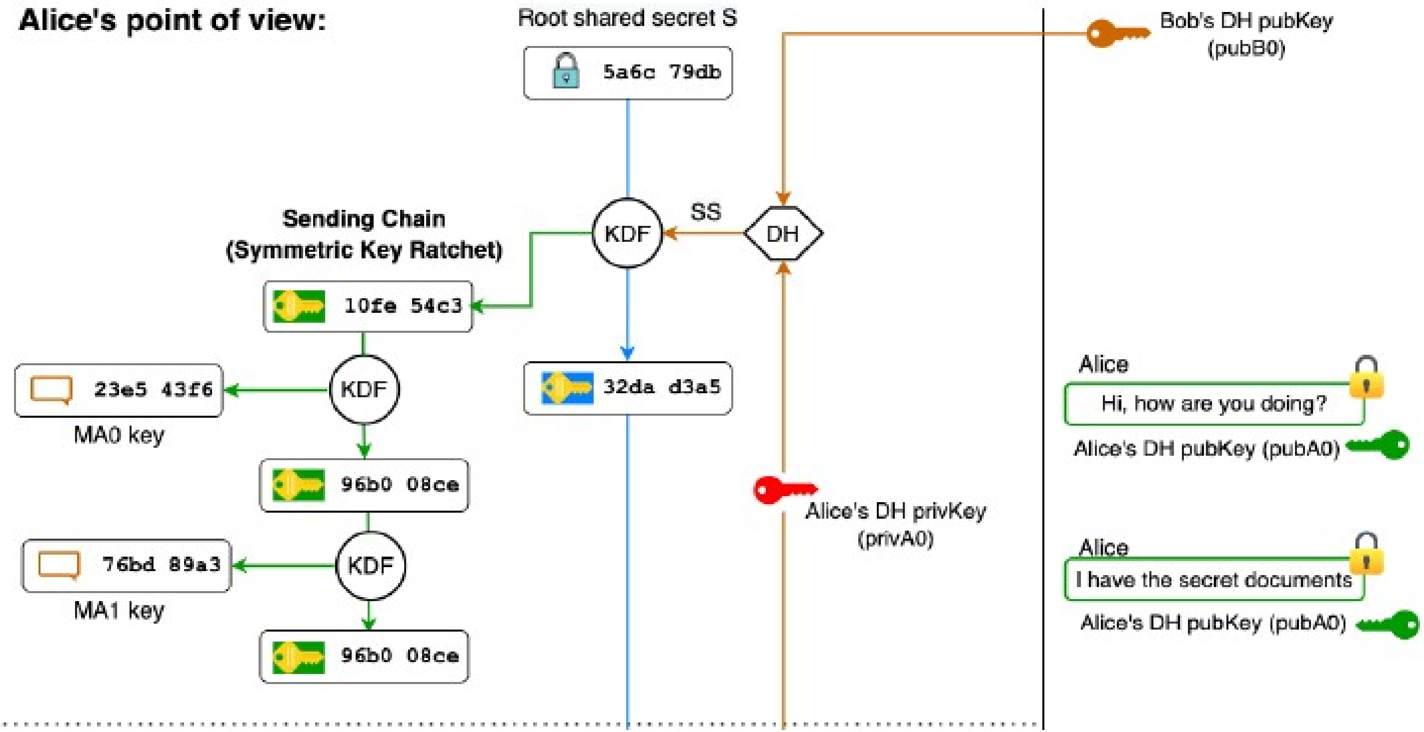
Double Ratchet Algorithm

- 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 for Bob



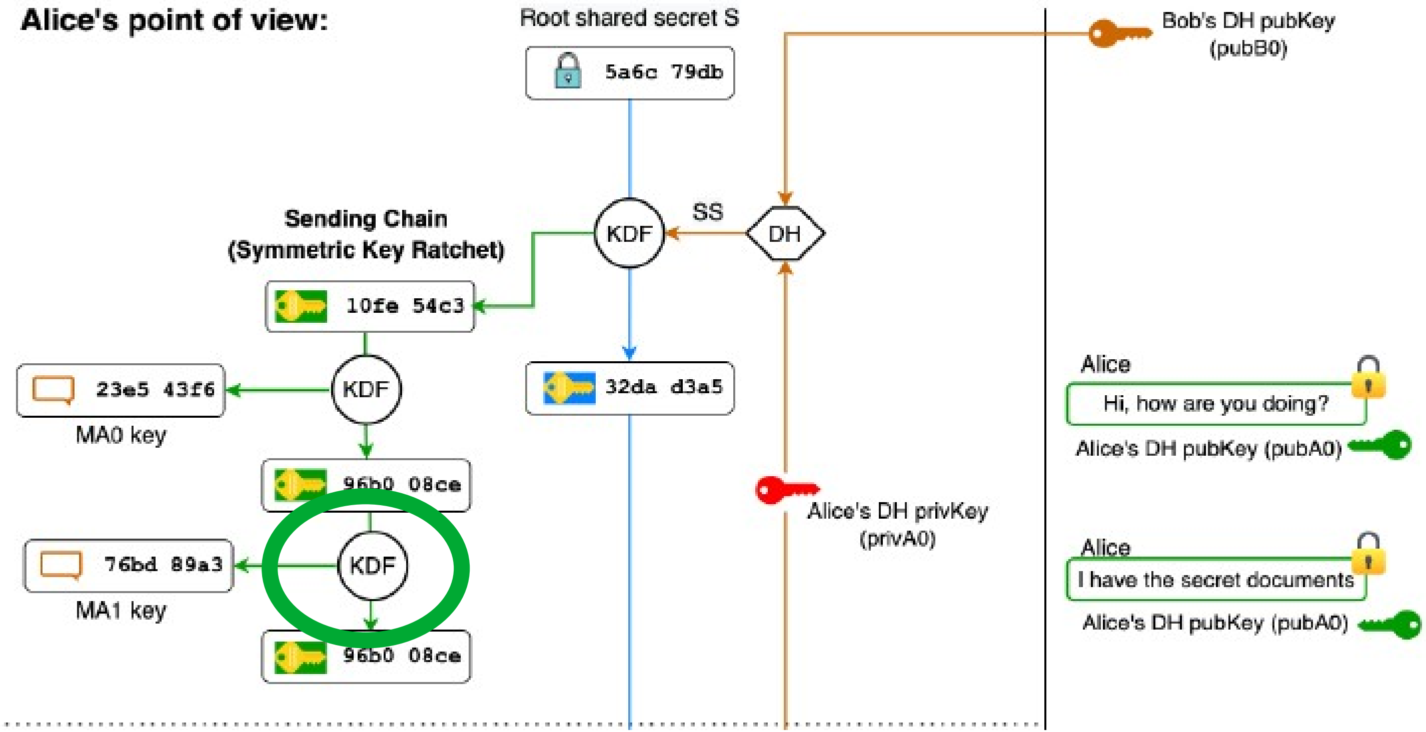
Double Ratchet Algorithm

- Alice -> Bob (again)
- (no new DH until Bob replies)
- Alice **derives a key** with her sending chain
- Alice uses this **MA1** key to encrypt her message for Bob



