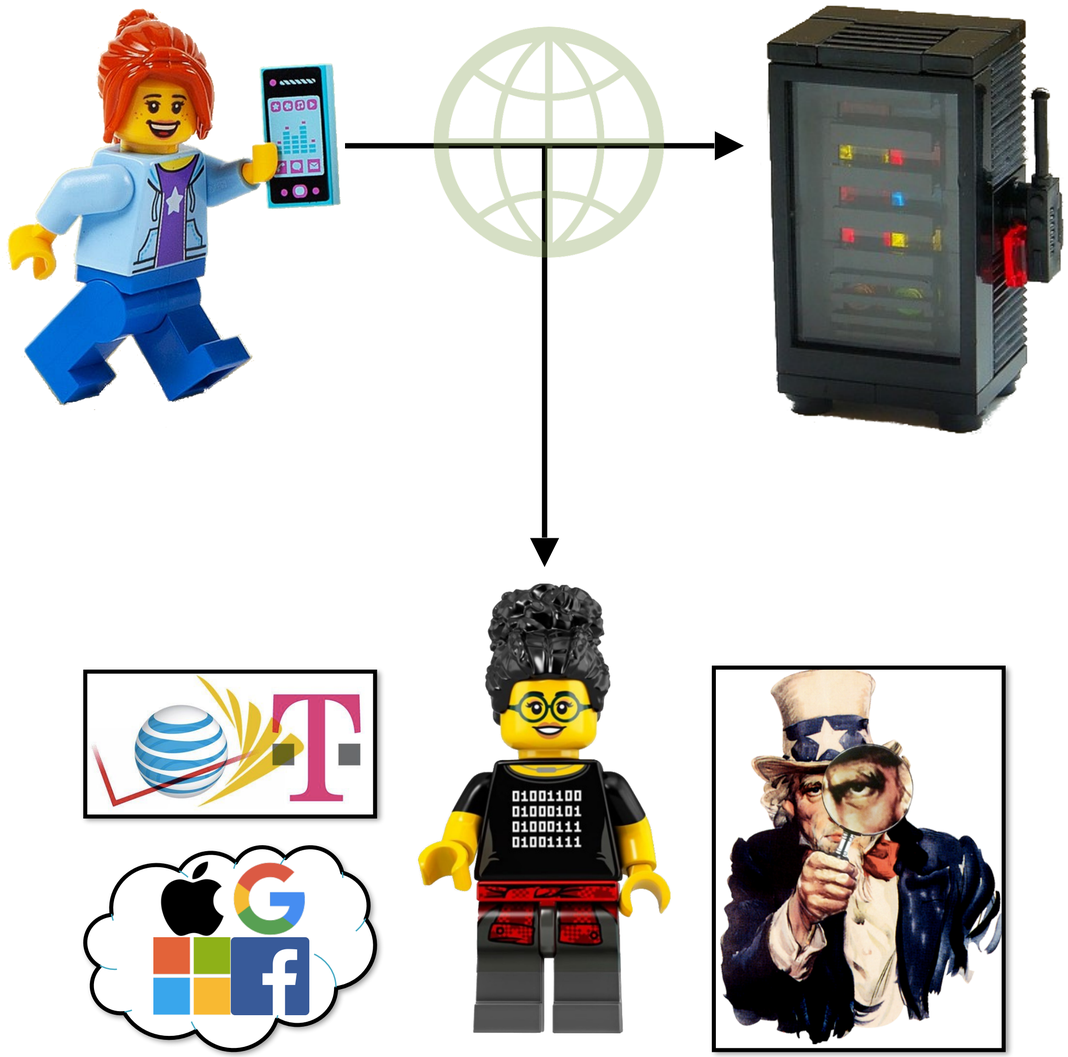
Lecture 17: Signal's double ratchet

Announcements

- Weekly reading: Boneh-Shoup, parts of chapter 9 on authenticated encryption
- HW 7 is due Friday 4/4
- Test 3 is next Monday 4/7 in IEC B10 during discussion lab (9:05-9:55 / 10:10-11)

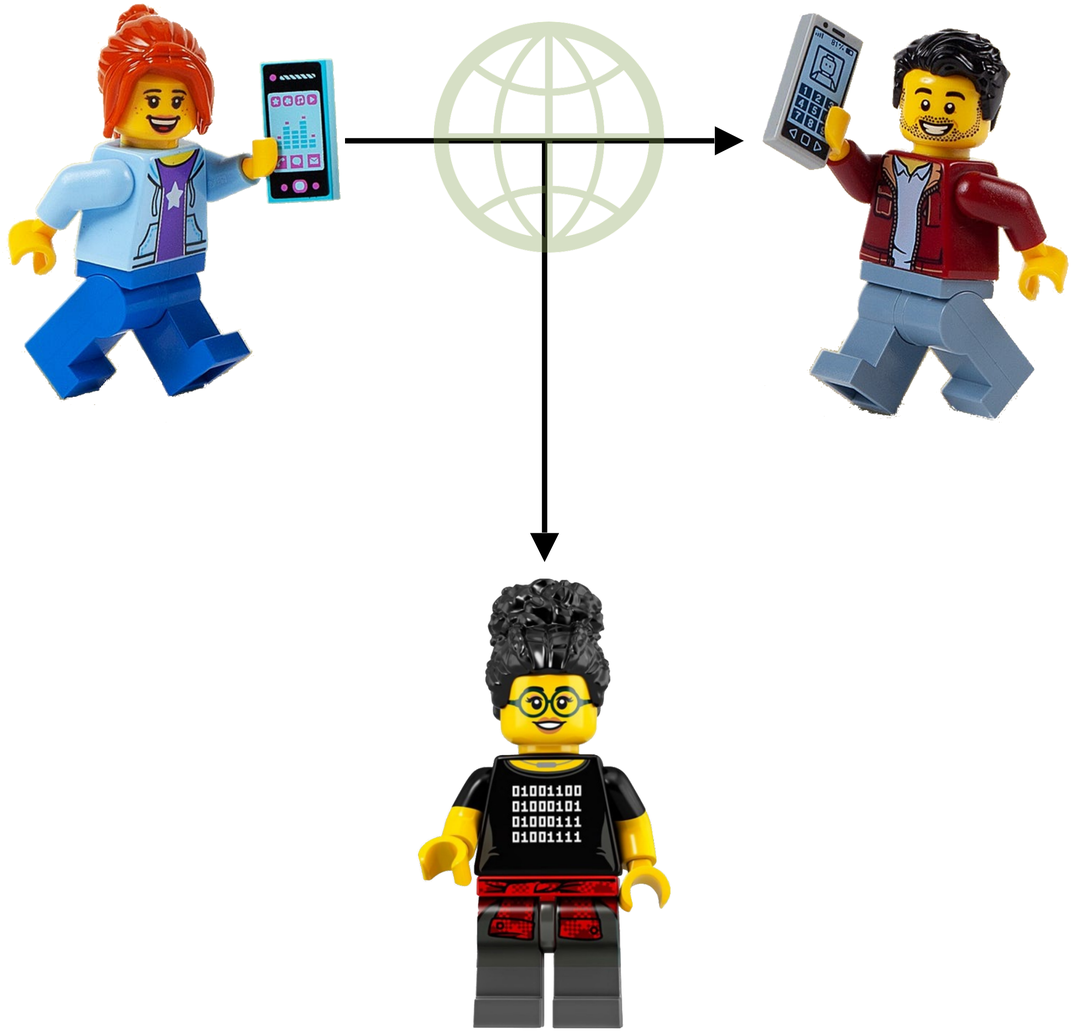
Review: client-server encryption

- Alice wants to talk with a server Bob over the Internet
- They do *not* yet possess a shared secret key
- Our adversary Mallory can read, tamper, add, drop data in transit
  - Mallory is a stand-in for anyone that owns Internet infrastructure



