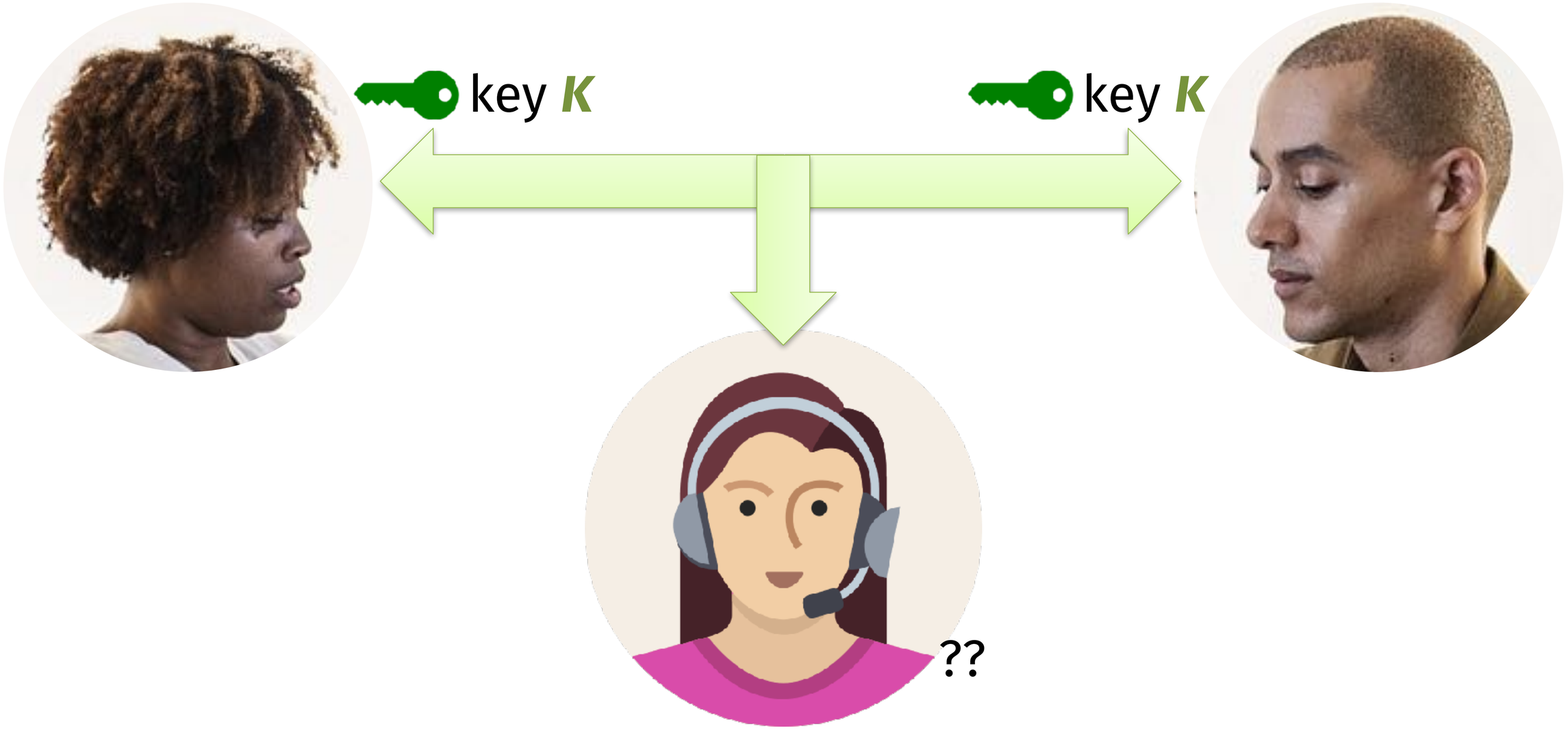
Lecture 15: Public key infrastructure

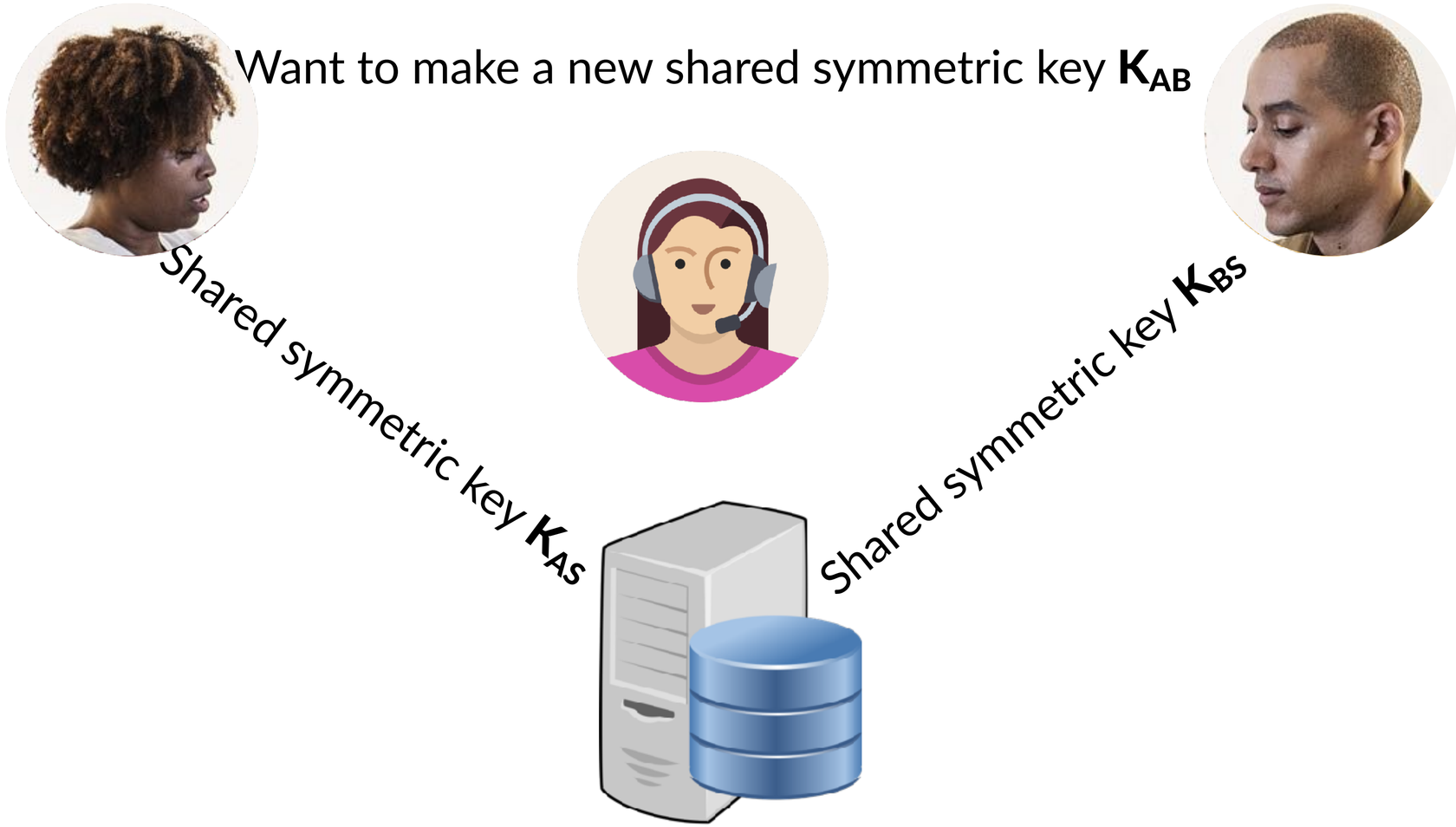
- Lab 7 due Monday 4/1 at 11pm
- Office hours this week moved to Wednesday at 11am-1pm
- Guest lectures on cryptography + law starting on Thursday
- Read *The Moral Character of Cryptographic Work* by Friday

