Solving the key exchange problem

Frank Braun

October 3, 2015

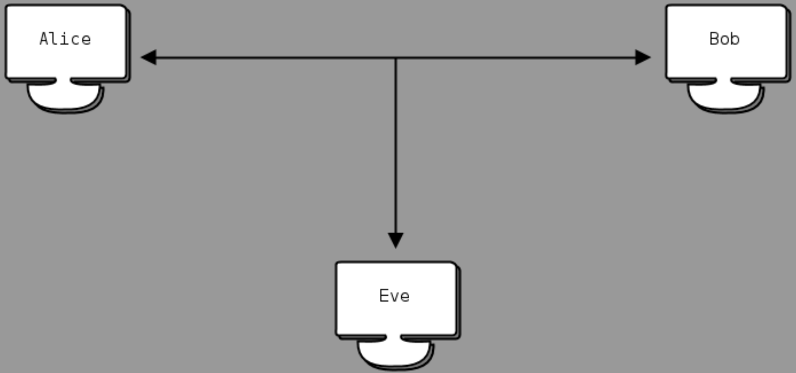
1 importance of the key exchange problem

2 previous attempts

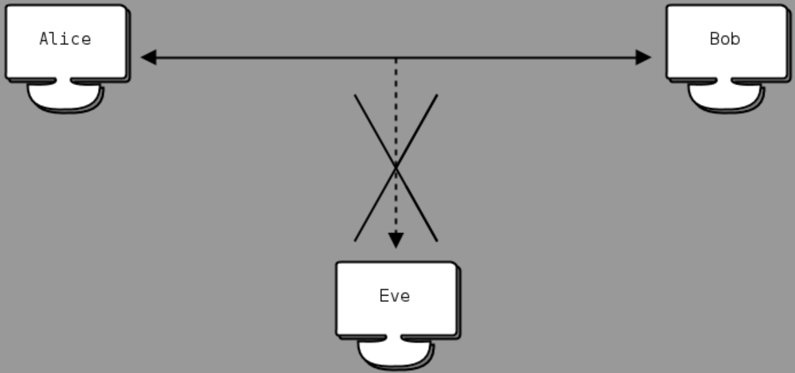
3 a new approach

4 conclusion

Alice and Bob have this thing going on...



...and they don't like Eve!

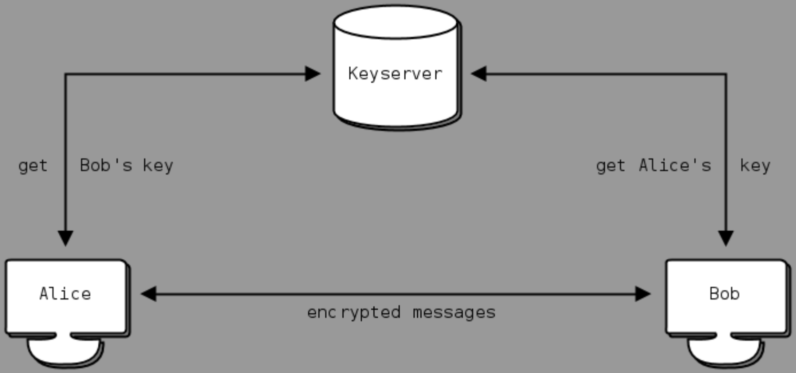


encryption: introduction

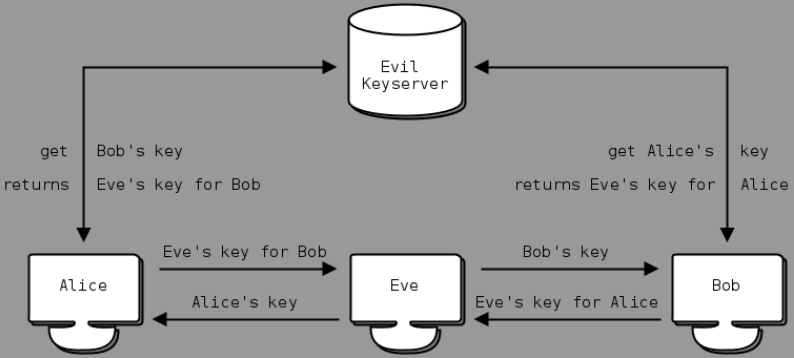
- "conventional" symmetric encryption uses one key for encryption and decryption (secure channel needed for key exchange)
- in contrast, public-key encryption is asymmetric and uses key pairs (a public and a private key)
- something encrypted for a given public key can only be decrypted by the corresponding private key
- the reverse operation is a digital signature: something encrypted (signed) by a private key can only be decrypted (verified) by the corresponding public key