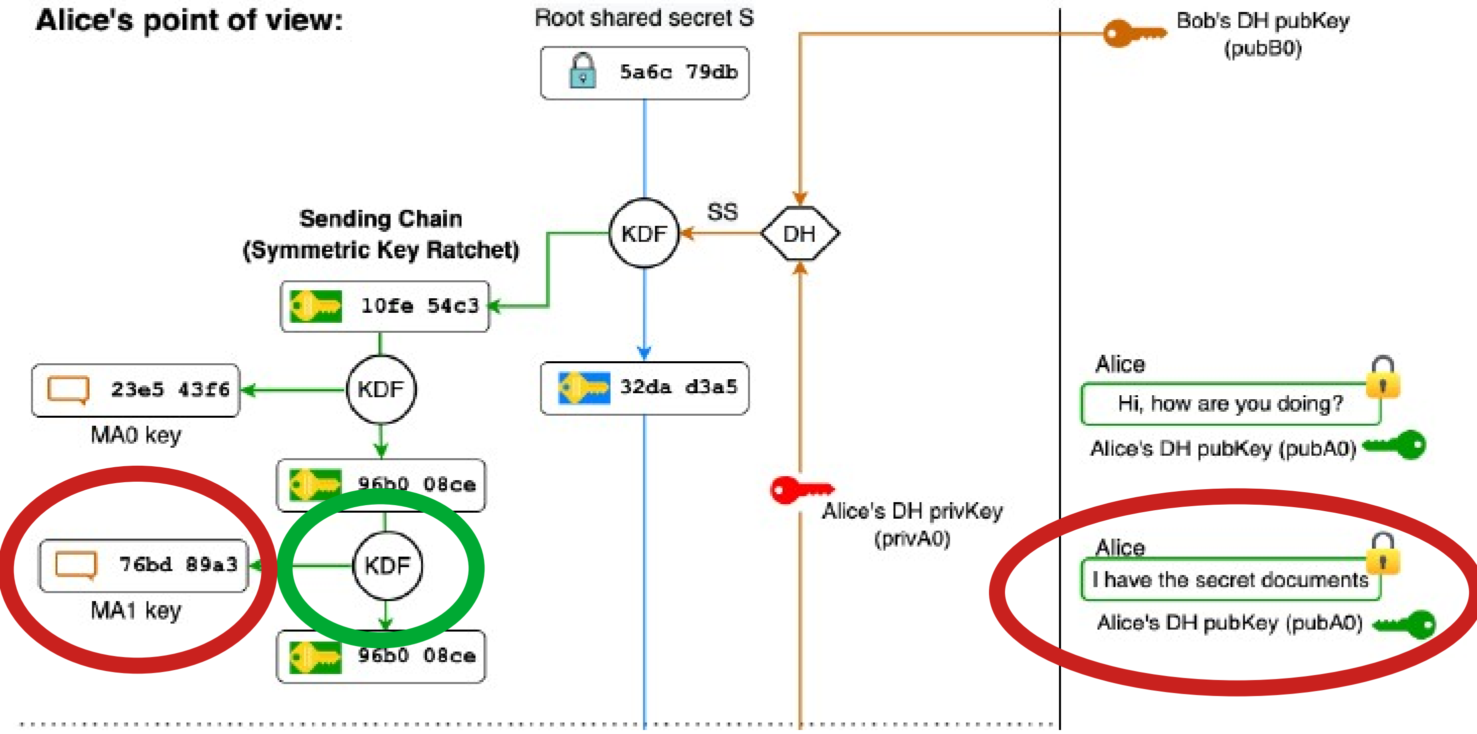
Double Ratchet Algorithm

- Alice -> Bob (again)
- (no new DH until Bob replies)
- Alice **derives a key** with her sending chain
- Alice uses this **MA1** key to encrypt her message for Bob