Part 3: Generate, exchange, evolve, and delete keys



Last time: Needham–Schroeder protocol

Objective: with the help of a trusted server, Alice + Bob agree on a shared key



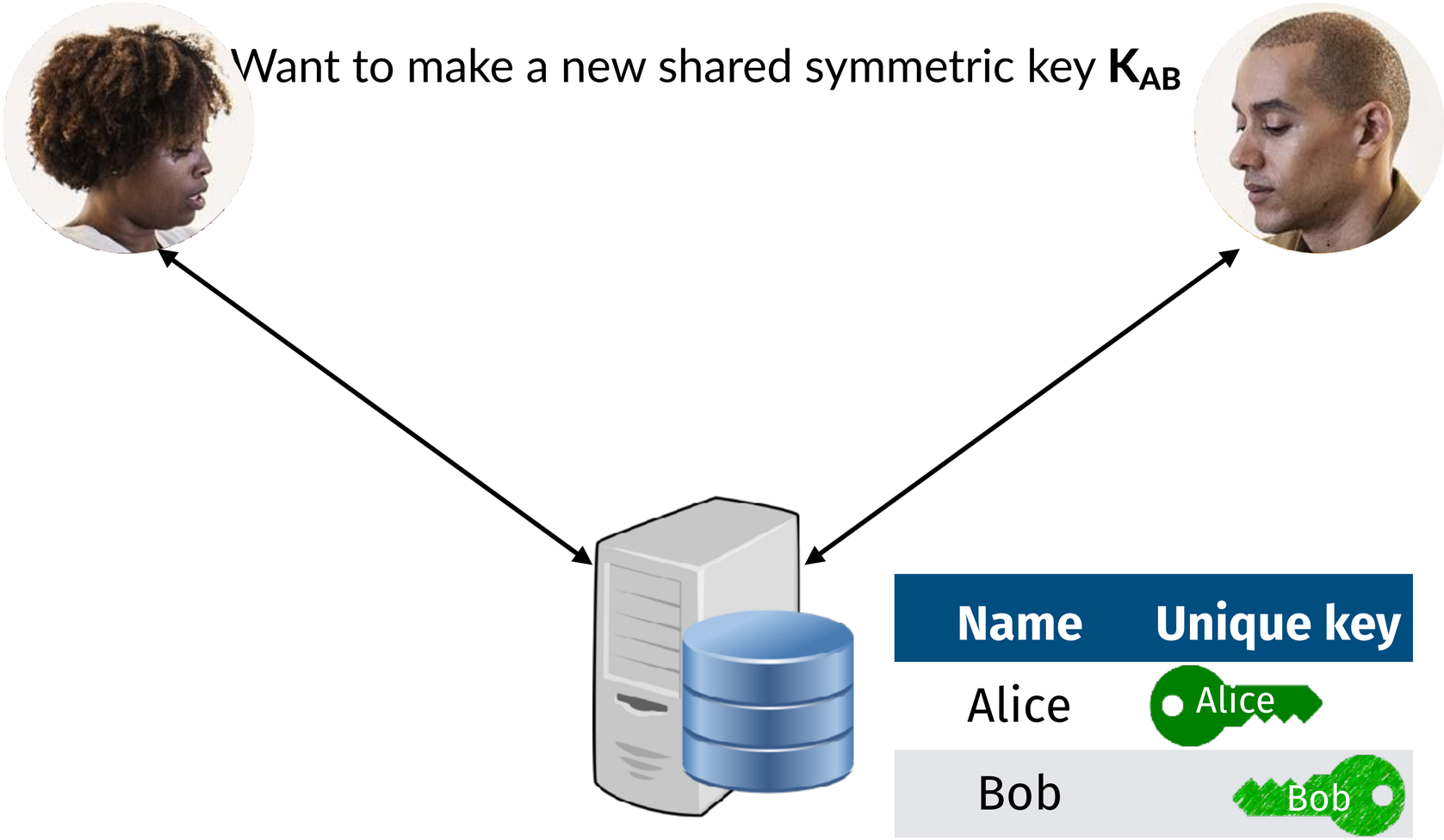
Kerberos = Key Management for Access Control

“This bar is pretty good, but you have to go stand in line for a ticket before they serve you.”

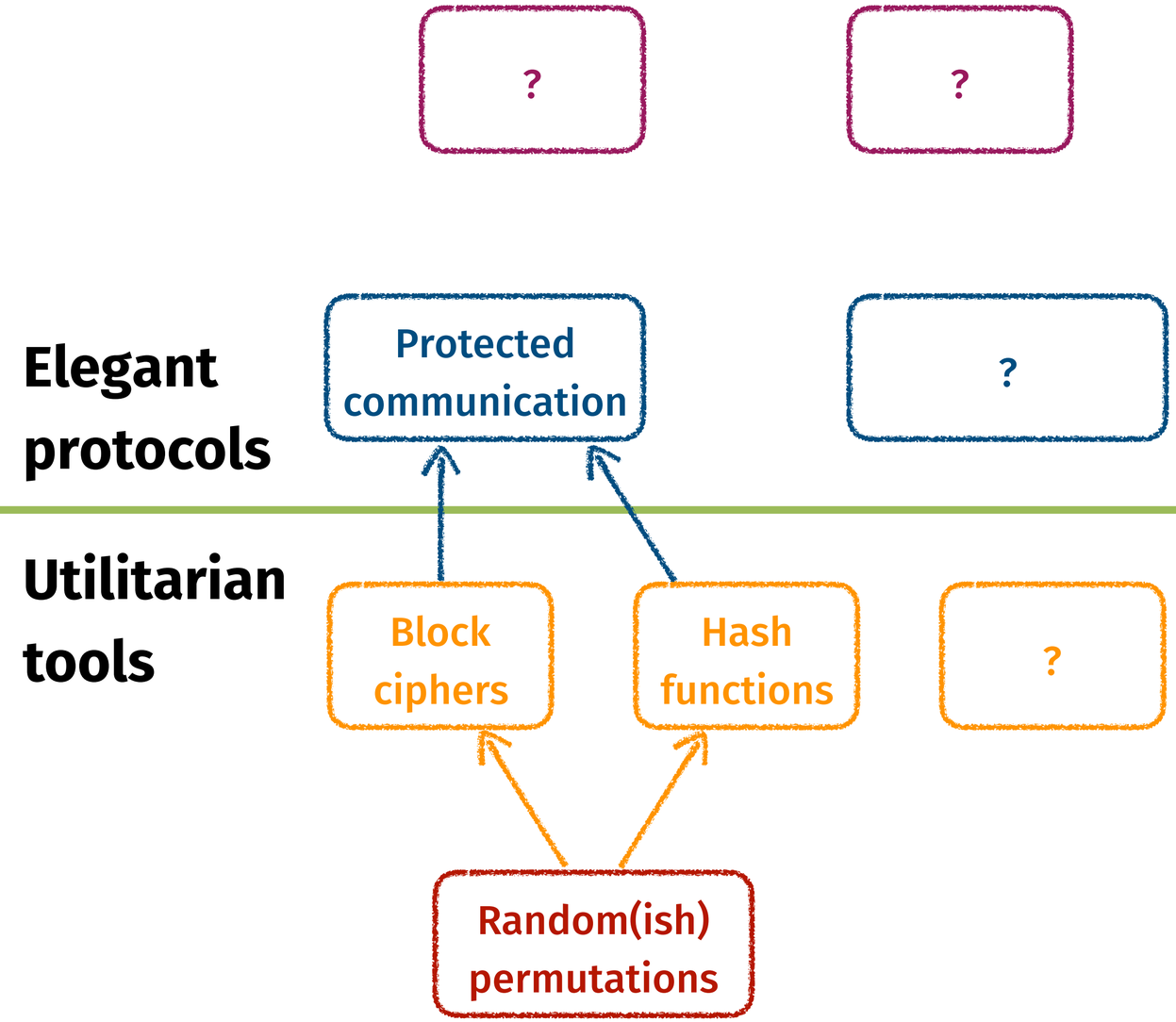
Source:
twitter.com/sweis/status/982272891948421120

