Lecture 4:
AKE, TLS Handshake

CPSC 444/544, 2026 Spring

Instructor: Fan Zhang

Yale

Computer Science

Logistics

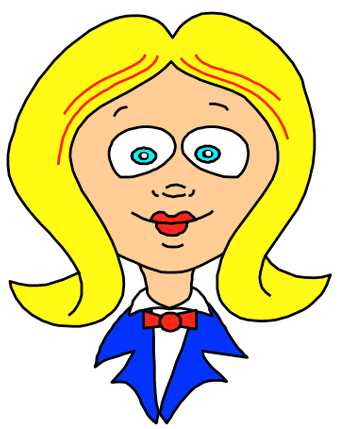
- Monday class will be **on Zoom**.
  - (I'll have research related travel.)
  - Zoom link will be provided on Canvas later
  - Lecture 3 notes published at https://www.fanzhang.me/teaching/444/l3/

Today

- Theory: Instantiating with PKE and Signature
- System: TLS handshake

Last time:AKE3

$sk, pk \leftarrow^{\$}$



P

$\mathsf{pk}, \mathsf{Cert}_P$

$c = \mathsf{E}(\mathsf{pk}, k), \sigma_1 = \mathsf{Sig}_Q(1, \mathsf{pk}, c, \mathsf{id}_P), \mathsf{Cert}_Q$

$\sigma_2 = \mathsf{Sig}_P(2, \mathsf{pk}, c, \mathsf{id}_Q)$

$k \leftarrow^{\$}$

Q

- Nice properties
  - ***Perfect Forward Secrecy (PFS)***: leaking the *long term* encryption key doesn't leak past session keys.
  - ***HSM Security***: even if the *ephemeral* key is leaked, attacker still need long-term signing key to run the protocol.
- Limitations: leaks who is talking to whom.

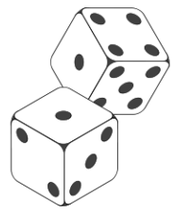
Identity Protection

- Privacy from Eavesdropper: P and Q's certs better be encrypted
- Full identity protection for P: only reveals identity to Q after P is sure who she is talking to.

- At least one party can't enjoy full identity protection
- Idea: encrypt certs (AKE4 next slide)

$sk, pk \leftarrow^{\$}$ 

pk

$k, k_1, k_2 \leftarrow^{\$}$ 

$$c = \mathsf{E}(\mathsf{pk}, (k, k_1, k_2)), c_1 = \mathsf{E}(k_1, \mathsf{Sig}_Q(1, \mathsf{pk}, c), \mathsf{Cert}_Q)$$

- PK.Decrypt c to get k, k_1, k_2
- Sym. key decrypt c_1 using k_1
- Verifies sig against Cert_Q and $1, pk, c$

$$c_2 = \mathsf{E}(k_2, \mathsf{Sig}_P(2, \mathsf{pk}, c), \mathsf{Cert}_P)$$

- Outputs k, and id_Q

- Decrypt c_2 using k_2
- Verifies sig against Cert_P and $2, pk, c$
- Output k, and id_P

Privacy from Eavesdropper for both. Full protection for P.

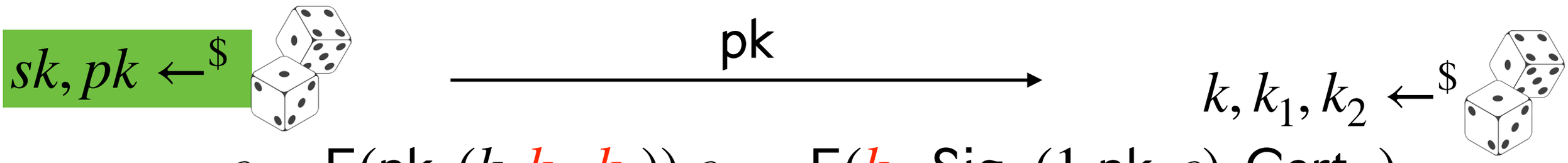
Pretty close to TLS 1.3 handshake!