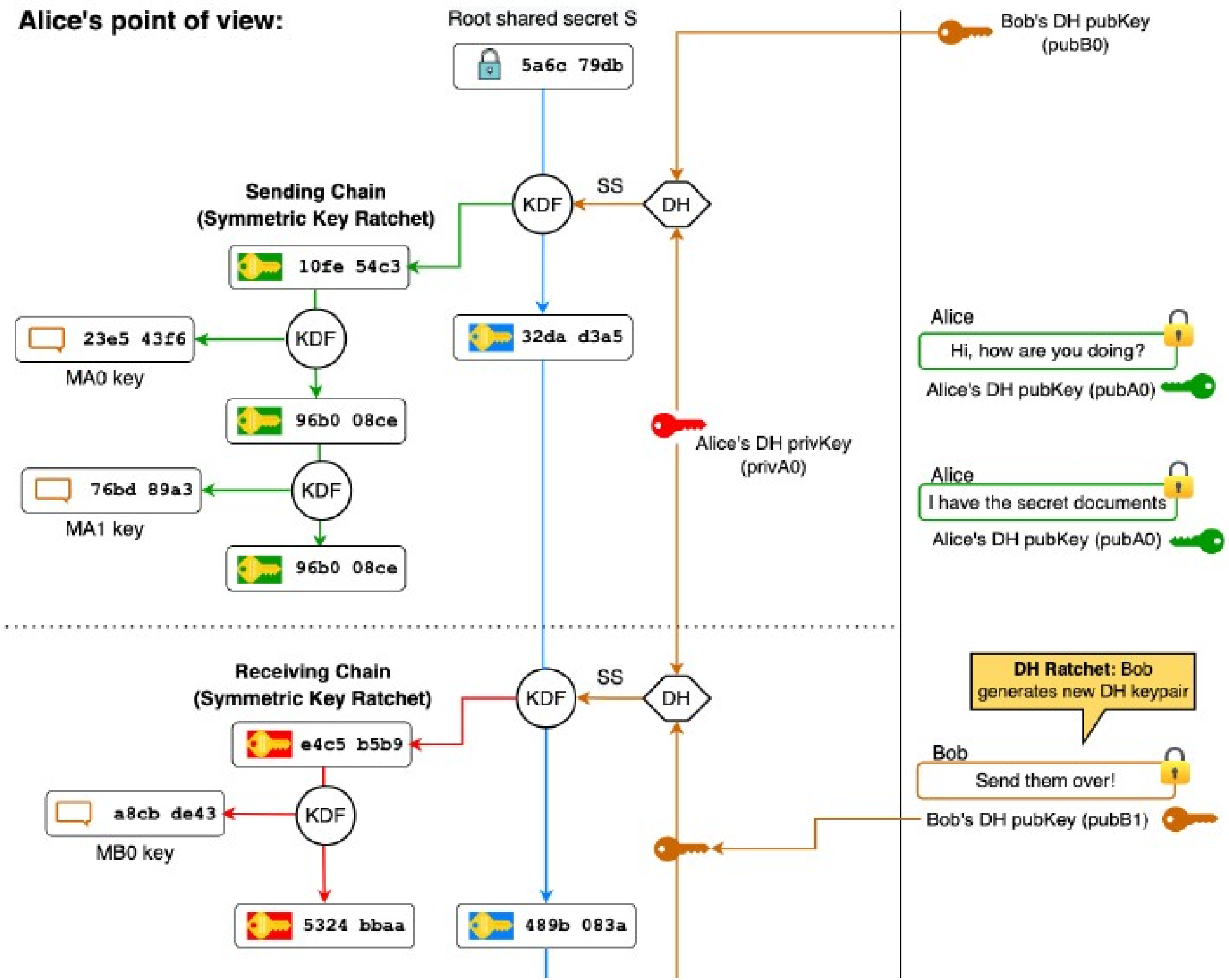


Double Ratchet Algorithm

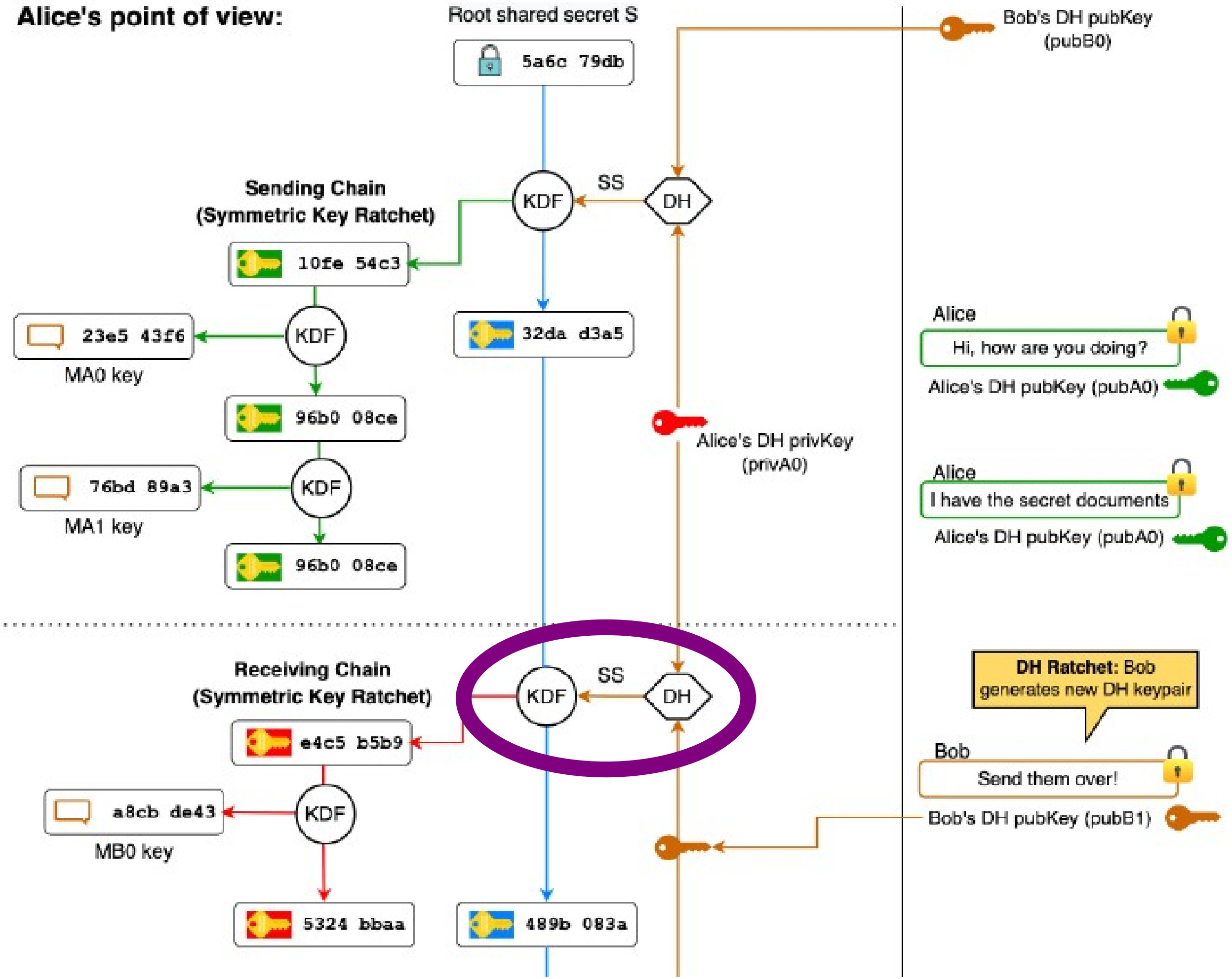
- Alice -> Bob (again)
- (no new DH until Bob replies)
- Alice **derives a key** with her sending chain
- Alice uses this **MA1** key to encrypt her message for Bob