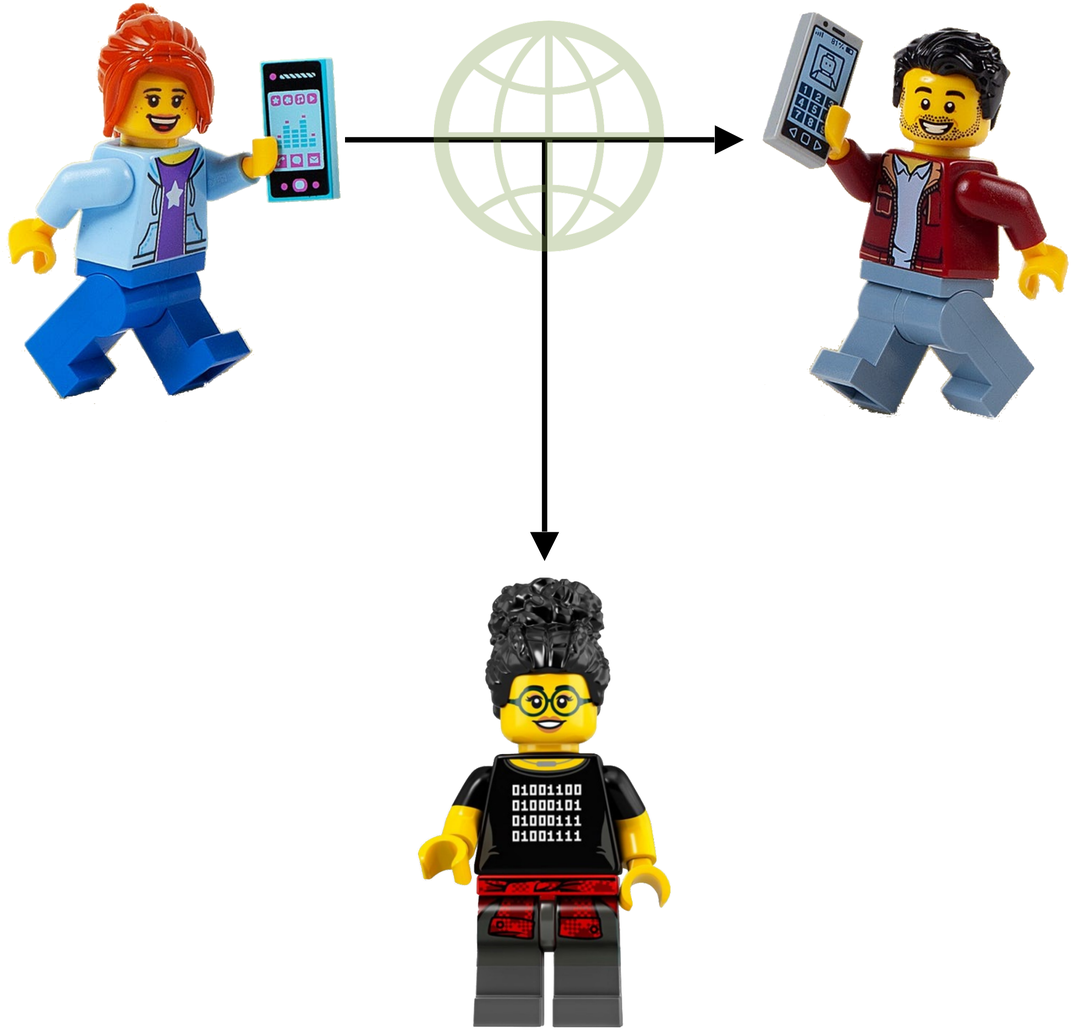
Review: end-to-end encryption

- Now Alice is communicating with Bob's personal laptop/phone
  - They have never met before in person to exchange a key
- Protecting both scenarios involves similar crypto



Objectives

- Protection from network
  - Message confidentiality
  - Sender authenticity + message binding
- Protection from endpoints
  - Secrecy before/after compromise
  - Sender deniability
- Non-goals
  - Hiding metadata (e.g., Alice and Bob's identity, message size)
  - Stopping replay, delay, re-ordering



Objectives

- Protection from network
  - Message confidentiality
  - Sender authenticity + message binding
- Protection from endpoints
  - Secrecy before/after compromise
  - Sender deniability
- Non-goals
  - Hiding metadata (e.g., Alice and Bob's identity, message size)
  - Stopping replay, delay, re-ordering

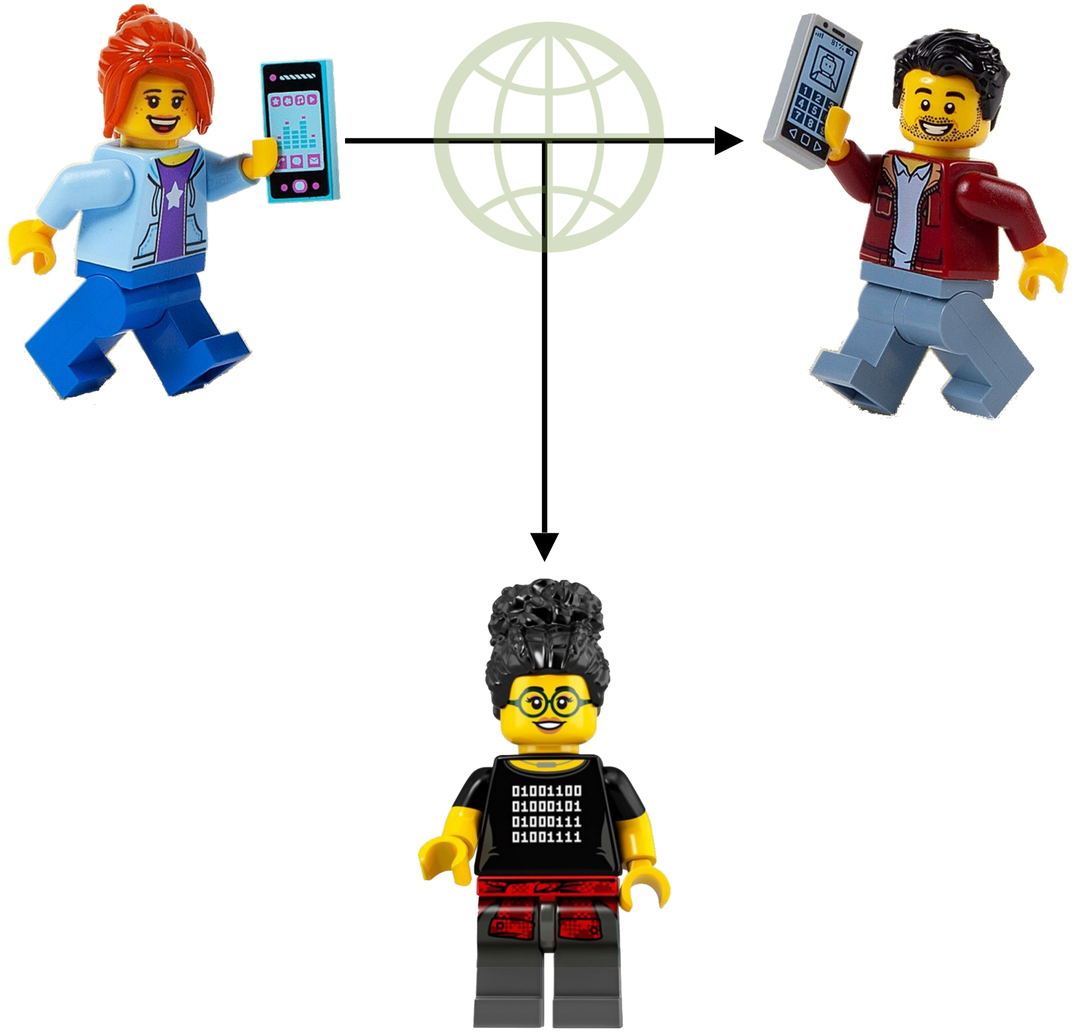
AuthEnc will protect communication on the network, if Alice and Bob already have a shared key *K*