AKE4 Security Intuition

Question 1: c_2 does not confirm that P will output id_Q (unlike AKE3). Is it possible that P outputs an identity ≠ Q?

Ruled out by CCA security of E. Whoever sent c_2 must have gotten the key k_2 from c, thus must have also gotten k_1. By CCA, it's impossible for adversary to create another message that decrypts using k_1, thus the sender of c_2 must know it's talking to Q.

AKE4 Security Intuition

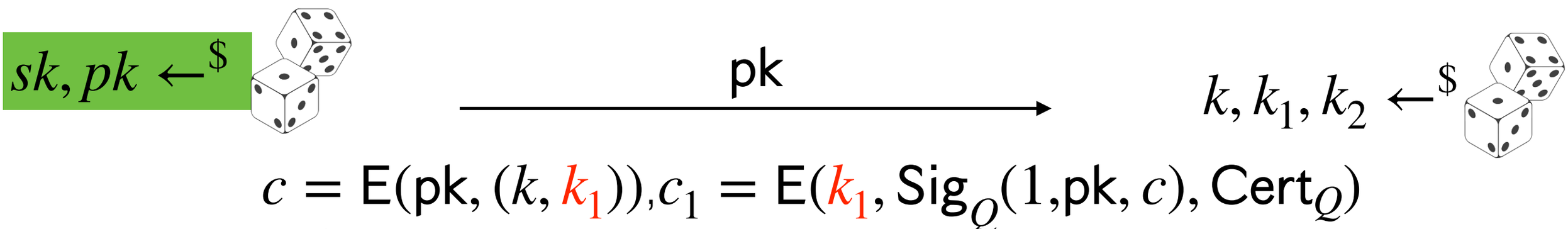


Follow-up Question:
Is it possible that Q outputs an identity ≠ P?

If E is CCA secure, whoever sent c_2 must know **sk** — either Adv or some party called P. Adv can only learn **sk** if she sends the first message, in which case Q will output A. Otherwise, Q outputs P.

One sided AKE4

Q do not care about authenticating P, common for web.



Next: We need PKE with CPA security, Symmetric-key Encryption with CCA security, and a secure Sig.

El Gamal:
a versatile PKE

You will see El Gamal in many applications (b/c it’s homomorphic)

El Gamal ('85)

- Fix $(\mathbb{G}, q, g)$ where DDH is hard
- Assuming message is space is $\mathbb{G}$

- $\mathsf{Gen}(1^n)$: $\mathsf{sk} \leftarrow^{\$} \mathbb{Z}_q$, $\mathsf{pk} = g^{\mathsf{sk}}$
- $\mathsf{Enc}(\mathsf{pk}, m)$: $y \leftarrow^{\$} \mathbb{Z}_p$, output $(g^y, \mathsf{pk}^y \circ m)$
  - Can view $\mathsf{pk}^y = g^{xy}$ as a DH key
- $\mathsf{Dec}(\mathsf{sk}, (c_1, c_2))$: c_2 / c_1^{sk}

Enc: 2 exps (fixed base)
Dec: 1 exp (var base)

IND-CPA security directly follows from DDH:
pk^y is indistinguishable from a uniform element.

El Gamal ('85)

- The previous scheme is **malleable**
  - Enc(pk, m)=(g^y, pk$^y \circ m$)
  - Enc(pk, m')=($g^{y'}$, pk$^{y'} \circ m'$)
  - From that we can get Enc(pk, $m \circ m'$)

- Malleability can be good or bad
  - Good: can compute without decrypting. We call it ***homomorphism***.
  - Bad: not CCA secure (e.g., if it encrypts a ballot, then attacker can change it)

Key-Encapsulation Mechanism (KEM)

- PKE are often used to encrypt fresh keys. KEM samples a key k then encrypt it in one shot.
  - $(pk, sk) \leftarrow^{\$} \mathsf{Gen}(1^n)$
  - $(c, k) \leftarrow^{\$} \mathsf{Encaps}(pk)$
  - $k \leftarrow \mathsf{Decaps}(sk, c)$
- CPA-Security: k is indistinguishable from a random key given (c, pk)
- CCA-Security: CPA + access to **Decaps**