Today: Public key infrastructure

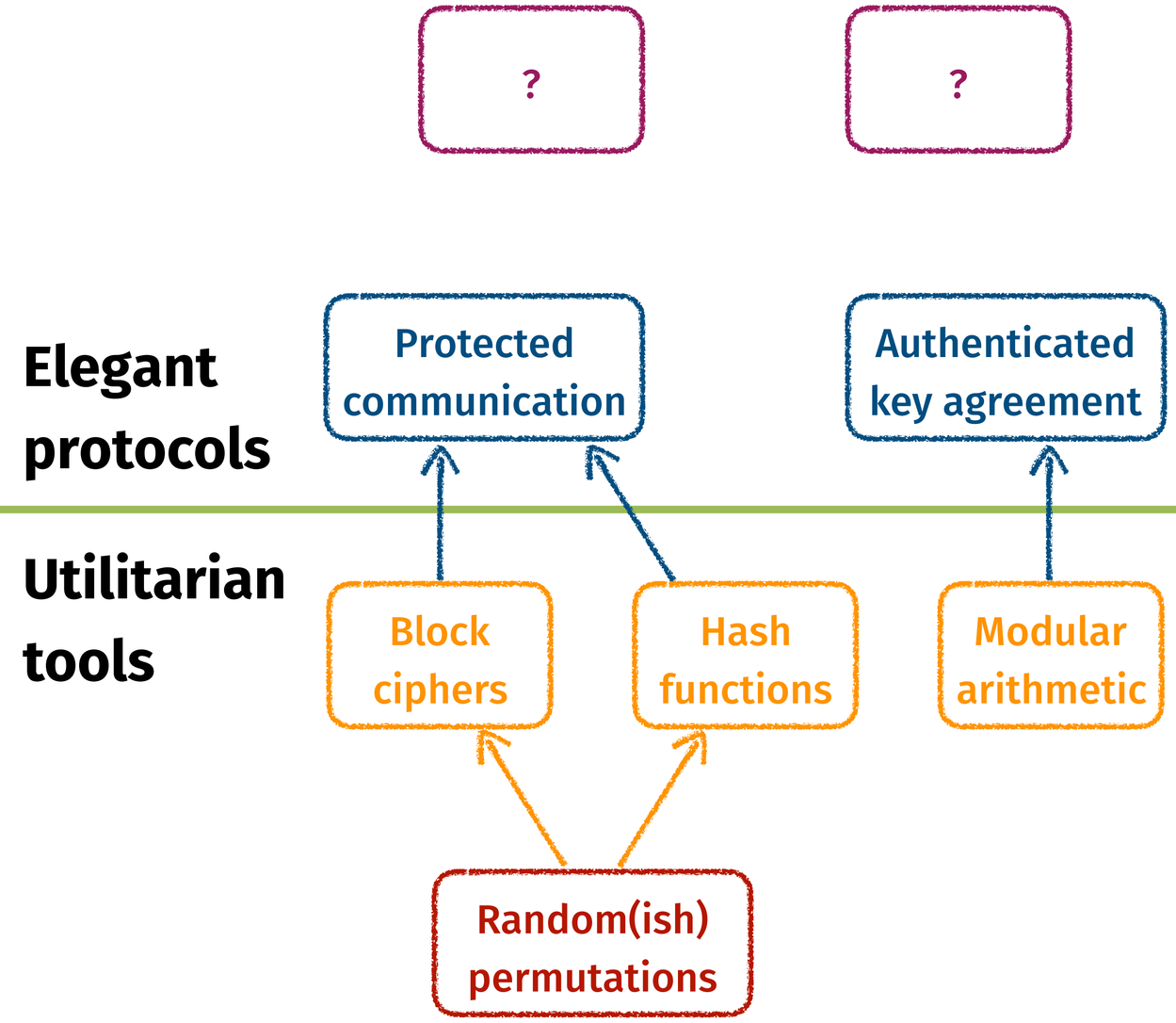
Lower trust in the server, at the expense of using more expensive cryptography



Course roadmap



Course roadmap

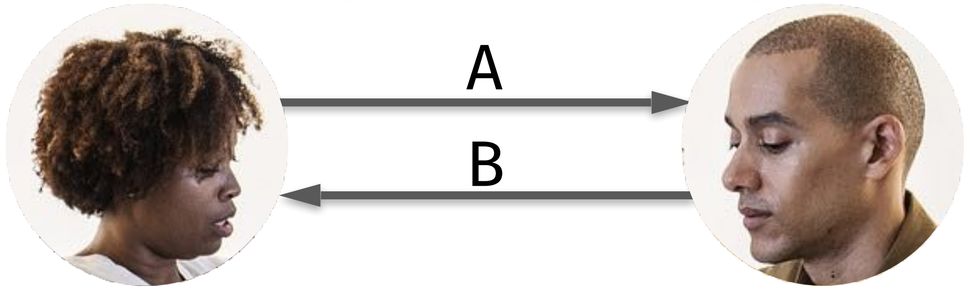


Diffie-Hellman key agreement (last Friday's discussion)

Protocol (for a publicly known g)

Choose *a* randomly
Compute $A = g^a$

Choose *b* randomly
Compute $B = g^b$



Output B^a

Output A^b

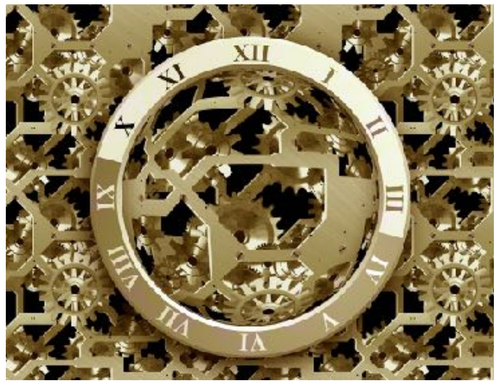
Shared secret = g^{ab}

Analysis

- *Correctness*: commutativity
  $A^b = (g^a)^b = g^{ab} = (g^b)^a = B^a$
- *Security*: to learn the key, a passive Eve must solve following problem
  - Knows g, g^a, g^b
  - Wants to find g^{ab}
- Forward secrecy: Choices of a, b are *ephemeral*, can delete when done
- Active attacker can cause problems!

How to perform key exchange securely?

Modular arithmetic

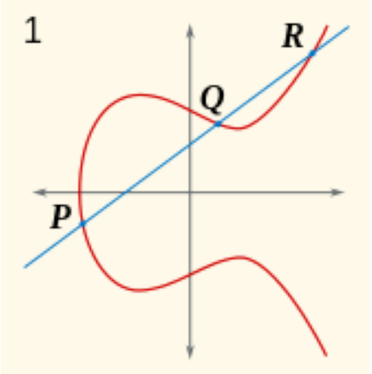


- Raise a constant to any power, e.g. $x \mapsto 3^x \pmod 7$

x	1	2	3	4	5	6
3^x	3	2	6	4	5	1

- Permutation, but hard* to invert

Elliptic curves



- Elliptic curve: a cubic equation $y^2 = x^3 + ax + b \pmod p$
- Consider set of points on this curve
- We can “multiply” points using the rule $P \cdot Q \cdot R = 1$

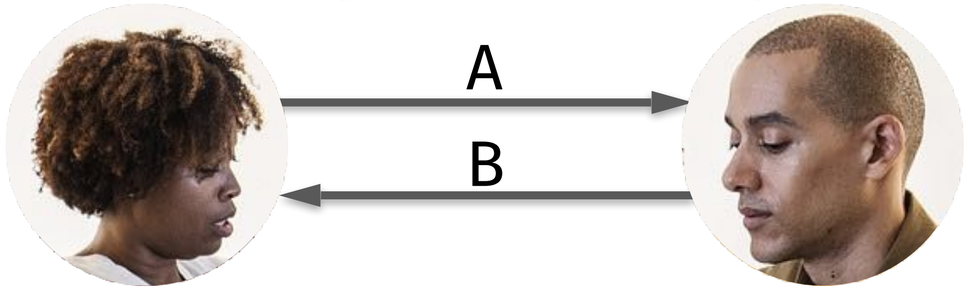
* = really need to take the group of quadratic residues (i.e., the even half of the truth table)