This is new to us…

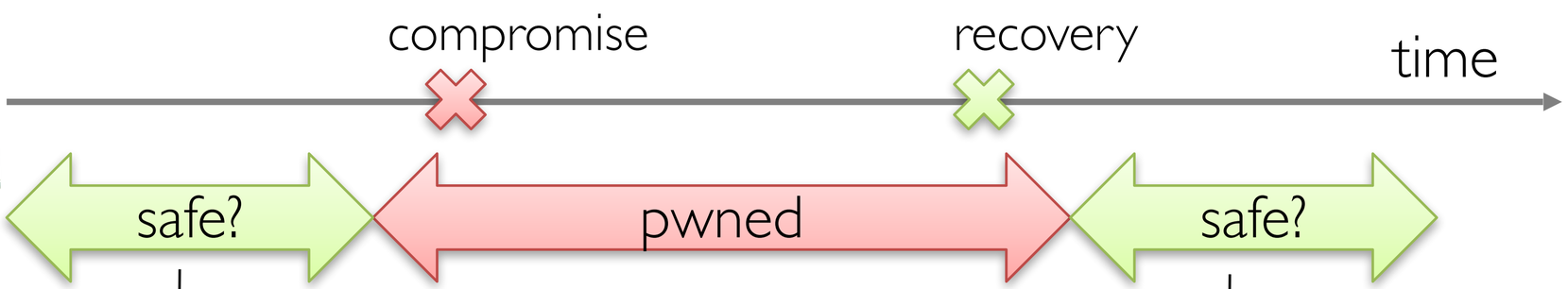
17.1 Forward + backward secrecy and deniability

Overview of Signal

- Key exchange with good deniability
  - Alice and Bob want to generate a shared key without ever having met before
  - Assistance from a partially-trusted entity that mediates this connection
- Key evolution (aka ratcheting) with forward/backward secrecy
  - Use each key to protect just 1 message, then *delete it*!
  - Protect message privacy + integrity against device compromise in past + future
  - Generate a new key for the next message