⇒ public-key encryption solves the key exchange problem
⇒ public-key encryption ~~solves the~~ has a key exchange problem!
why is that?

keyserver: distributing public keys



man-in-the middle attack / evil keyserver

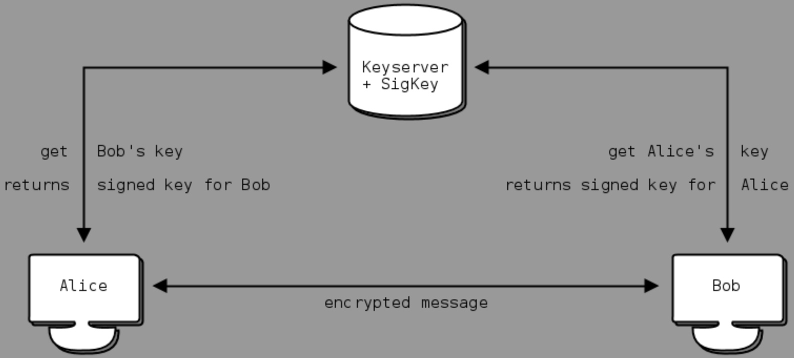


key exchange: harder than expected

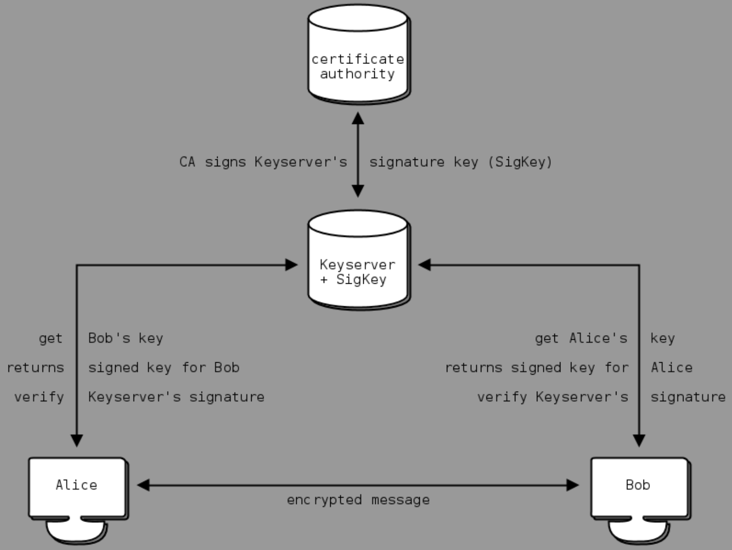
- during the development of public-key cryptography the key distribution / key exchange problem was considered a minor one
- **but**: after the complicated mathematics was solved the key exchange problem remained
- Crypto: How the Code Rebels Beat the Government Saving Privacy in the Digital Age, Steven Levy, **2001**

let's look at some previous attempts and their shortcomings...