Diffie-Hellman key agreement

Protocol (for a publicly known g)

Choose *a* randomly
Compute $A = g^a$

Choose *b* randomly
Compute $B = g^b$



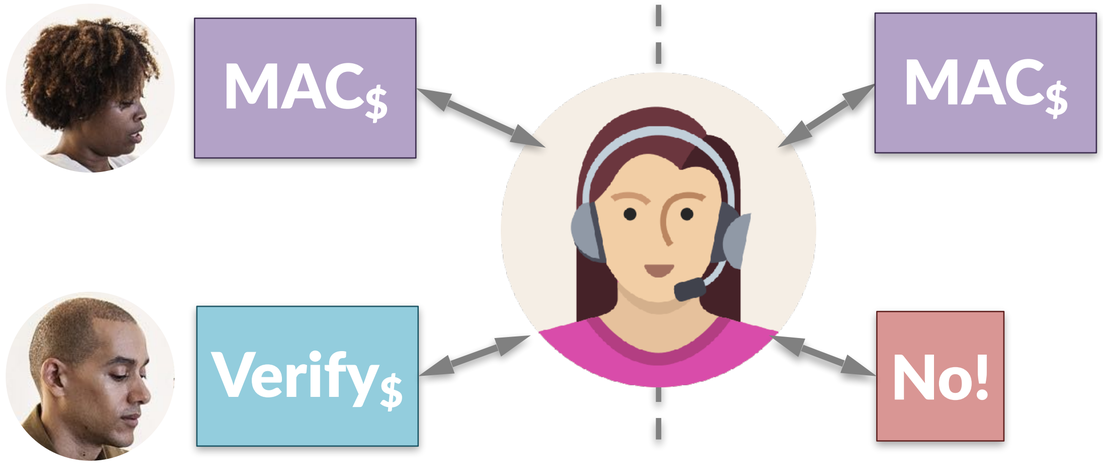
Output B^a

Output A^b

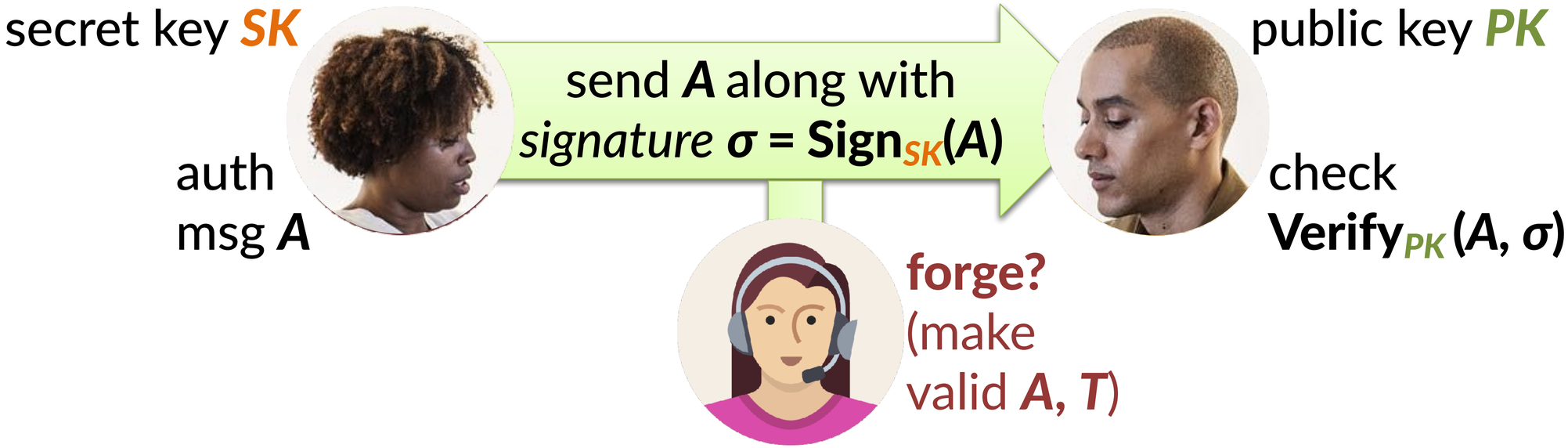
Shared secret = g^{ab}

How do Alice and Bob know that they're talking with each other?

Solution: Use a MAC?



Let's build a *public* method to authenticate message origin

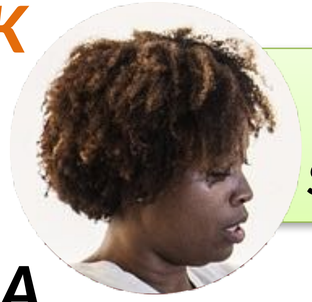


- In the symmetric case, Alice & Bob have a key that nobody else has
  - As a result, Bob knows Alice sent A *and* that the message was intended for him
  - Tag is also deniable because either Alice or Bob could have made it
- In the public case, Alice has a secret *SK* and *everyone* knows corresponding *PK*
  - So, anyone in the world can verify that Alice wrote that message (to somebody…)
  - Also, asymmetry leads to non-deniability: Bob can't make ***σ*** anymore

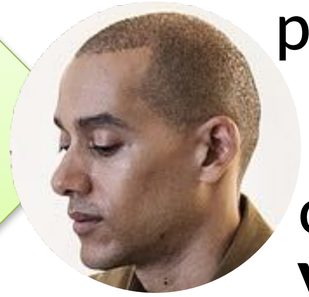
Security for public-key signatures

secret key ***SK***

auth msg ***A***



send ***A*** along with *signature* $\sigma = \mathbf{Sign}_{SK}(A)$



public key ***PK***

check $\mathbf{Verify}_{PK}(A, \sigma)$

EU-CMA security similar to before: Alice baits Mallory into producing a forgery

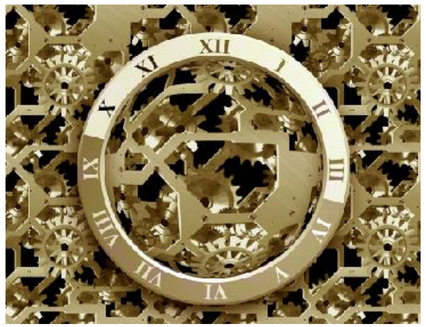
❶ choose ***SK***, give Mallory ***PK***

Mallory wins if
1. Valid forgery
2. It's new

How to make digital signatures?

Modular arithmetic

- Similar math as with key exchange



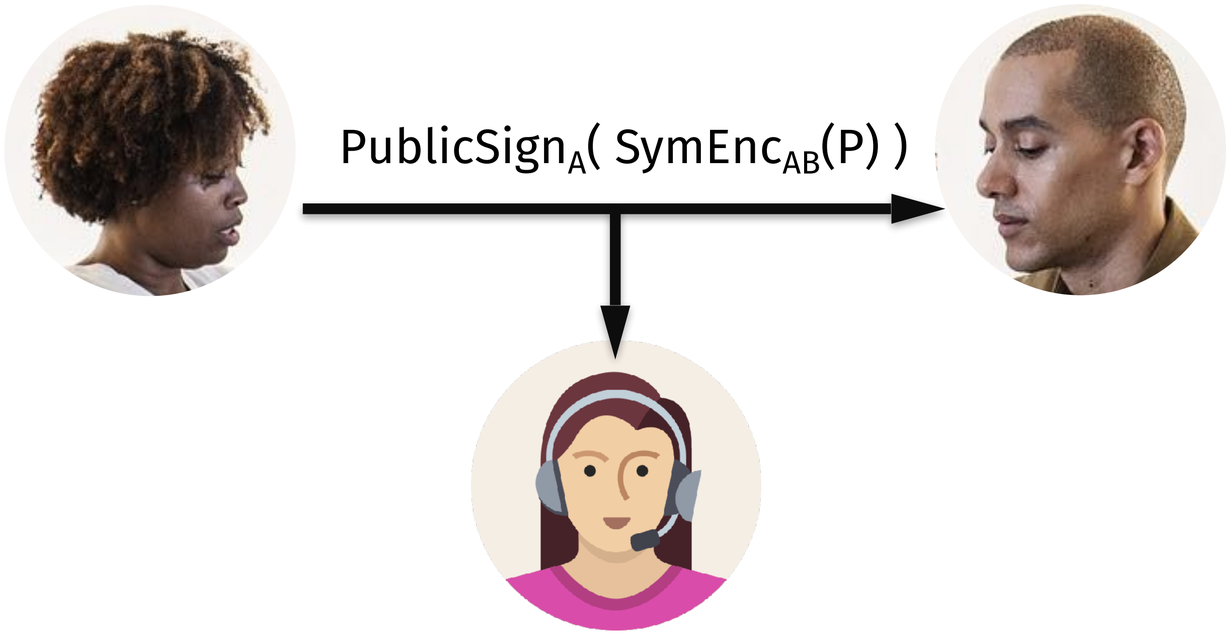
- Two common methods
  - (EC)DSA — NIST standard
  - Schnorr signatures — simpler but patented, will see on Friday