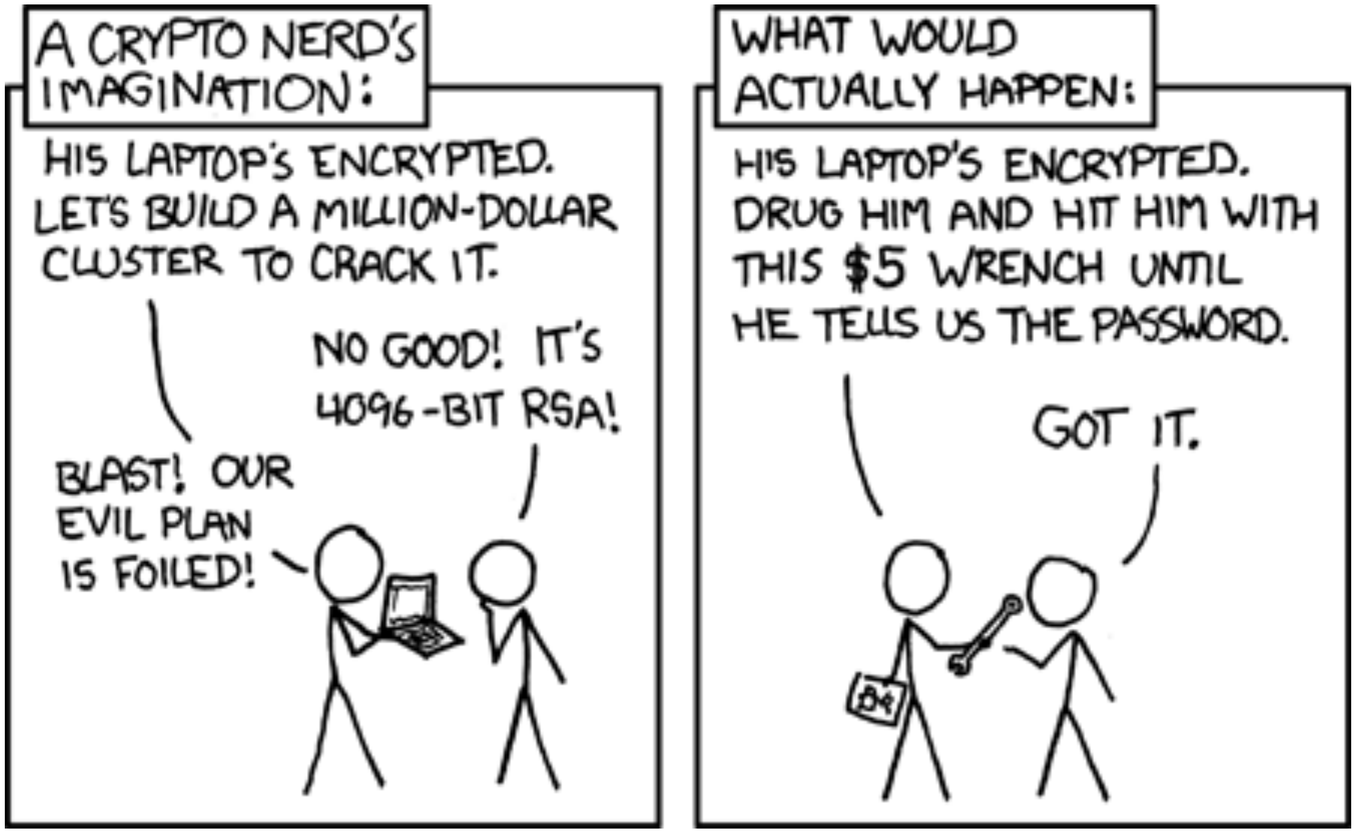
Forward and backward secrecy



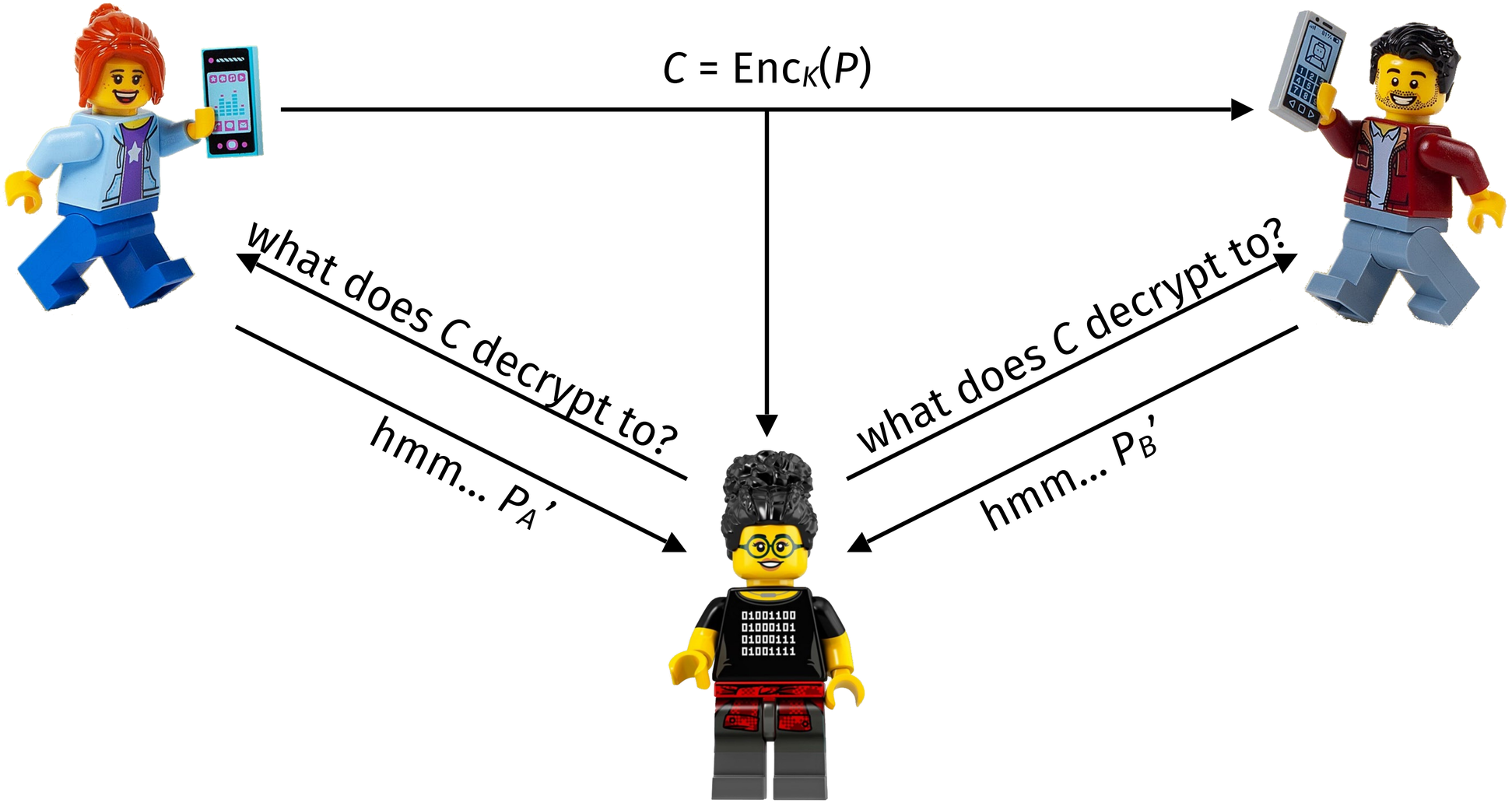
Forward (pre-compromise) secrecy
past messages remain private even if secret state is exposed later

Backward (post-recovery) secrecy
future messages remain private even if secret state is compromised and then the device recovers

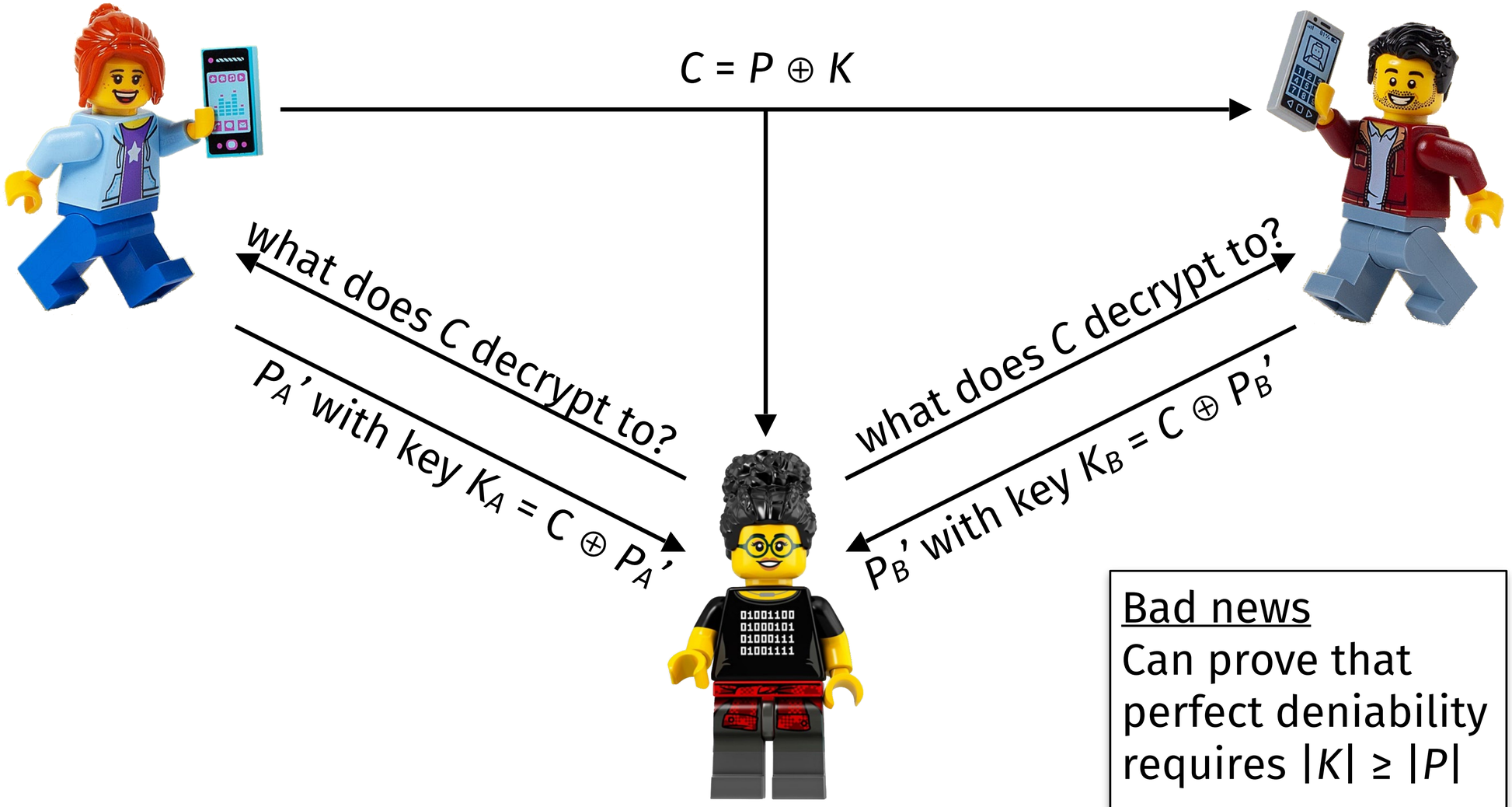
Non-deniable crypto (xkcd.com/538)