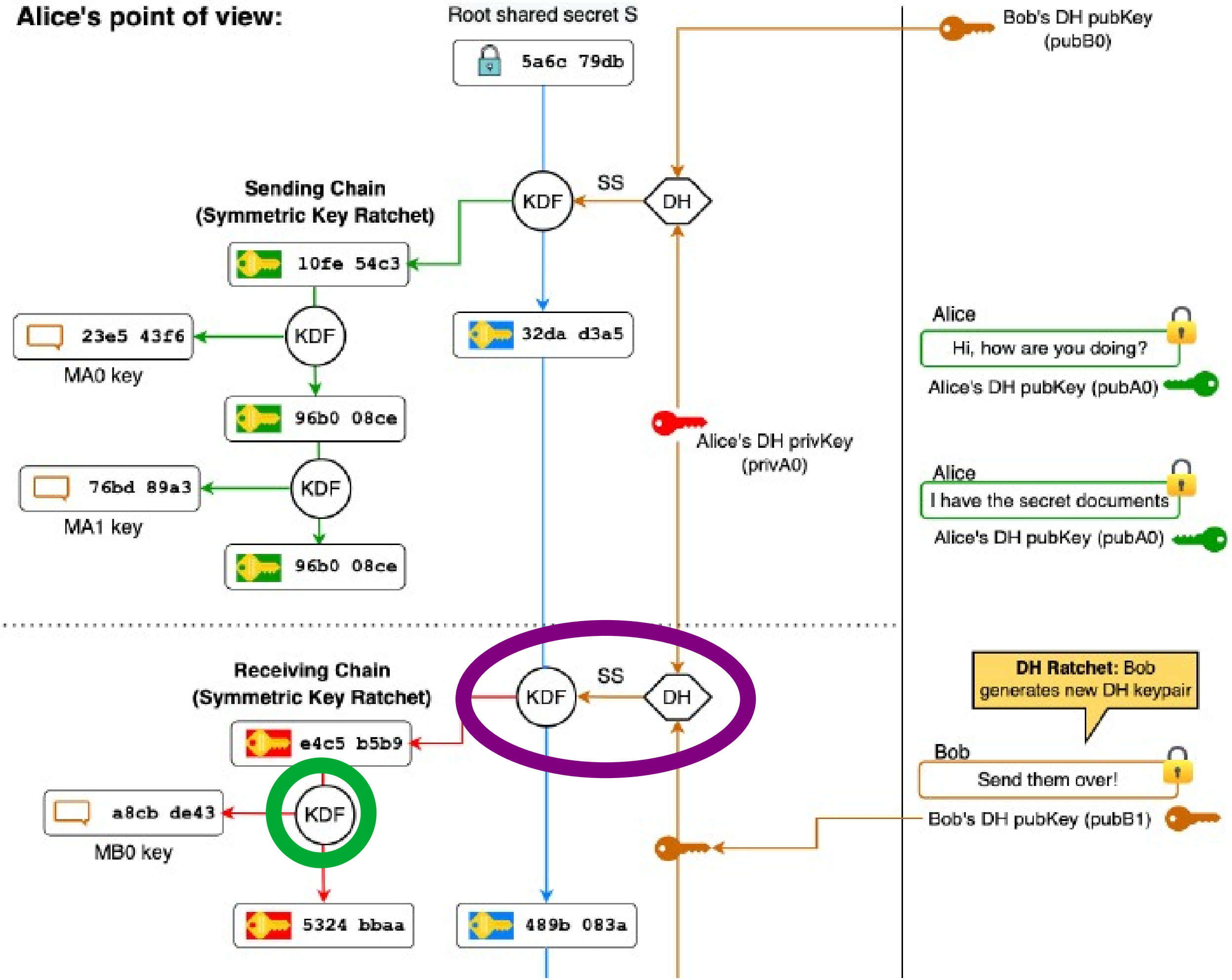
- Bob -> Alice
- Alice and Bob do **DH** and get Alice's receiving chain/Bob's sending chain
- Alice **derives a key** with her receiving chain
- Alice uses this **MB0** key to decrypt her message from Bob