RSA (Rivest, Shamir, Adleman)

- Relies (more or less) on the hardness of factoring N = p q
- Less commonly used nowadays
- Will explore in this week's lab

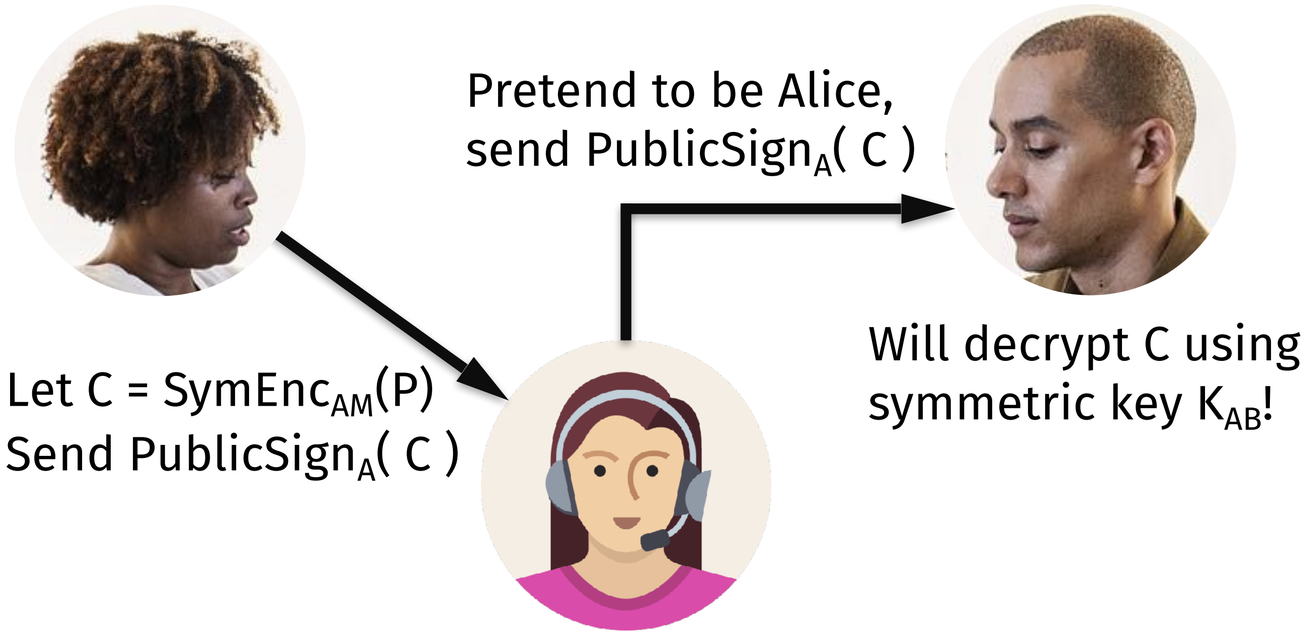
Combining symmetric encryption + public signatures

- In the symmetric case, we learned that Enc-then-MAC is the best option
  - Intuition: Never expose the decryption key to an invalid message
- Does this technique work as well with public key signatures?

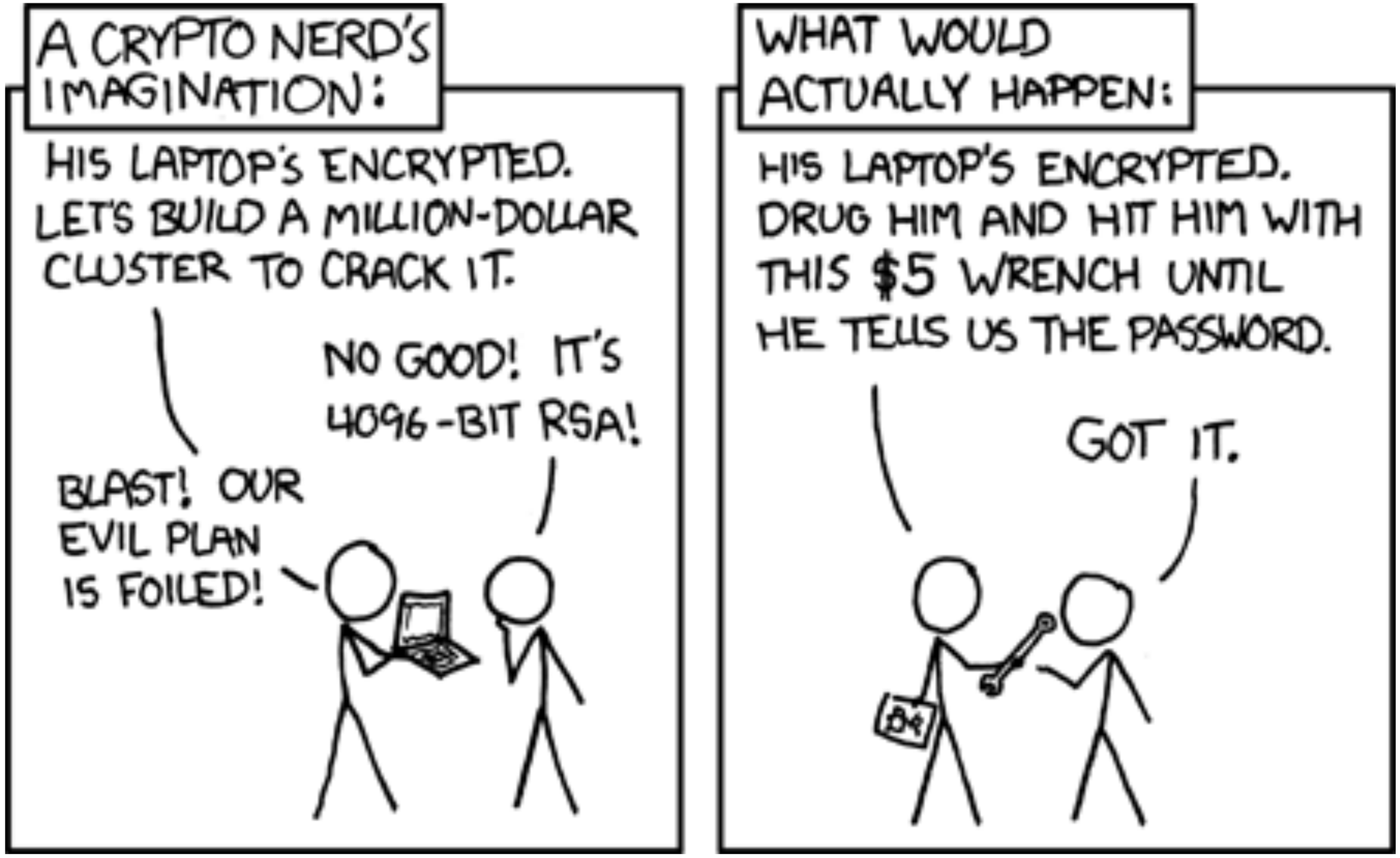


Combining symmetric encryption + public signatures

- In the symmetric case, we learned that Enc-then-MAC is the best option
  - Intuition: Never expose the decryption key to an invalid message
- Does this technique work as well with public key signatures?
- Answer: No!
  - Issue: Mallory can receive ciphertexts from Alice, claim them as her own!
- Can lead to an oracle attack, as occurs with Apple's iMessage



