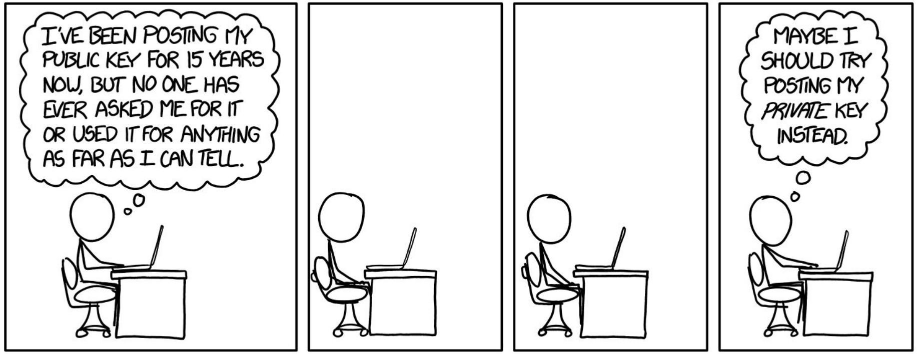
The double-ratchet algorithm: its security and privacy properties

Sofía Celi

In the beginning…

Why OTR was created?

- Paper in 2004 by *Ian Goldberg*, *Nikita Borisov* and *Eric Brewer*
- Conversations in the "digital" world should mimic casual real world conversations
- PGP: protect communications. Sign messages and encrypt them.
- Problems: there is a record, there is a ‘proof’ of authorship



https://xkcd.com/1553/

Let’s start with properties

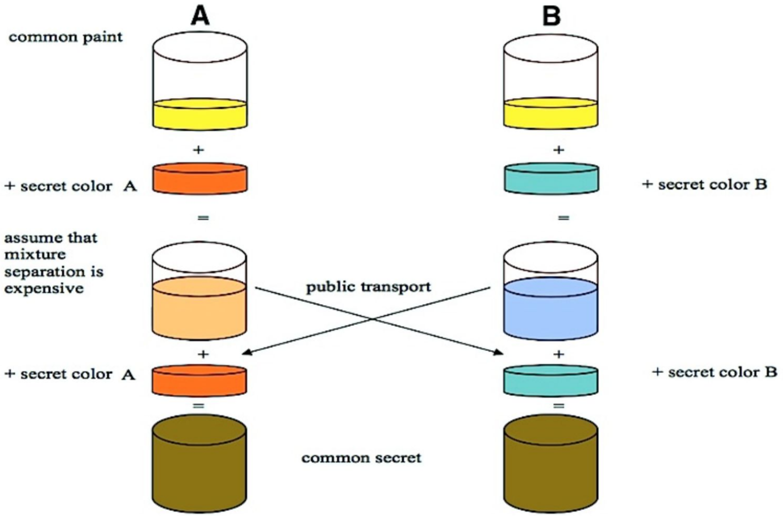
- Forward secrecy:
  - Usage of unique keys for the encryption of each message

  - “The idea of perfect forward secrecy (sometimes called break-backward protection) is that previous traffic is locked securely in the past.” (Menezes, A., Oorschot, P., Vanstone, S. (1997), *Handbook of Applied Cryptography*, CRC Pres.)
  - “A classical adversary that compromises the long-term secret keys of both parties cannot retroactively compromise past session keys” (Bellare, M., Pointcheval, D., & Rogaway, P. (2000). *Authenticated Key Exchange Secure Against Dictionary Attacks*. In Advances in Cryptology–EUROCRYPT)

- Usage of Diffie-Hellman key exchange:
    - Generate *a,* perform DH exchange
    - Use the shared secret *K ((g^b)^a)* to generate *MK*
    - Encrypt messages with *MK*
    - Forget *a* after key exchange; forget *MK* after session

- But there are problems with this…

  what about out-of-order messages?



- Post-compromise security (sometimes referred as backward secrecy):
  - Even if a message key gets compromised, no future messages can be decrypted
  - “A protocol between Alice and Bob provides Post-Compromise Security (PCS) if Alice has a security guarantee about communication with Bob, even if Bob’s secrets have already been compromised” (Cohn-Gordon, K., Cremers, C., & Garrat, L. (2016). *On Post-Compromise Security*. Department of Computer Science, University of Oxford)

Double Ratchet Algorithm

- Happens after an *AKE*
- Designed by Trevor Perrin and Moxie Marlinspike

Alice:

- Has a shared secret *K*
- Bob's public key: *bob_dh_pub_0*

Bob:

- Has a shared secret *K*
- Bob's private key: *bob_dh_priv_0*

- Generates:
    - alice_dh_priv_0, alice_dh_pub_0 = generateDH()
- Calculates:
    - shared_secret_1 = DH(alice_dh_priv_0, bob_dh_pub_0)

Alice:

- Derives:
  - RK_0, CKs_0 = KDF(K, shared_secret_1)
- Wants to send message 1 “Hello”
- Derives
  - CKs_1, MK_0 = KDF(CKs_0)
- Encrypts:
  - c_1 = ENC(MK_0, “Hello”)
- Sends: c_1 || alice_dh_pub_0

Bob:

- Calculates:
  - shared_secret_1 = (bob_dh_priv_0, alice_dh_pub_0)
- Derives:
  - RK_0, CKr_0 = KDF(K, shared_secret_1)
- Derives
  - CKr_1, MK_0 = KDF(CKr_0)
- Decrypts
  - “Hello” = DEC(MK_0, c_1)

- If, at that point, Bob wants to send messages, he:

- Generates:
  - bob_dh_priv_1, bob_dh_pub_1 = generateDH()
- Calculates:
  - shared_secret_1 = DH(bob_dh_priv_1, alice_dh_pub_1)

- Double-ratchet algorithm: “Ping-pong” mechanism
- Post-compromise in the sense of giving a timeframe (aka channel healing)
- Alwen, Coretti and Dodis: Immediate Decryption and Message-loss Resilience

Important to note

- Happens after an AKE: a shared secret should have been generated.
- Keys are ‘advertised’ on sent messages.
- There are many other values to keep track of for out-of-order
- The header can be encrypted

What it does not give...

- Authentication
- Deniability

To take into account

- Stored keys should be expired
- Needs secure deletion
- What happens if both participants initialize at the same time?
- Does not protect against device compromise

Why is it so used?

The state of the art

	OTRv3	OTRv4	Signal	OMEMO	Olm/Megolm	Telegram
Forward secrecy	Weak	Interactive: full Non- interactive: weak	Weak	Weak	None	Weak*
Post-compromise secrecy	Full	Full	Full	Full	Full	Full*
Online Deniability	○	● ◐	○	○	○	○
Offline Deniability	◐	●	◐	◐	●	●

● provides property

◐ partially provides property

○ does not provide property

Thanks!

Sofía Celi
@claucece