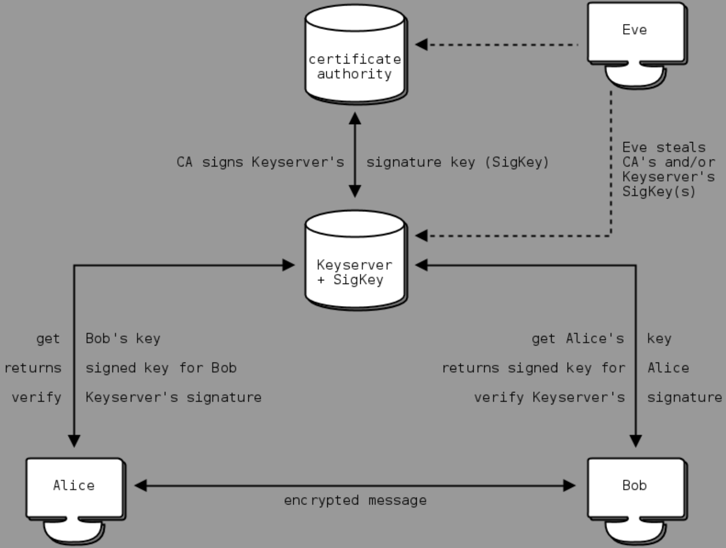
signing keyserver



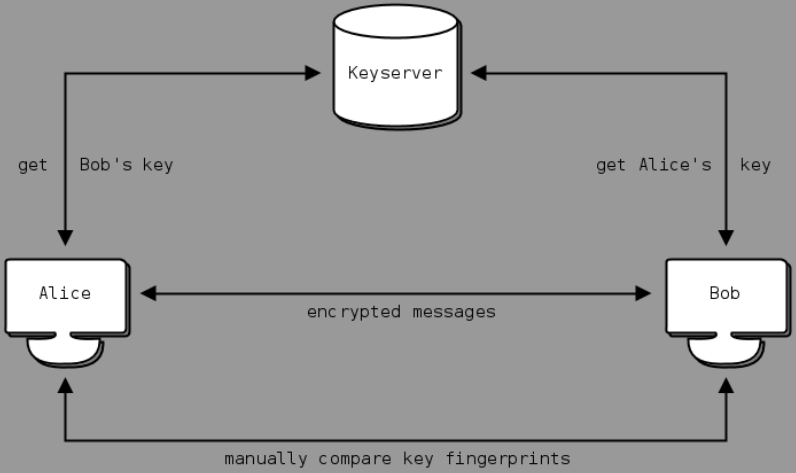
public-key infrastructure (e.g, in SSL)



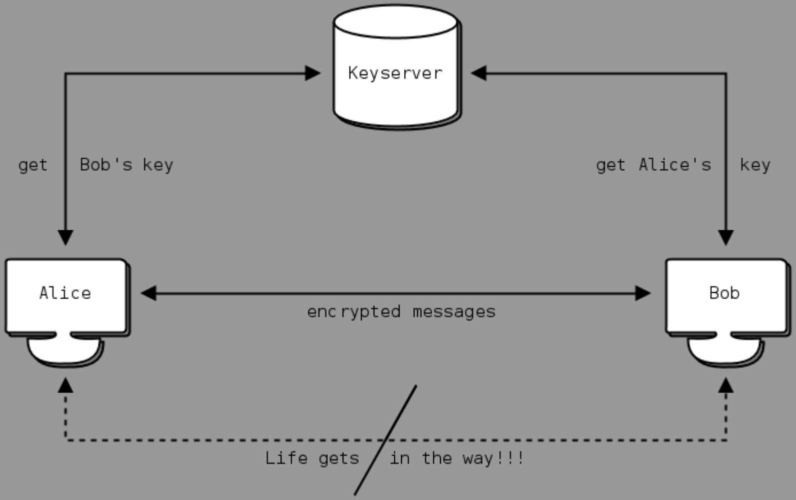
PKI problem (e.g., NSA)



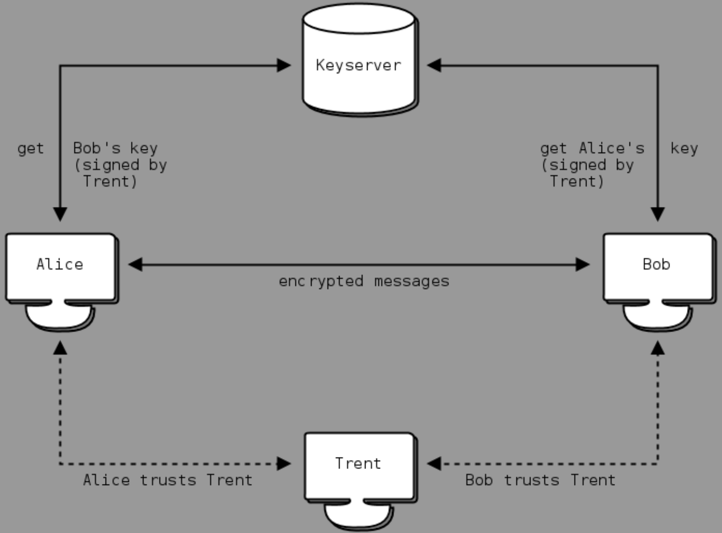
manual fingerprint comparison: idea (used for PGP)