El Gamal based KEM

- (pk, sk) $\leftarrow^{\$}$ Gen(1^n)
  - Same as El Gamal
- $(c, k) \leftarrow^{\$}$ Encaps(pk)
  - Samples $r \leftarrow^{\$} \mathbb{Z}_q$, output $c := g^r$ and $k = H(\text{pk}^r)$.
- $k \leftarrow$ Decaps(sk, c)
  - $k := H(c^{\text{sk}})$

- El Gamal KEM is CCA secure assuming ICDH and ROM.

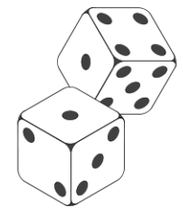
$(c, k) \leftarrow^{\$}$ Encaps(pk)

- Samples $r \leftarrow^{\$} \mathbb{Z}_q$, output $c := g^r$ and $k = H(\text{pk}^r)$

AKE4

$\text{sk}, \text{pk} \leftarrow^{\$}$ 

pk →

$\beta \leftarrow^{\$}$ 

$k, k_1, k_2 = H(\text{pk}^{\beta})$

← $c = g^{\beta}, c_1 =$ same as before

$c_2 =$ same as before →

Output $k, \_, \_ = H(\text{pk}^{\beta})$

Digital Signatures (Schnorr Family)

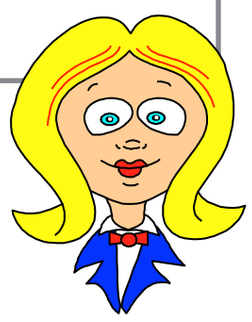

What's your favorite signature scheme?
Aren't they all the same?
imgflip.com

Schnorr ID Protocol

Schnorr ID scheme

Goal: Only Alice can authenticate; Bob doesn't learn anything more about y.

Private / Public Keys:
$(x, y = g^x)$



Bob = adv.

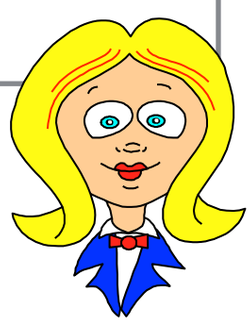
Alice's public key: y

E.g., Bob is a card reader

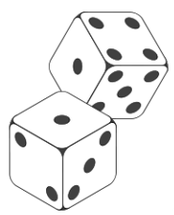
First try

Problem?

Private / Public Keys:
$(x, y = g^x)$



r

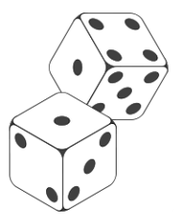


Alice's public key: y

$s = rx$

$$g^s \stackrel{?}{=} y^r$$

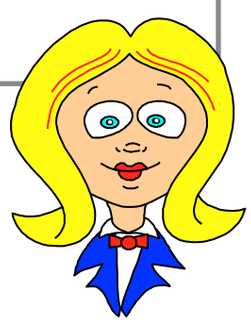
Second try

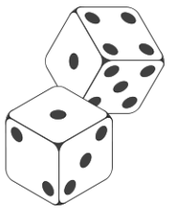


r

Alice's public key: y

Private / Public Keys:
$(x, y = g^x)$





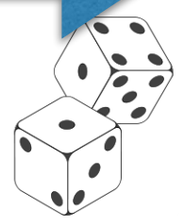
One-time private key: k

$s = rx + k$

???