Deniable crypto = can pretend you said something else

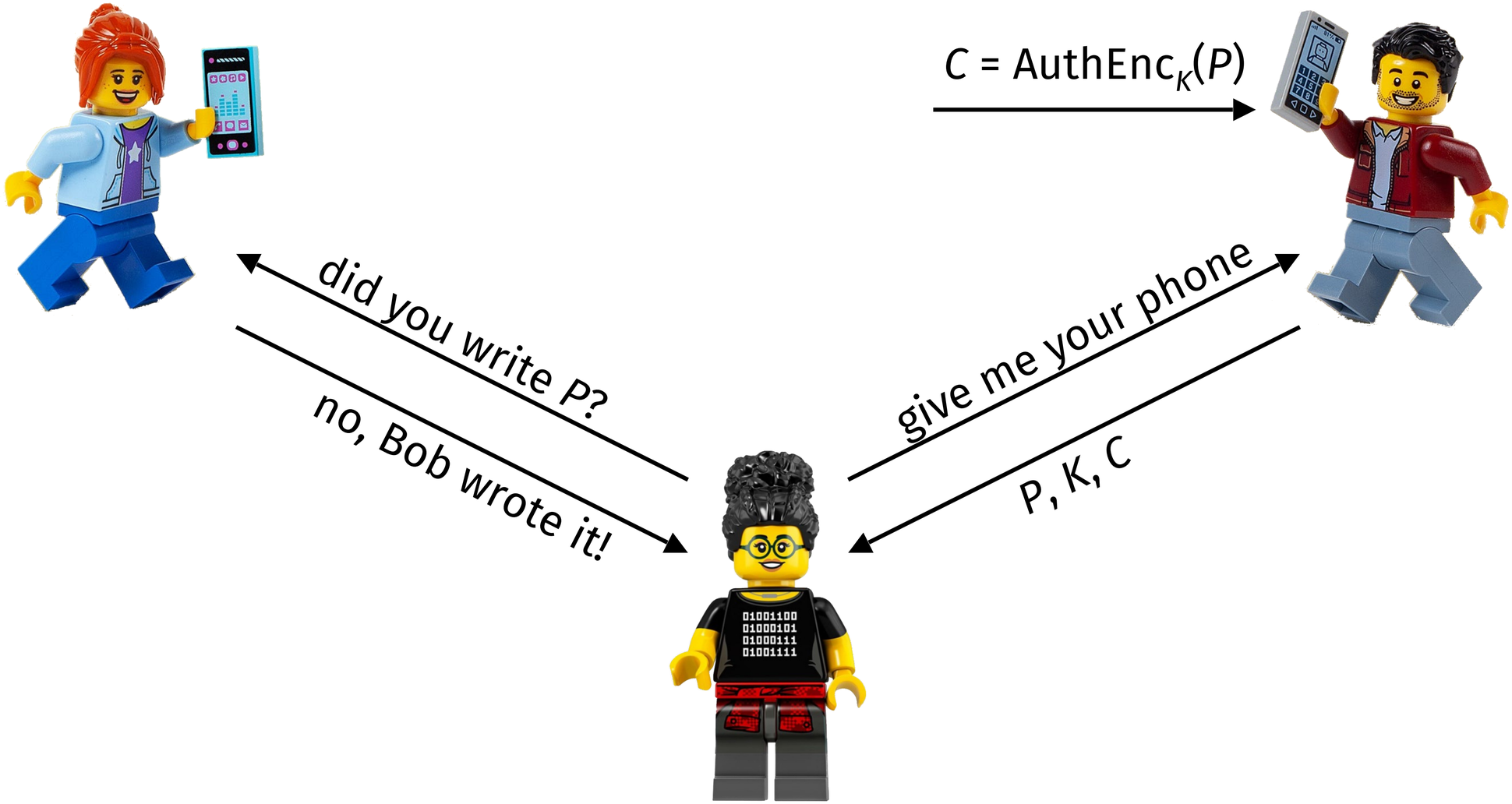


One time pad → perfect deniability



Bad news
Can prove that perfect deniability requires $|K| \geq |P|$

Auth encryption → partial sender deniability



Deniability in practice

The Trump Administration Accidentally Texted Me Its War Plans

U.S. national-security leaders included me in a group chat about upcoming military strikes in Yemen. I didn't think it could be real. Then the bombs started falling.

By Jeffrey Goldberg

Andrew Harnik / Getty

[T]he DKIM authenticity stamp has been widely used by the press, primarily in the context of political email hacks. It's real, it's important, and it's meaningful.

The most famous example is also one of the most divisive: back in 2016, Wikileaks published a batch of stolen emails stolen from John Podesta's Google account. Since the sourcing of these emails was murky, WikiLeaks faced a high burden in proving to readers that these messages were actually authentic. DKIM provided an elegant solution: every email presented on Wikileaks' pages publicly states the verification status of the attached DKIM signatures, something you can see today.

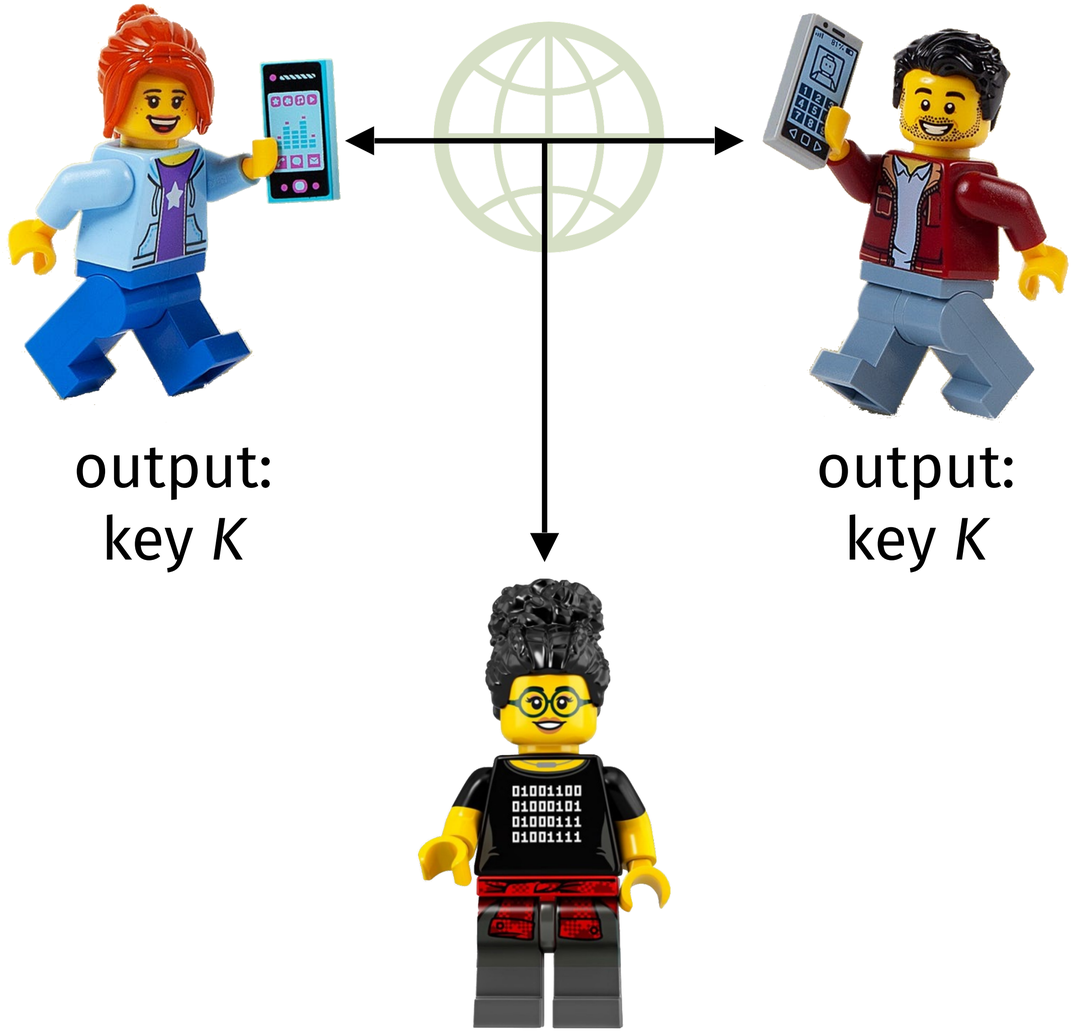
But the Podesta emails weren't the end of the DKIM story. In 2017, ProPublica used DKIM to verify the authenticity of emails allegedly sent to a critic by President Trump's personal lawyer Mark Kasowitz. In 2018, the Associated Press used it once again to verify leaked emails tying a Russian lawyer to Donald Trump Jr.