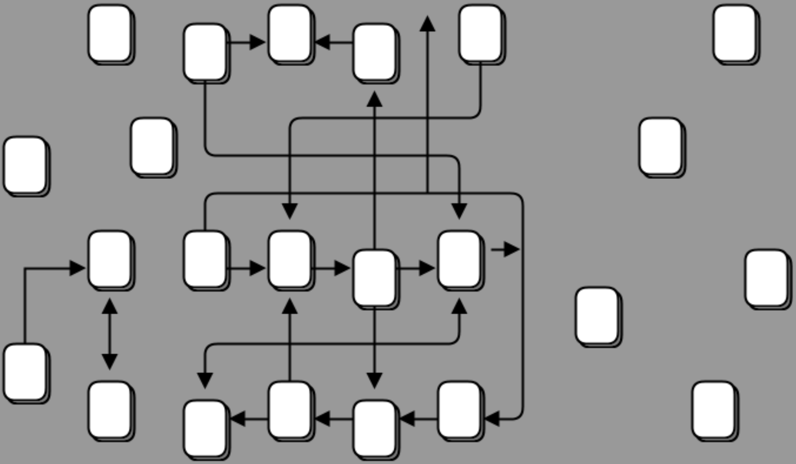
manual fingerprint comparison: reality (also PGP)



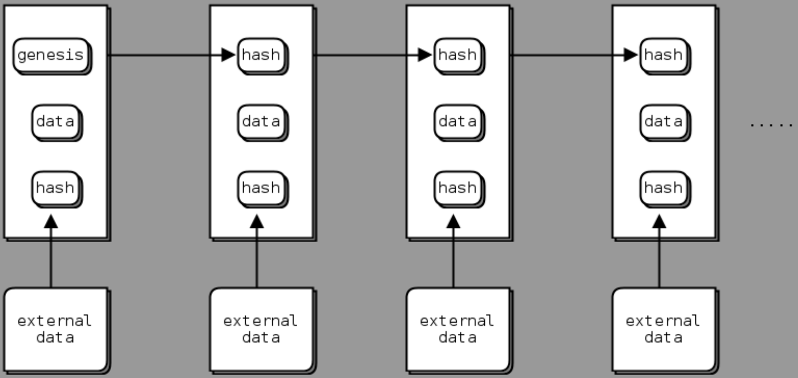
web-of-trust / WOT (used for PGP)



web-of-trust problem (nobody likes keyparties)



Namecoin / Blockchains (Hashchains)



Namecoin / Blockchain problems

blockchains have interesting properties... but not a cure-all!

some problems of Namecoin for key exchange:

- not possible to revoke keys
- chain simulation attack has no attribution
- long confirmation times for key updates
- enumeration of all user IDs easily possible

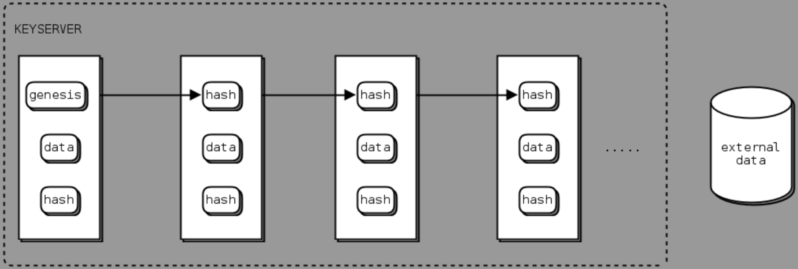
and that's only half the picture...

secure (asynchronous) messaging for the 21th century also needs:

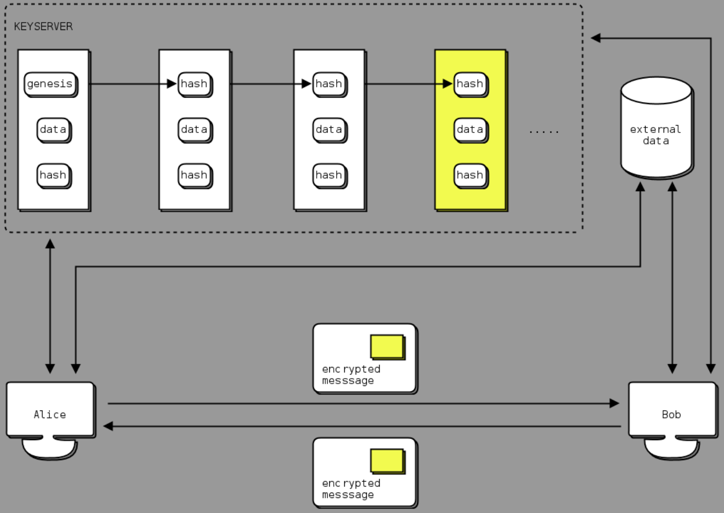
- identity-key binding with human-readable identities
  $\rightarrow$ long-term keys
- perfect forward secrecy (PFS): old messages are unreadable when long-term keys are lost
- PFS needs distribution of short-term keys
- ideally: PFS setup with one-way handshake (convenience)
- secure updates of long-term keys