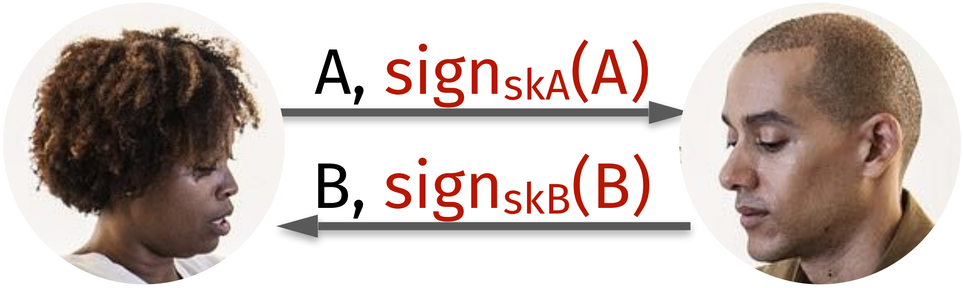
Non-repudiable crypto (xkcd.com/538)



Better combination of public signatures + symmetric crypto

Choose *a* randomly
Compute A = g^a

Choose *b* randomly
Compute B = g^b



Output B^a

Output A^b

Remaining question: how do Alice and Bob learn the other's public key?

1. Alice + Bob sign their D-H key exchange messages
2. Alice + Bob verify signatures on each others' messages
3. Use agreed-upon key for (deniable) symmetric authenticated encryption

Google.com in Firefox:

Technical Details

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

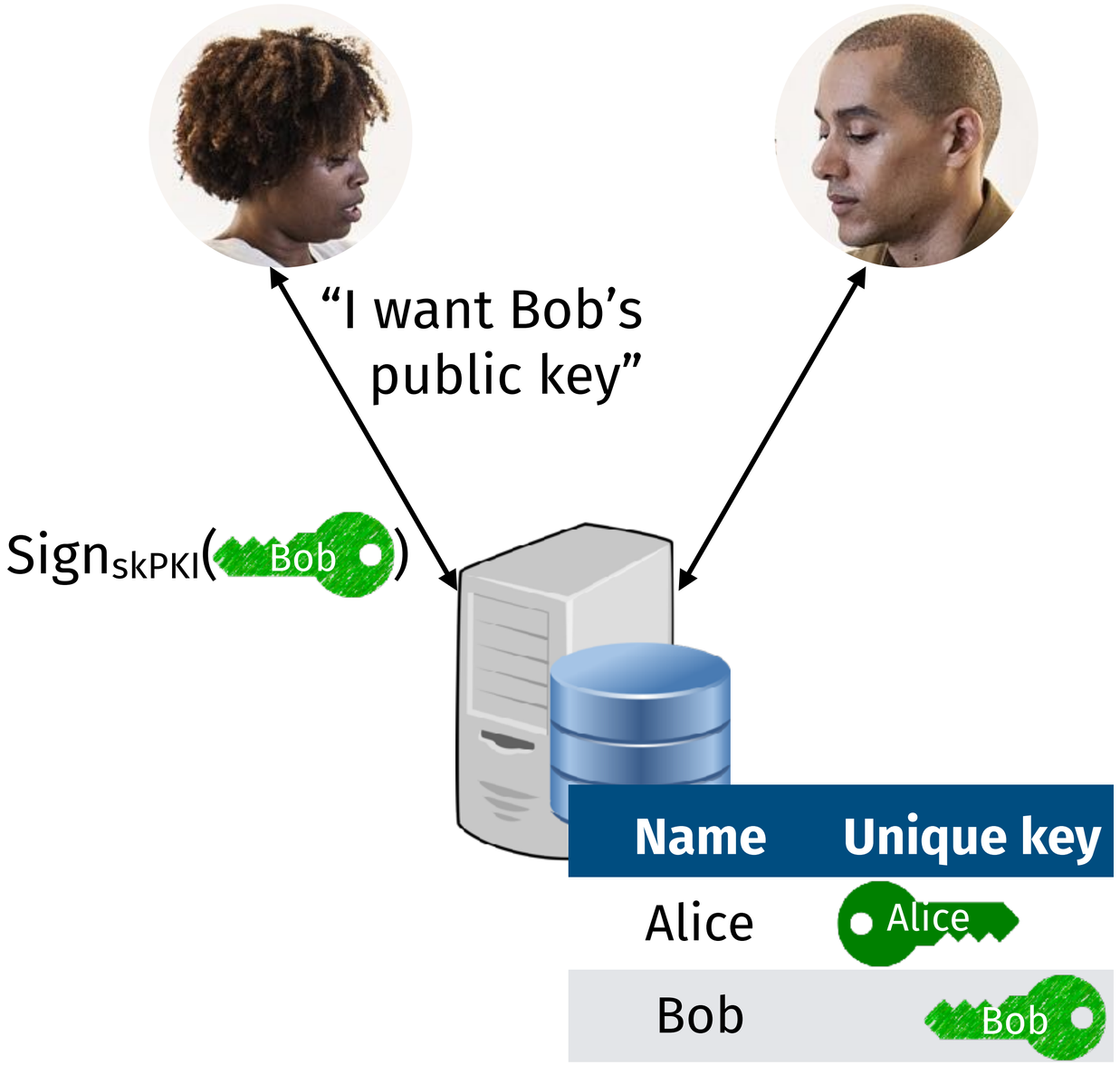
BU login page in Firefox (2017):