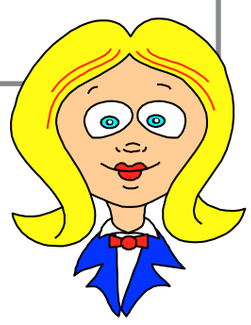
Schnorr's ID scheme

I

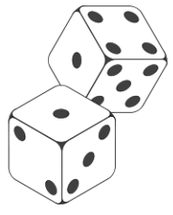


r

Private / Public Keys:
$(x, y = g^x)$



Alice's public key: *A*



One-time private key: k

$s = rx + k$

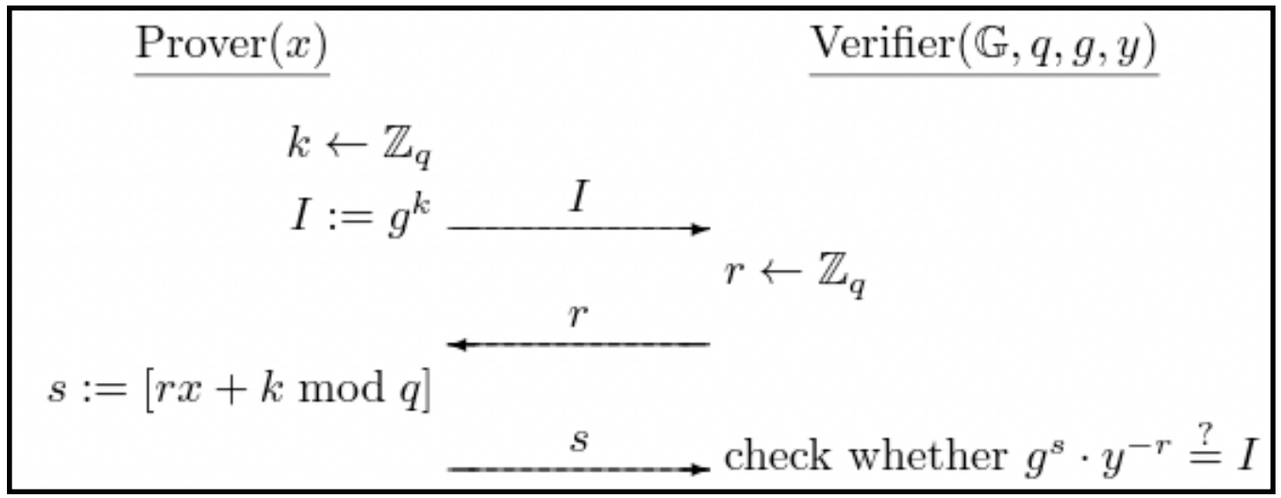
Blinding factor: $I = g^k$

$$g^s \stackrel{?}{=} I \circ y^r$$

Intuition:

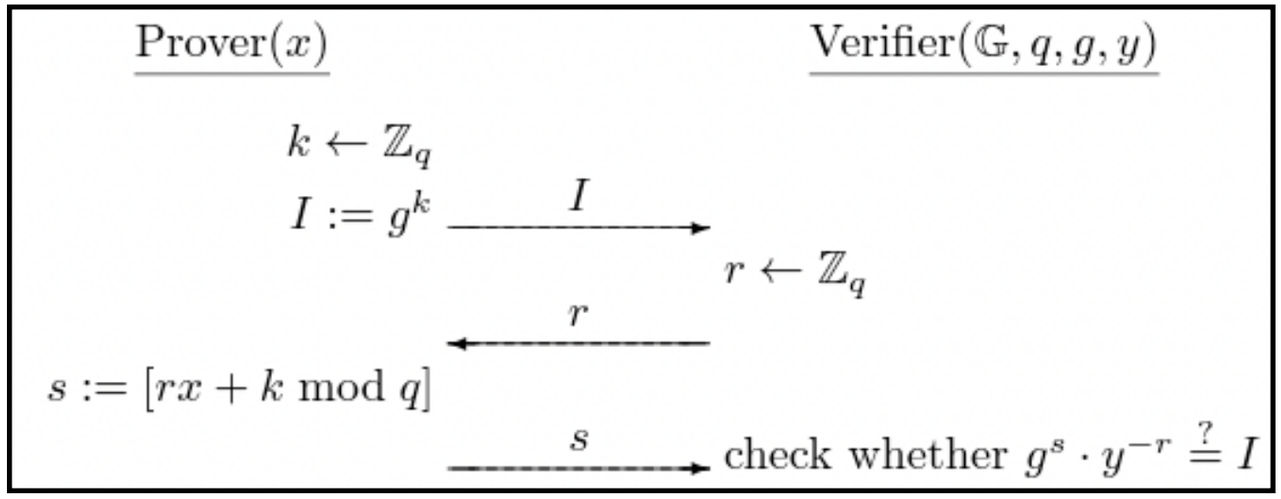
- Bob can verify that *x* is properly "mixed in"—so it's really Alice.
- Bob doesn't learn private key: *k* is a one-time value that blinds *x*.
- Can we slip the first two steps?