Source: https://blog.cryptographyengineering.com/2020/11/16/ok-google-please-publish-your-dkim-secret-keys/

17.2 Authenticated Key Exchange

Generating the first shared secret

- Alice and Bob have
  - Never met in person, or else they could exchange a key face-to-face
  - Lack any shared secrets, or else they could run PBKDF2 on them
- They do have individual secrets!
- Question: can Alice and Bob generate a symmetric key *K* and keep it secret from Eve/Mallory?



output:
key *K*

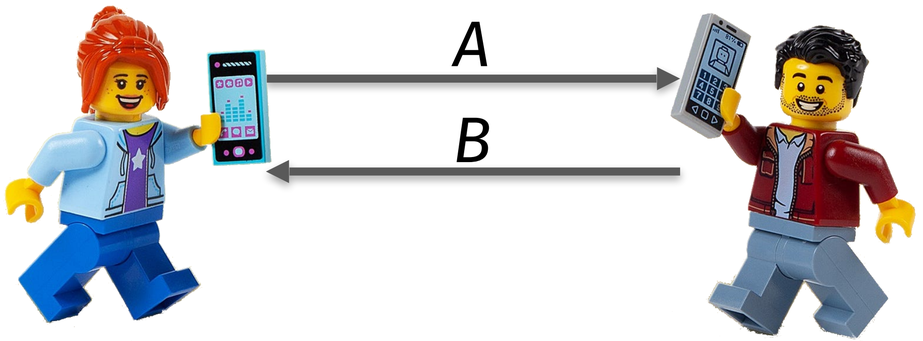
output:
key *K*

Diffie-Hellman key agreement (against a passive Eve)

Protocol

Choose a randomly
Compute $A = g^a$

Choose b randomly
Compute $B = g^b$



Output $K = B^a$

Output $K = A^b$

$\text{AuthEnc}_K(P)$

Delete a, K

Delete b, K

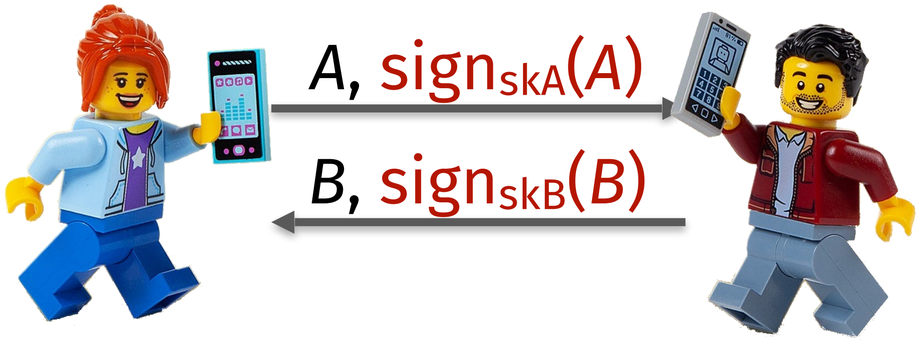
Analysis

- *Correctness*: shared secret since $A^b = (g^a)^b = g^{ab} = (g^b)^a = B^a$
- *Secrecy*: to learn K, a passive Eve given g, g^a, g^b must find g^{ab}
  - There exist mathematical spaces in which this problem is hard!
- *Forward secrecy*: Choices of a, b are ephemeral; delete afterward so even you cannot compute K

D-H + signatures = Authenticated key exchange

Choose a randomly
Compute $A = g^a$

Choose b randomly
Compute $B = g^b$

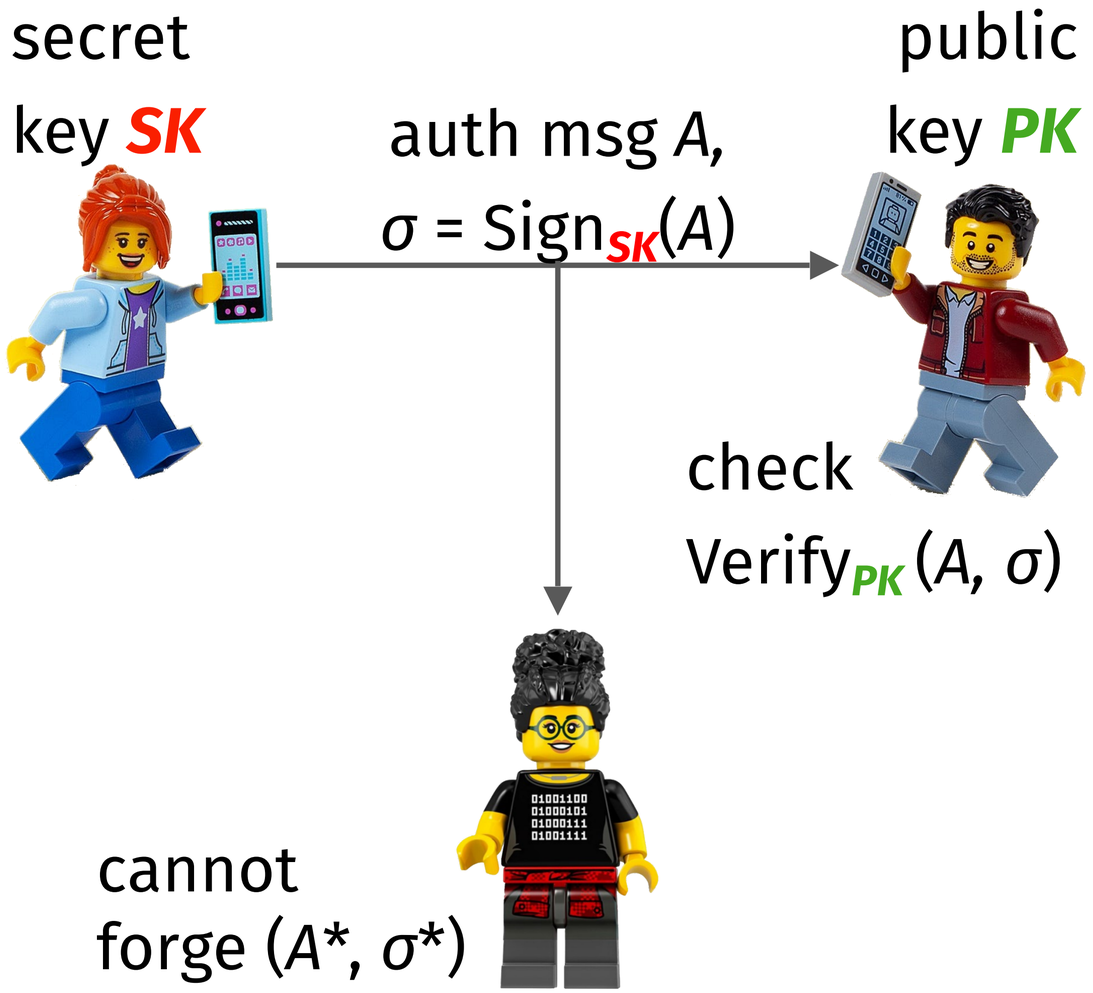


1. Alice and Bob sign their messages during Diffie-Hellman key exchange
2. Alice and Bob verify signature of each other's messages
3. Use shared key $A^b = B^a$ for (deniable) symmetric authenticated encryption

Question: how do Alice and Bob learn each other's public keys?