Technical Details

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

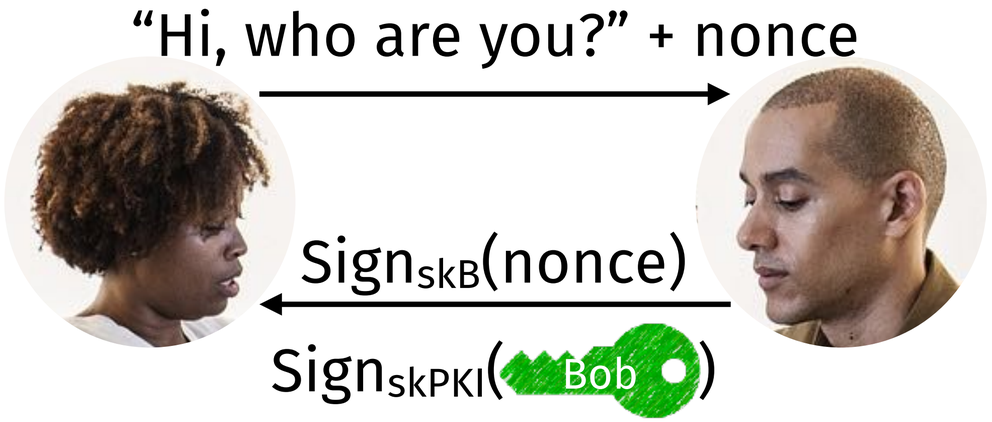
Public key infrastructure

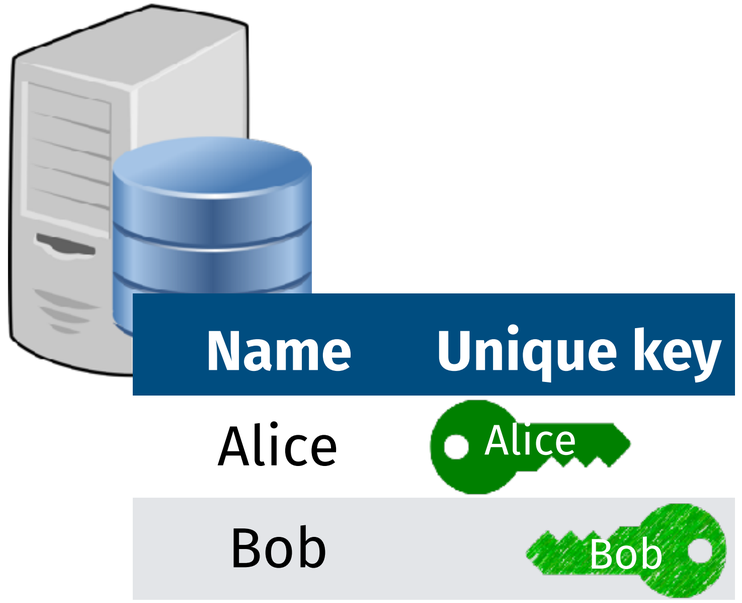
- There is a *certificate authority* that knows everybody's keys
  - (Think of it like a telephone book)
- Anyone can query the authority to learn someone else's key
- CA signs responses so that everybody knows they are legit
- Alice knows the CA's public key because it is included in her OS



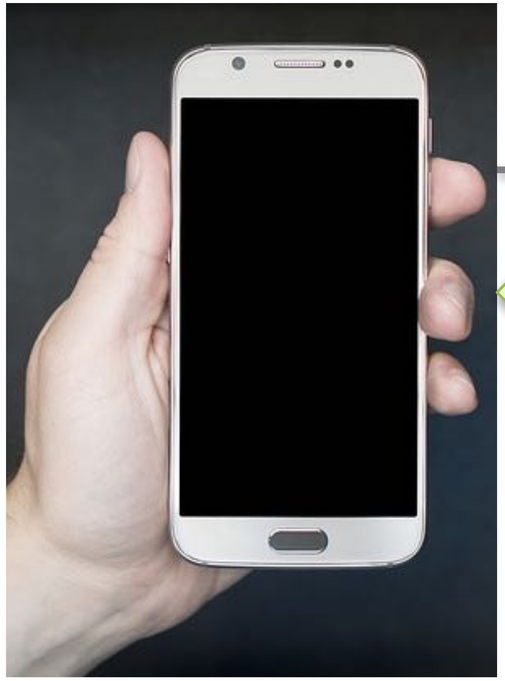
Slight improvement

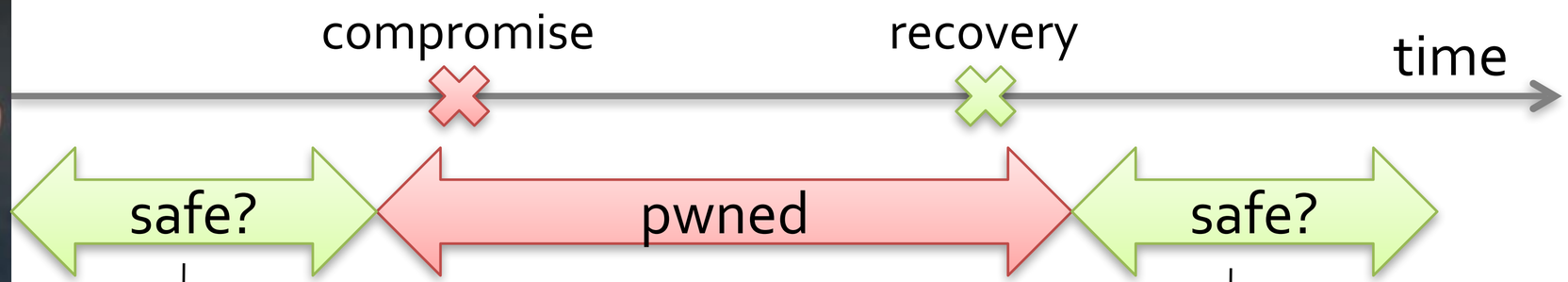
- Alice wants to talk to Bob, not CA
- Bob can forward CA's attestation that signing key belongs to him
- (Shown: simplified TLS handshake)





What happens if Bob's secret key *SK* is compromised?



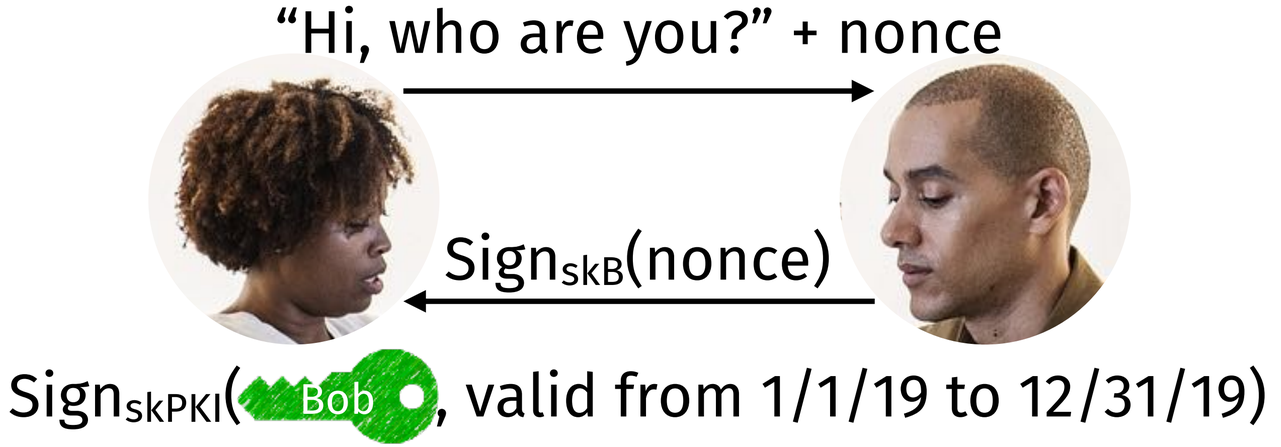


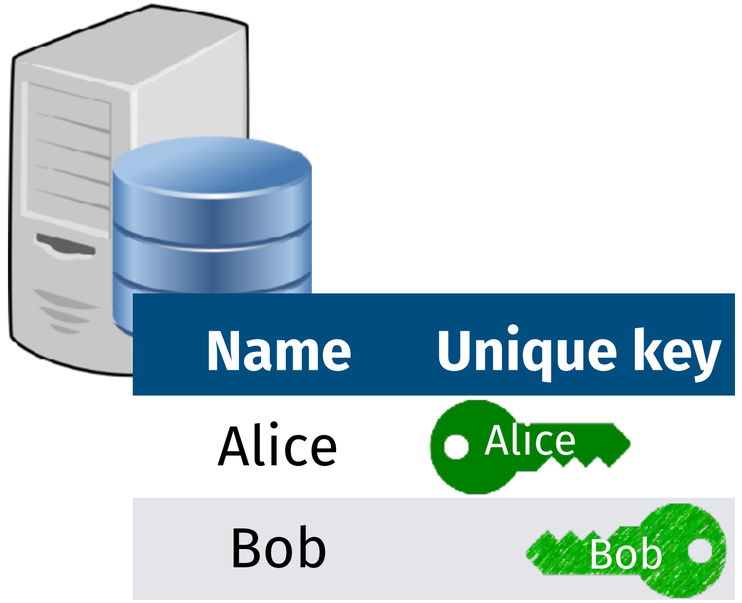
Yes!
Unless Mallory has a time machine, signatures that Alice verified before a breach must be valid.