Soundness proof



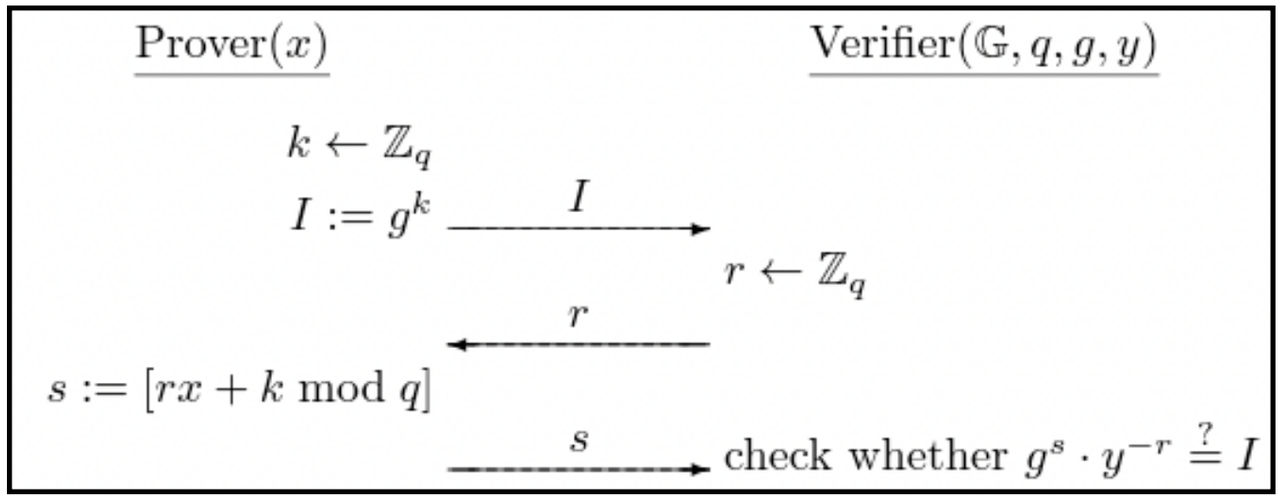
- Goal: show that if $\mathscr{A}$ (a program) can authenticate as Alice without knowing x, we can use $\mathscr{A}$ to solve DL of y (which is g^x).
- Fix I, given r_1, r_2, $\mathscr{A}$ can output s_1, s_2 such that
  - $g^{s_1} \circ y^{-r_1} = I = g^{s_2} \circ y^{-r_2}$
  - From this: $x = (s_1 - s_2)(r_1 - r_2)^{-1} \mod q$
- This property is known as "**special soundness**" [KL Theorem 13.11]

“Secrecy”



