Signal E2E-Crypto

Why Can't I Hold All These Ratchets

oxzi

23.03.2021

In the next 30 minutes there will be

- a rough introduction in end-to-end encrypted instant messaging,
- an overview of how Signal handles those E2E encryption,
- and finally a demo based on a WeeChat plugin.

Historical Background

- Signal has not reinvented the wheel - and this is a good thing!
- Goes back to Off-the-Record Communication (OTR)[1].

OTR Features

- Perfect forward secrecy
- Deniable authentication

[1] Borisov, Goldberg, and Brewer. "Off-the-record communication, or, why not to use PGP", 2004

Influence and Evolution

- OTR influenced the *Signal Protocol*, Double Ratchet.
- Double Ratchet influence OMEMO; supports many-to-many communication.
- Also influenced Olm, E2E encryption of the Matrix protocol.
- OTR itself was influenced by this, version four was introduced in 2018.

Double Ratchet

> *The Double Ratchet algorithm is used by two parties to exchange encrypted messages based on a shared secret key.*

The Double Ratchet algorithm[2] is essential in Signal's E2E crypto.

But first, some basics.

[2] Perrin, and Marlinspike. "The Double Ratchet Algorithm", 2016

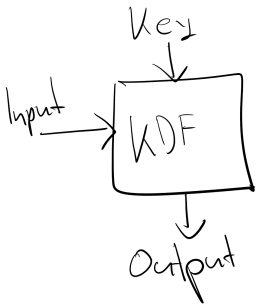
Cryptographic Ratchet

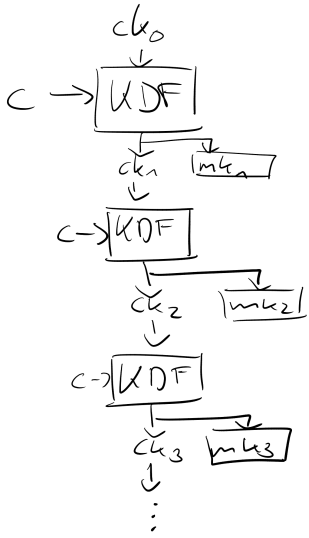
A ratchet is a *cryptographic* function that only moves forward. In other words, one cannot easily reverse its output.

Triple Ratchet, I guess.[3]

[3]By Salvatore Capalbi, https://www.flickr.com/photos/sheldonpax/411551322/, CC BY-SA 2.5

Symmetric-Key Ratchet





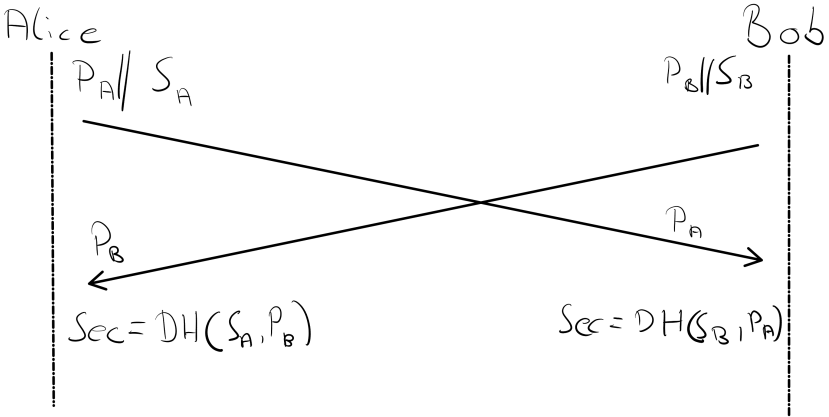
Symmetric-Key Ratchet

In everyday life, Keyed-Hash Message Authentication Code (HMAC) or HMAC-based KDFs (HKDF) are used.