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

Backward security technique #1: Cert expiration

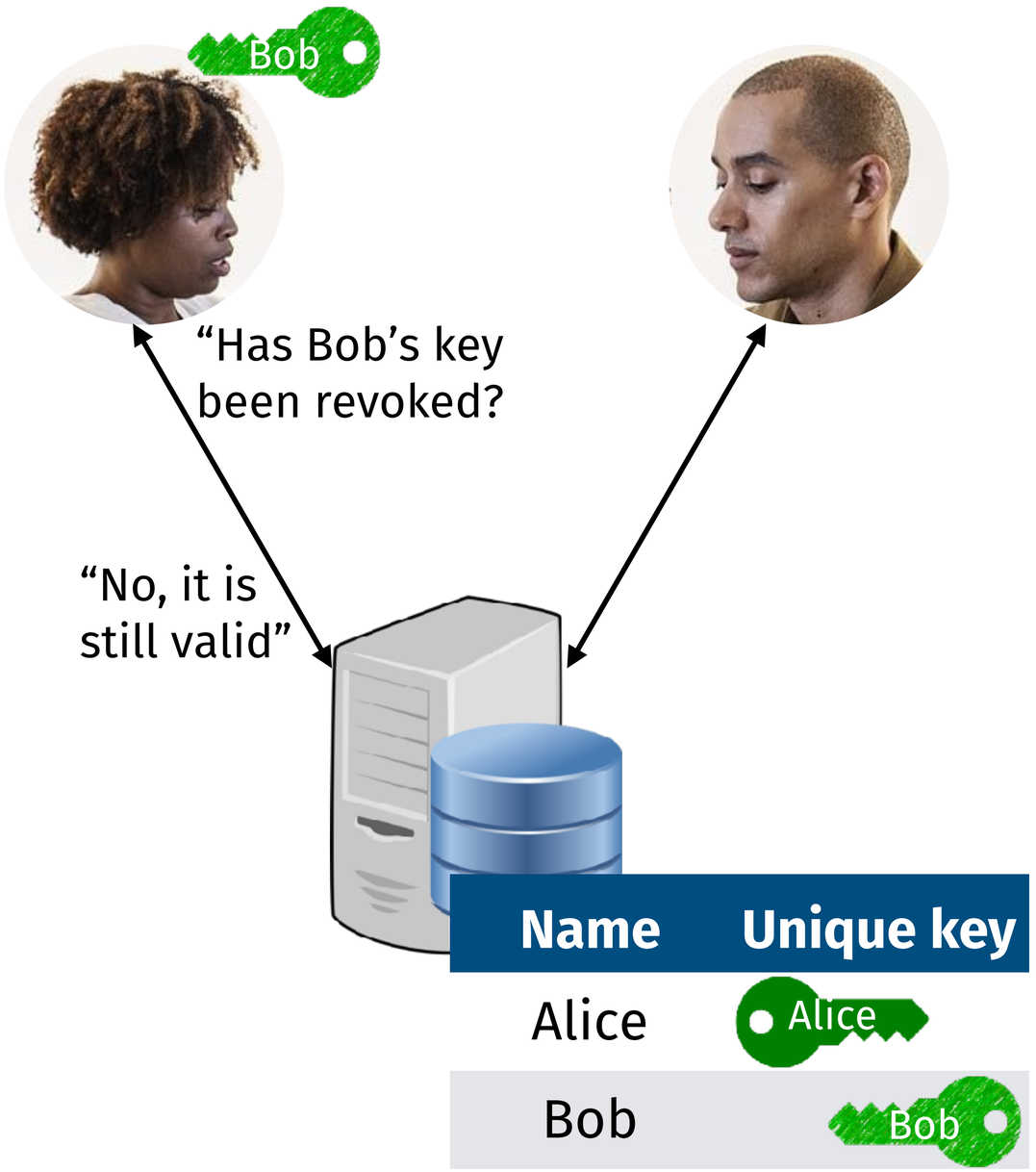
- Alice wants to talk to Bob, not CA
- Bob can forward CA's attestation that signing key belongs to him
- (Shown: simplified TLS handshake)





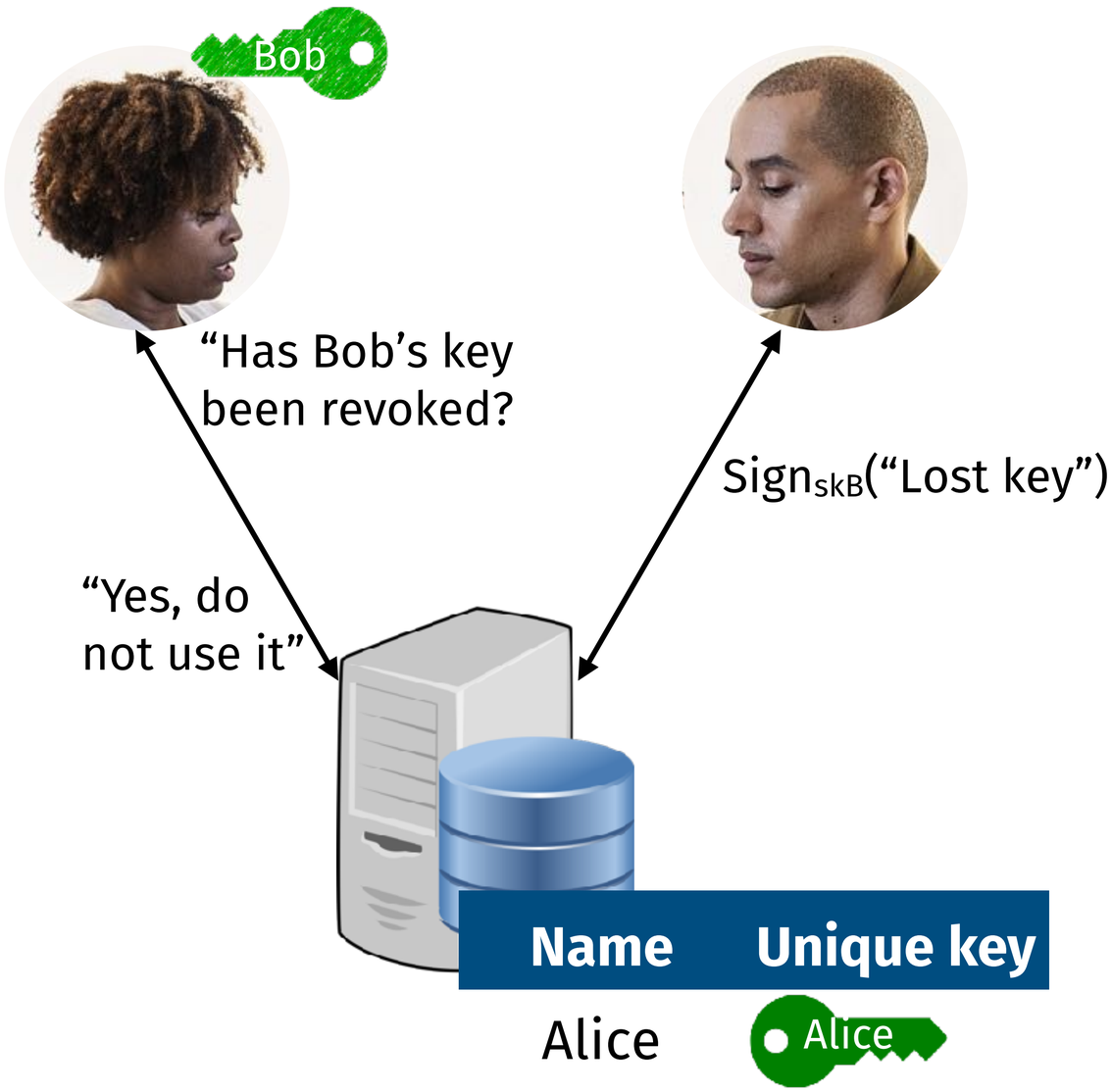
Backward security technique #2: Key revocation

- The PKI binds a public key to your identity
- 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: Key revocation

- The PKI binds a public key to your identity
- 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



Key management = initial exchange + subsequent evolution

Server trust?	Crypto used	Method
Full	Symmetric	Needham-Schroeder ⟹ Kerberos system
Partial	Asymmetric	(Authenticated) Diffie-Hellman key exchange
None	Symmetric	Key evolution, starting from an initial shared symmetric key

Next time: Key evolution