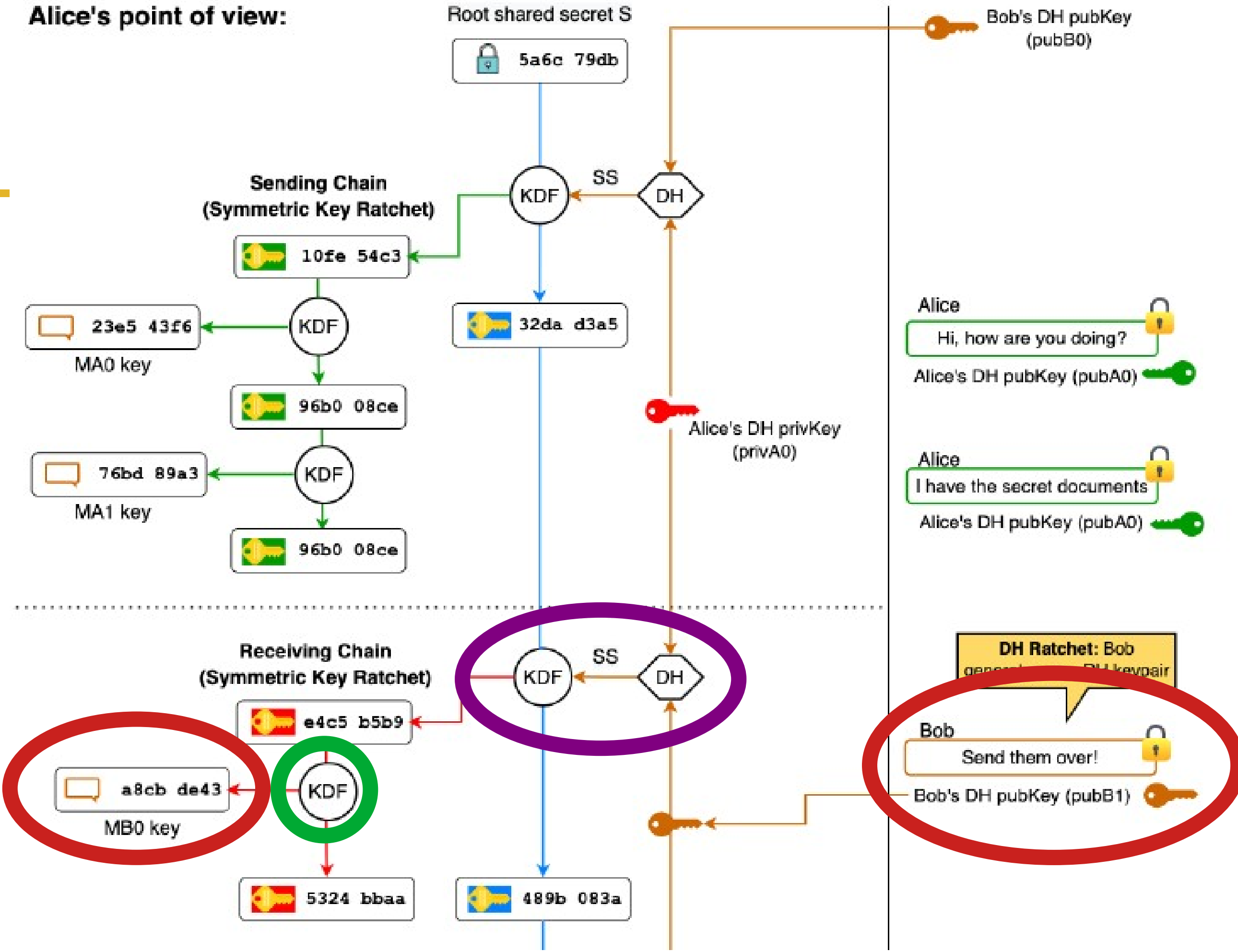
- Bob -> Alice
- Alice and Bob do **DH** and get Alice's receiving chain/Bob's sending chain
- Alice **derives a key** with her receiving chain
- Alice uses this **MB0** key to decrypt her message from Bob



- 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 this **MB0** key to decrypt her message from Bob



- 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 this **MB0** key to decrypt her message from Bob