17.3 Digital Certificates & the Public Key Infrastructure (PKI)

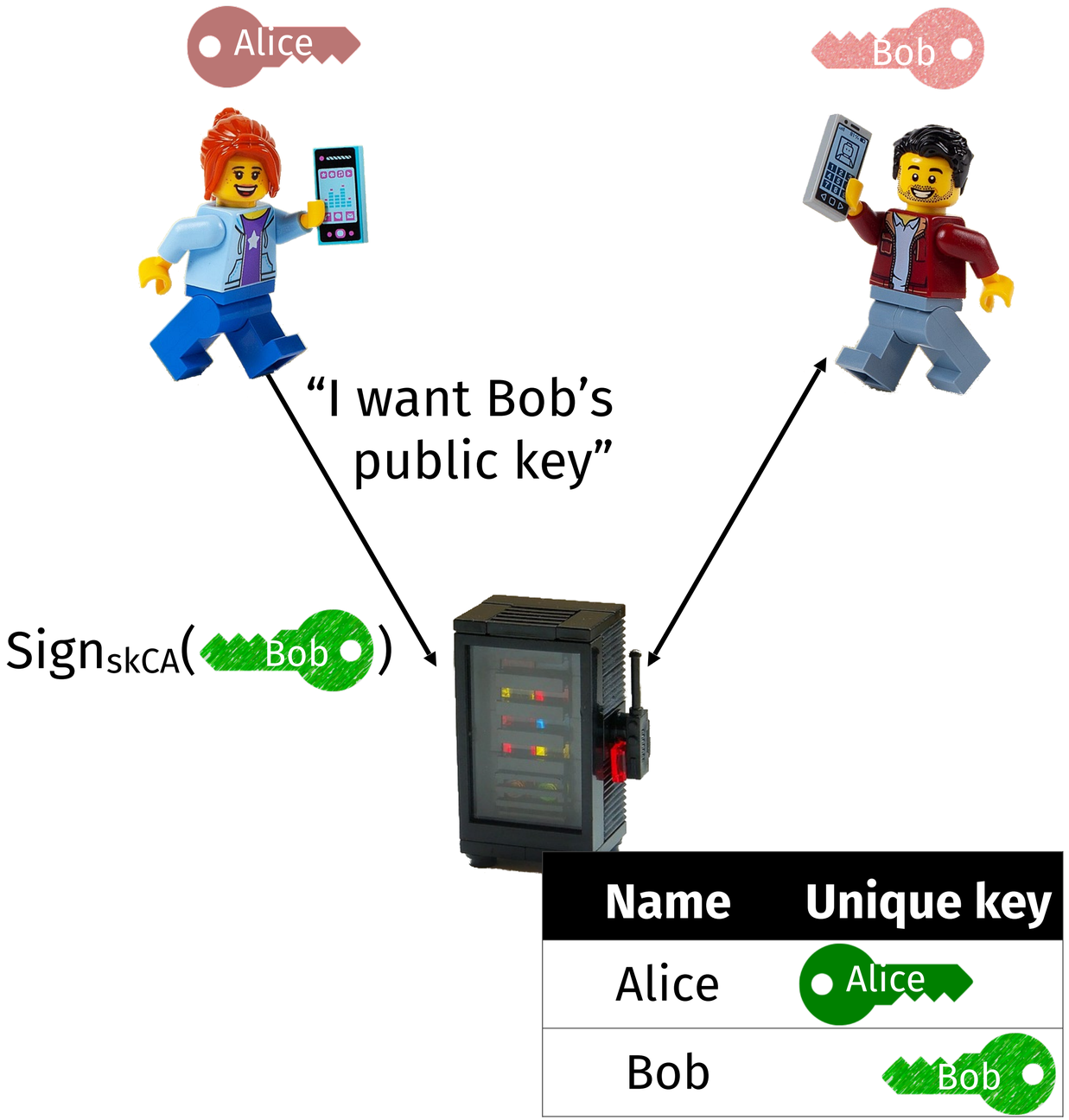
Recap: Digital signatures provide *public* authentication



Property	MAC	Sign
Sender auth: Bob knows Alice sent *A*	✓	✓
Msg auth: Bob can detect tampering	✓	✓
Receiver auth: Bob knows *A* for him	✓	✗
Partial deniability: Alice can deny *A*	✓	✗

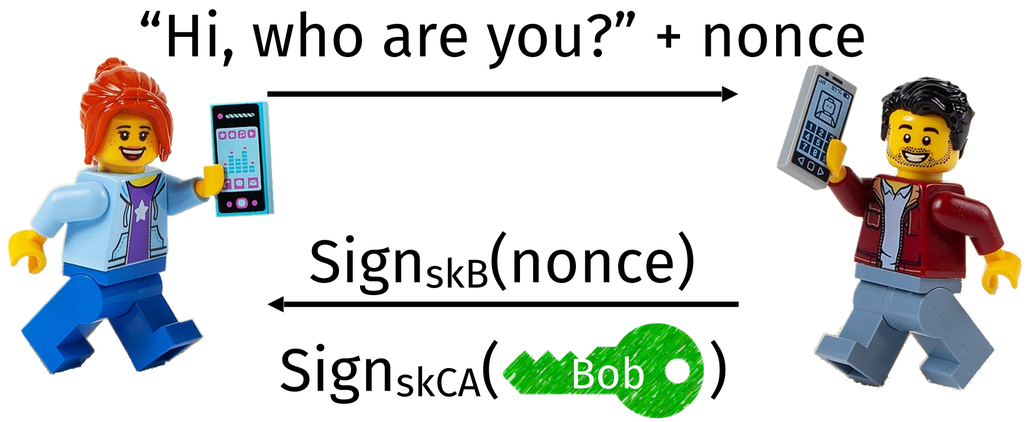
Public key infrastructure

- A *certificate authority* stores all public keys (like a phone book)
  - Server does *not* learn private keys
- Anyone can query the authority to learn someone else's key
- CA signs response *certificates* so they can be verified as legit
- Alice knows the CA's public key because it is included in her OS



PKI improved

- Alice talks with Bob, not CA
- Bob includes a certificate that the signing key belongs to him
- This figure shows a simplified version of the Transport Layer Security (TLS) handshake



Google.com in Firefox:

Technical Details

Connection Encrypted (TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, 128 bit keys, TLS 1.2)