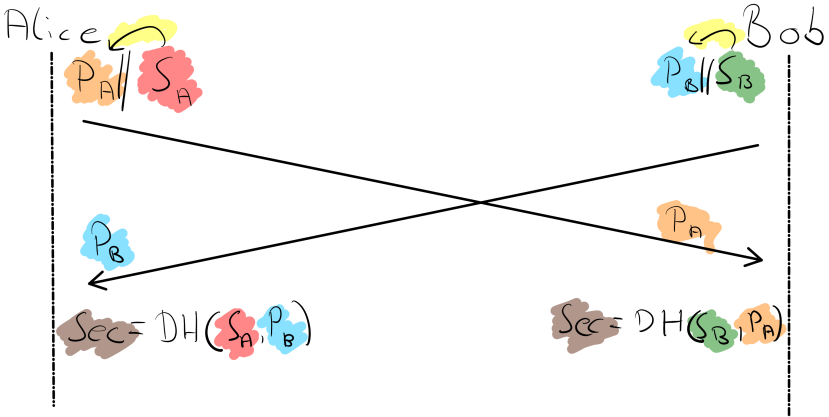
```
func ratchet(ckIn []byte) (ckOut, mk []byte) {
  kdf := hmac.New(sha256.New, ckIn)
  kdf.Write(c)  // publicly known constant c
  out := kdf.Sum(nil)
  return out[:32], out[32:]
}

ck0 := []byte{0x23, 0x42, ...}  // some initial shared secret
ck1, mk1 := ratchet(ck0)
ck2, mk2 := ratchet(ck1)
```

Diffie-Hellman Key Exchange



Diffie-Hellman Key Exchange



Diffie-Hellman Key Exchange

Originally, DH uses primitive residue classes modulo n. Nowadays, one might wanna use Elliptic-curve DH (ECDH).

```
secAlice := make([]byte, curve25519.ScalarSize)
_, _ = rand.Read(secAlice)
pubAlice, _ := curve25519.X25519(secAlice, curve25519.Basepoint)

// Alice receives Bob's public key
sharedSec := curve25519.X25519(secAlice, pubBob)
```

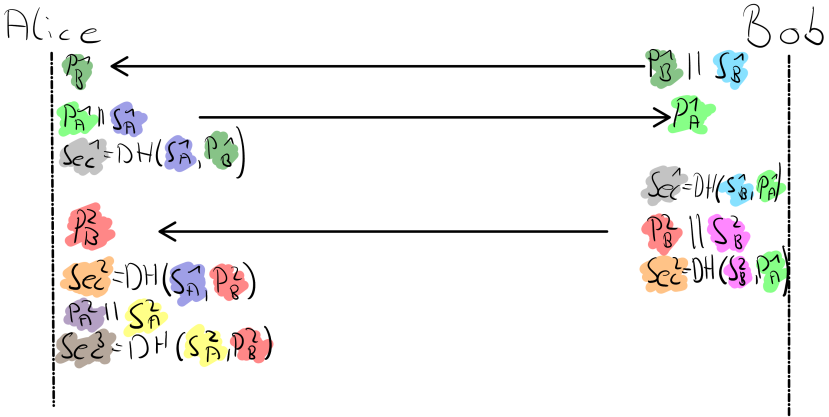
Diffie-Hellman Ratchet

Alice and Bob are performing nonstop DH key exchanges. The shared secret will be used as the initial shared secret for the symmetric-key ratchet.