Let’s take a breath

- Here are some more pictures of axolotls

Photo: LeDameBucolique

Photo: uthlas

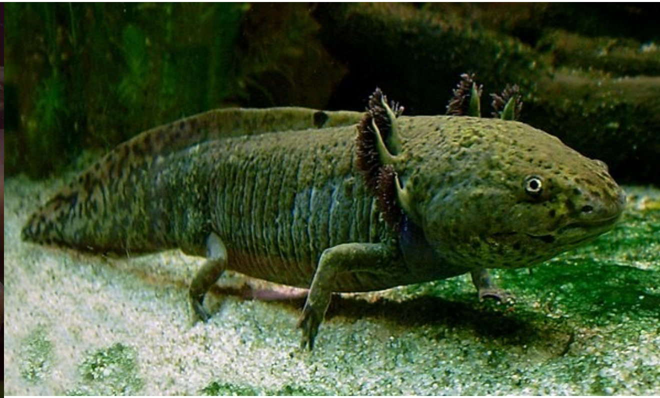


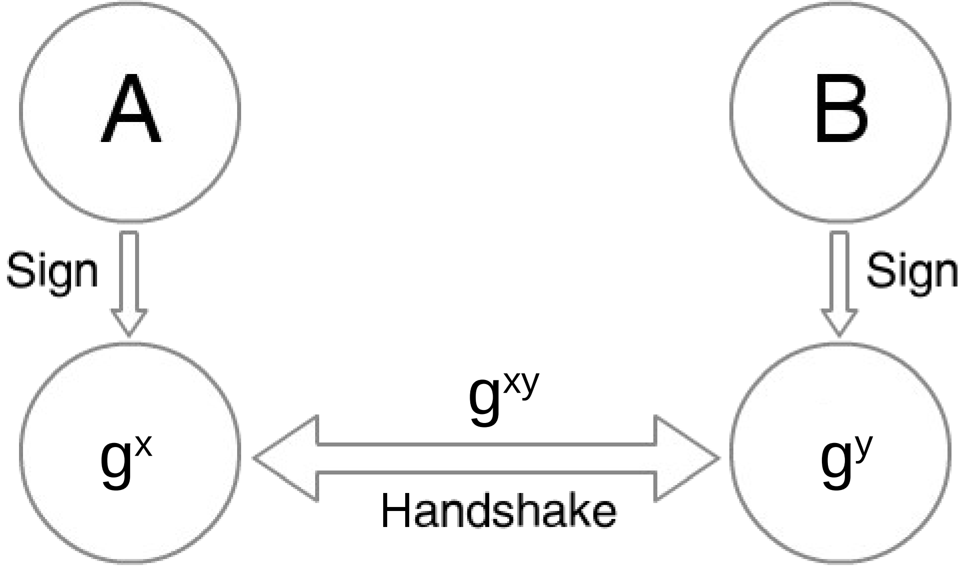
Photo: LoKiLeCh

Deniability in Signal

- Alice and Bob use MACs (like in OTR)
- But what if they can make it even more deniable?

Deniability in OTR

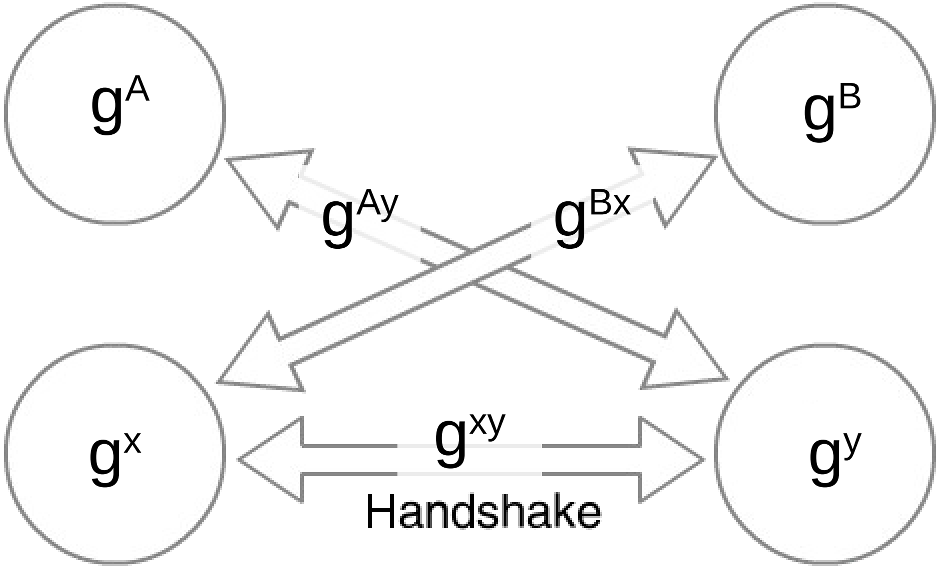
- DH(a,b) can only be created by Alice or Bob
  - A: long-term (Alice)
  - B: long-term (Bob)
  - x: ephemeral (Alice)
  - y: ephemeral (Bob)



Deniability in Signal: 3DH

- DH(A,y) || DH(x,B) || DH(x,y) can be created by anyone
- But if Alice knows x, only Bob could know y
- Why?

https://signal.org/blog/simplifying-otr-deniability/



That’s more theoretical

- Signal actually uses a more complicated eXtended Triple Diffie-Hellman (X3DH) key agreement protocol which involves some signatures
- X3DH is useful for enabling asynchronous communication
  - More mobile-friendly
- We won’t talk about it, but it’s well-documented here: https://signal.org/docs/specifications/x3dh/

Quick Recap

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