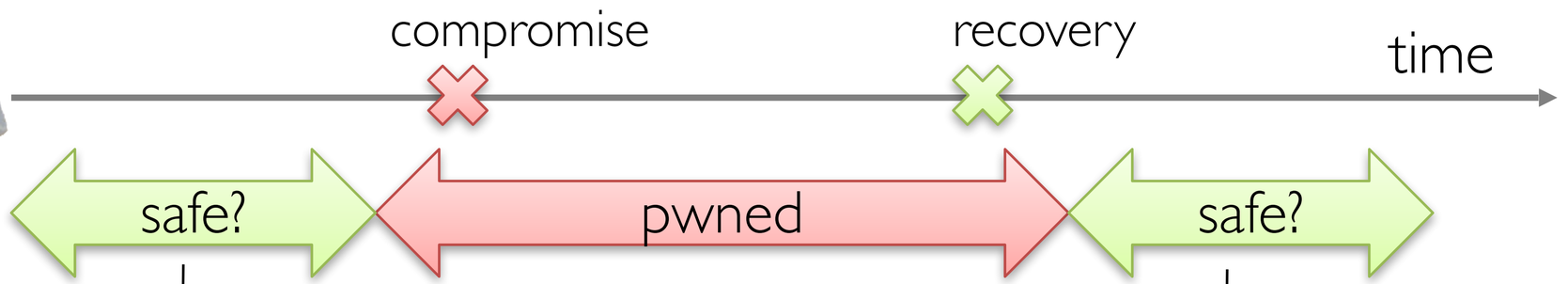
BU login page in Firefox (2017):

Technical Details

Connection Encrypted (TLS_RSA_WITH_AES_256_CBC_SHA, 256 bit keys, TLS 1.2)

Name	Unique key
Alice	Alice
Bob	Bob

What if Bob's secret signing key is compromised?



Forward (pre-compromise) secrecy

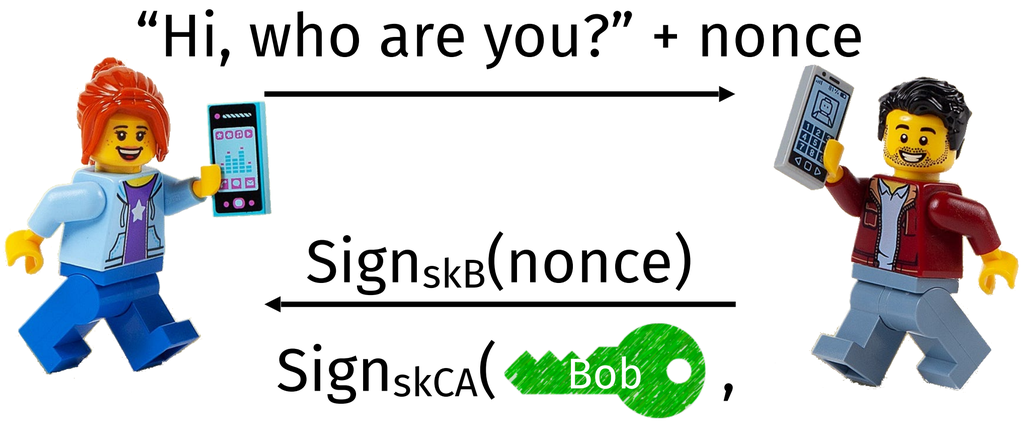
Yes! Unless Mallory has a time machine, signatures Alice verified before a breach must be valid.

Backward (post-recovery) secrecy

No. If Mallory has Bob's secret key, she can sign messages and Alice will believe they are from Bob.

Backward security technique #1: Cert expiration

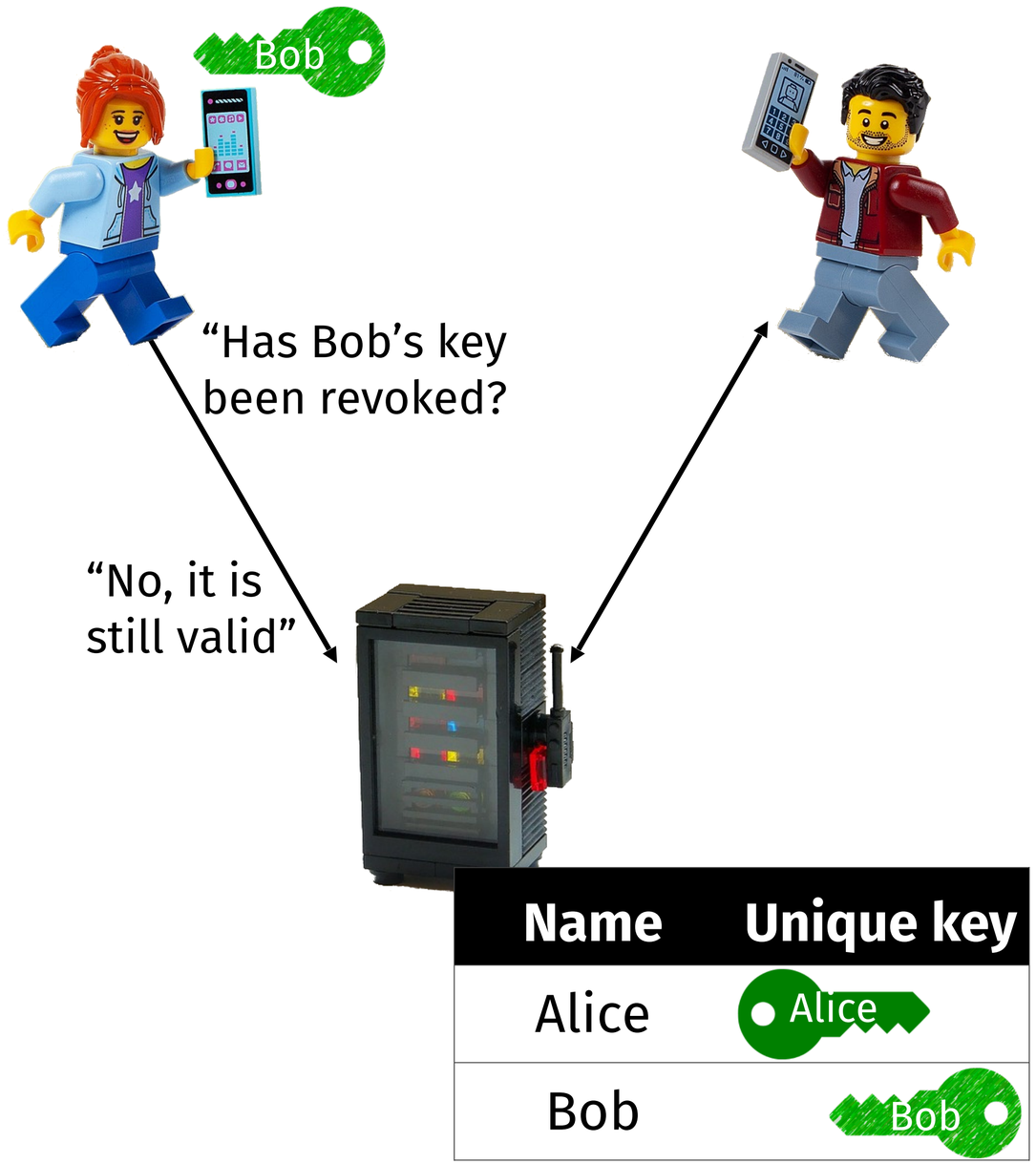
- Certificate states that Alice should only trusts Bob's key for a limited time
- Afterward, Bob must register a new public key with the CA
- (Cert expiration also helps to deal with Moore's law: keys become bigger over time)



Name	Unique key
Alice	Alice
Bob	Bob

Backward security technique #2: Cert revocation

- CA binds public key to a name
- If you lose control of your public key, you should tell the CA to break this binding
- Every CA maintains a certificate revocation list that anyone can query



Backward security technique #2: Cert revocation

- CA binds public key to a name
- If you lose control of your public key, you should tell the CA to break this binding
- Every CA maintains a certificate revocation list that anyone can query