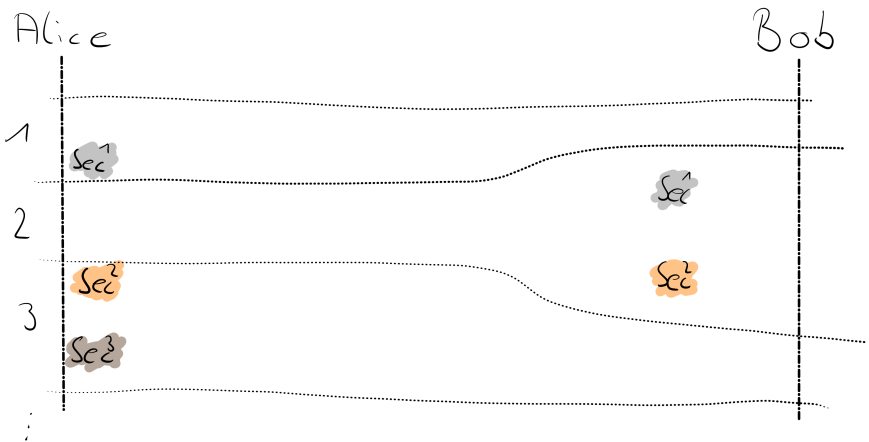
Diffie-Hellman Ratchet

Diffie-Hellman Ratchet

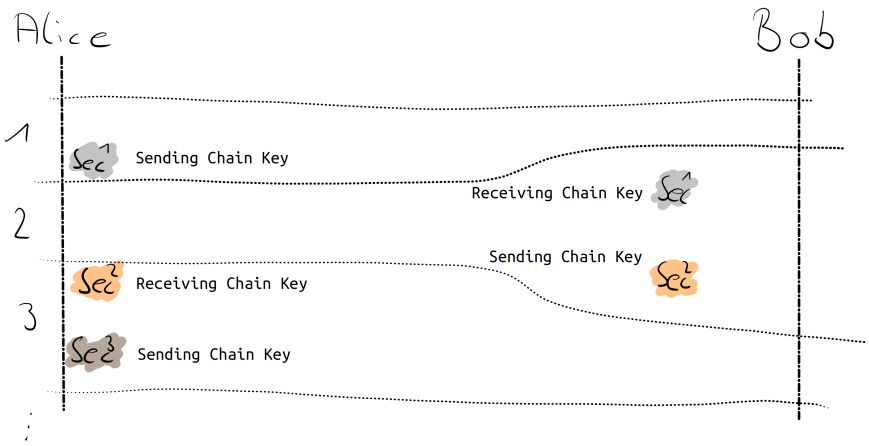
Diffie-Hellman Ratchet



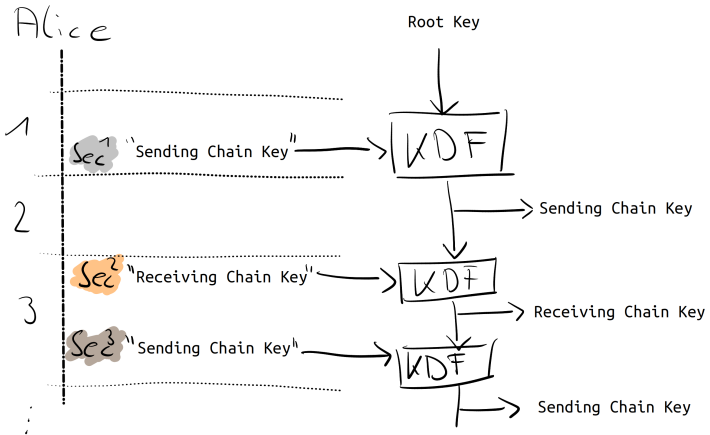
Diffie-Hellman Ratchet



Diffie-Hellman Ratchet



Diffie-Hellman Ratchet



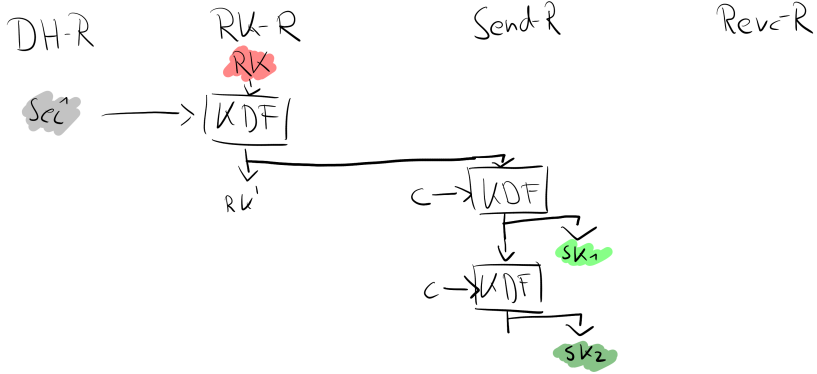
Diffie-Hellman Ratchet

- Likewise, Bob derives his chain keys from a KDF with the same initial root key.
- Due to the root key, different chain key are created in case of a MITM.
- At this point, there is no verification yet.