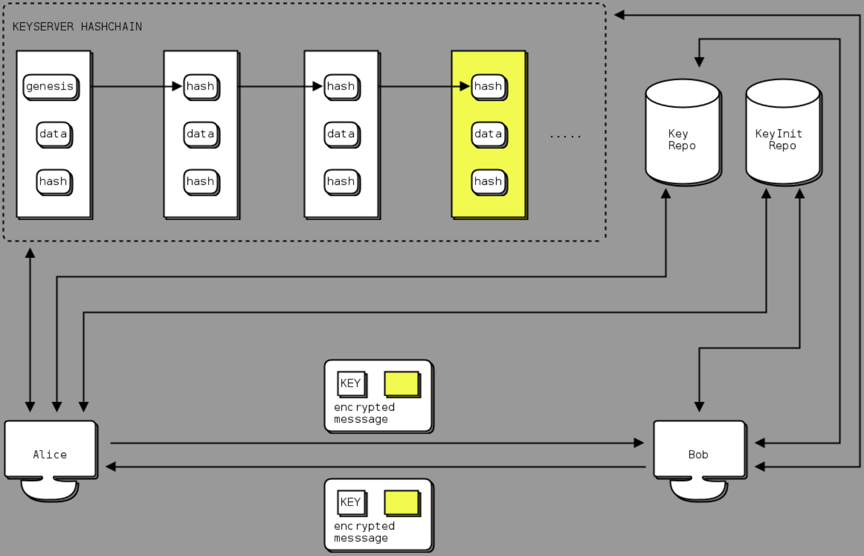
these are all key exchange / key distribution problems!

new approach: a trustless keyserver



a trustless keyserver in action




KEYSERVER HASHCHAIN
genesis
data
hash
hash
data
hash
hash
data
hash
hash
data
hash
.....
Key Repo
Key Init Repo
KEY
encrypted messsage
Alice
Bob
KEY
encrypted messsage

trustless keyserver implementation in Mute

properties:

- exchange of last hashchain entries is explicit consensus
- fixes WOT: clear semantic and no manual intervention

⇒ transfers trust in a few contacts to all of them

- allows to prevent leaking of contacts to keyserver
- enumeration of all user IDs in hashchain not possible, explicit search necessary
- **never** forks in the hashchain

availability of the design:

- client source code is open (BSD-style license)
- protocols are open / specifications published
- key server source is closed (but **trustless**!)

message encryption uses modified Axolotl ratchet (TextSecure)

`mutekeyd`: trustless keyserver walk-through

1. Alice and Bob download the hashchain of the keyserver
2. Alice searches hashchain to check if `alice@mute.one` is free
3. Alice sends `UIDMessage` with `SIGKEY` to keyserver
4. Keyserver adds UIDMessage to hashchain and replies signature
5. Alice sends PFS-keys to KeyInit repository
6. Alice updates hashchain to check `alice@mute.one` was added
7. Alice tells Bob (who registered `bob@mute.one`) about her ID
8. Bob updates his hashchain and searches for `alice@mute.one`
9. Bob fetches one of Alice's PFS-keys from the KeyInit repo
10. Bob sends PFS-message to Alice which contains his own keys
11. Alice can reply without keyserver (only hashchain search)

conclusion

- keyserver operations handled transparently by the client
- users only exchange human-readable, unique identities (e.g., `alice@mute.one`)
- user clients ensure that the trustless keyserver is trustworthy
- if keyserver cheats **once for one user**, the client can **prove** it

$\Rightarrow$ attribution!

- updates of long-term signature keys happen transparently
- message protocol intertwined with keyserver protocol

pointers

Mute:

- Mute α release: `https://github.com/mutecomm/mute`
- trustless keyserver specification also on GitHub
- register for news and β invitation: `http://mute.berlin`

acknowledgments: Jonathan Logan (Mute's chief architect)

contacts:

- `frank@cryptogroup.net` (please use PGP, key on key server)
- `94CC ADA6 E814 FFD5 89D0 48D7 35AF 2AC2 CEC0 0E94`
- `#agora` IRC channel / community: `https://anarplex.net/`

thank you very much for your attention!