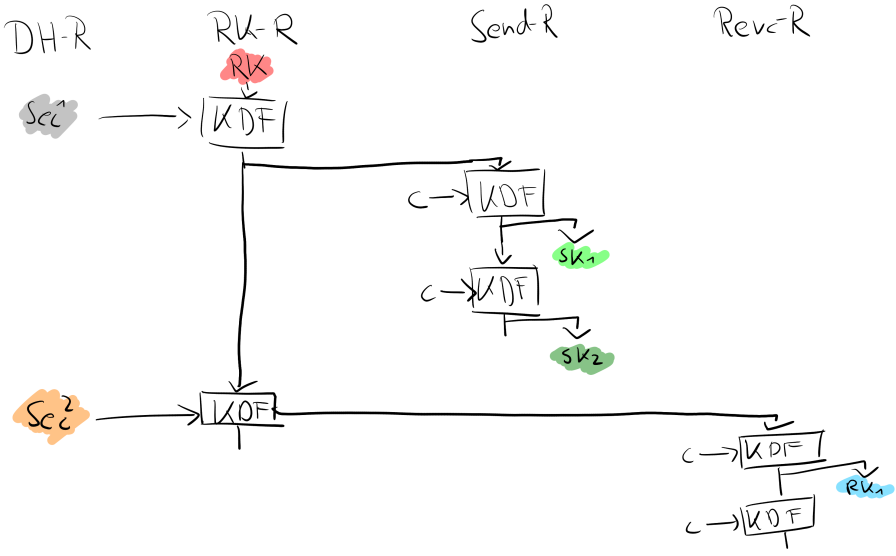
Double Ratchet

- Each message will be encrypted with an unique key.
- Double Ratchet combines a DH ratchet and two symmetric ratchets.
- One symmetric ratchet for sending keys and one for receiving keys.
- Alice's sending ratchet is Bob's receiving ratchet, et vice versa.
- Each DH ratchet step creates a new symmetric ratchet.

Double Ratchet



Double Ratchet



Double Ratchet

Each encrypted message is prefixed with the sender's current public DH key. Thus, the recipient knows if a DH ratchet is necessary.

- -> $P_A^1, ENCRYPT(SK_1, \text{"hello bob"})$
- -> $P_A^1, ENCRYPT(SK_2, \text{"how are you"})$
- <- $P_B^1, DECRYPT(RK_1, CIPHERTEXT_1)$
- Alice receives a new DH public key from Bob $\rightarrow$ Alice performs a DH Ratchet step:
  - New Receiving Chain based on the first DH secret.
  - New Sending Chain based on the second DH secret.
- -> $P_A^2, ENCRYPT(SK_1, \text{"blub blub blah"})$

Double Ratchet

The *ENCRYPT* / *DECRYPT* function must perform an authenticated encryption with associated data (AEAD).

- On encryption, a MAC is attached to the cipher text. It is based on the sending chain key.
- On decryption, this MAC is verified first, based on the receiving chain. In case of a MITM during the DH Ratchet, the chain key differs due to the root key. Thus, a tempered session will be detected.

Big Picture

Both the root key and some associated data needs to be known by both parties.

Therefore, an initial DH key exchange can be used. Signal has defined its own Extended Triple Diffie-Hellman (X3DH) Key Agreement Protocol[4].

[4] Marlinspike, and Perrin. "The X3DH Key Agreement Protocol", 2016

Demo Time

Demo of WeeChat Xochimilco[5] based on xochimilco[6], an implementation of X3DH and Double Ratchet.

[5] https://github.com/oxzi/weechat-xochimilco
[6] https://github.com/oxzi/xochimilco