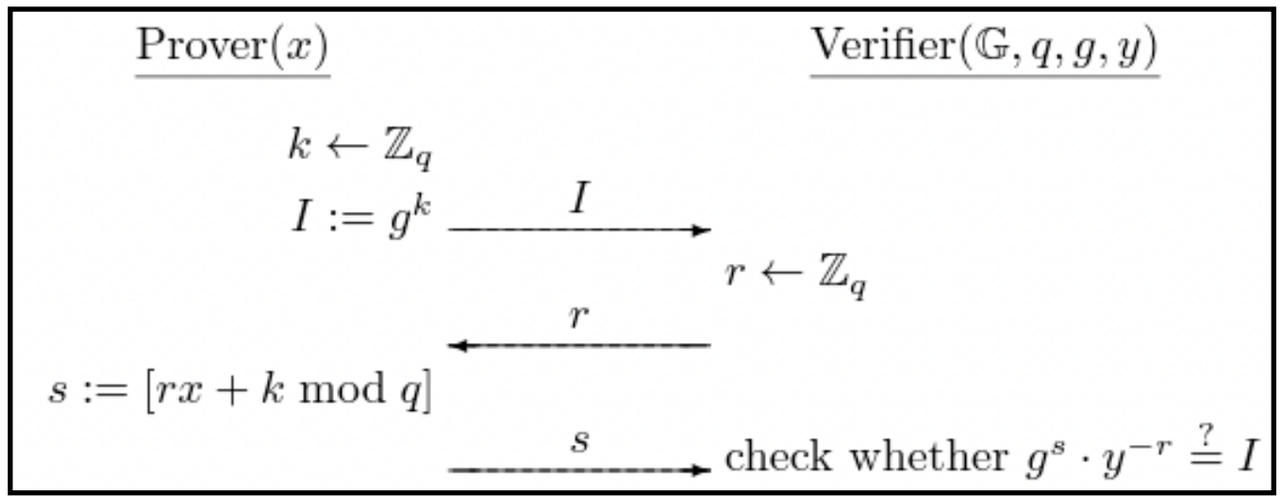
- Intuitively, the verifier should not learn ***anything*** it couldn’t have computed without interacting with the prover.
- This property has a cool name “zero-knowledge”
- Formal definition via **simulation**:
  - Let $\text{View}_V^{P(x)}$ be the verifier’s view when interacting with a prover who knows secret x.
  - If there exists a simulator S, that knows no secret, such that: $\text{View}_V^{P(x)} \approx S(\text{public input})$ then the interaction reveals no extra information.

“Zero-knowledge”



- $\text{View}_V^{P(x)}$: (I, r, s) two uniformly random, the third implied.
- Simulator: sample s, r from $\mathbb{Z}_q$, compute $I = g^s \circ y^{-r}$
  - (I, r, s) output by S is ***identically distributed*** as $\text{View}_V^{P(x)}$
- This is called “special honest-verifier zero-knowledge”
  - Malicious V* can’t be simulated in general. E.g., V* can choose r=Hash(I), now you can see that simulation is impossible.
  - To fix - need Random Oracle Model.

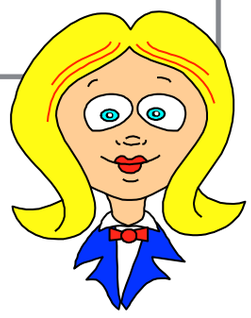
Why is this not a signature?



- Where is message?
- Interactive
- Not transferrable: Presenting (I, r, s) to a third party is not convincing evidence that you know x

Schnorr ID → signature

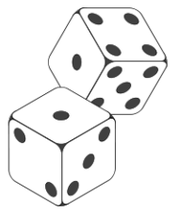
Private / Public Keys:
$(x, y = g^x)$



$r = H(I, m)$

Alice's public key: y

Signature:
$(I, s = rx + k)$



One-time private key: k

Public key: $I = g^k$

Check $I \stackrel{?}{=} g^s y^{-H(I,m)}$

Intuition:

- Generate "challenge" r from m and I using **a hash function**
- Still doesn't allow Alice to select I after seeing r

Schnorr ID → signature

Private / Public Keys:
$(x, y = g^x)$

$r = H(I, m)$

Alice's public key: y

Signature:
$(I, s = rx + k)$

One-time private key: k

Public key: $I = g^k$

Check $I \stackrel{?}{=} g^s y^{-H(I,m)}$

Does this really have to be one-time?

One way to enforce uniqueness: k = H(sk ll M)

Schnorr signature scheme

- Gen(1^n):
  - $sk \leftarrow^{\$} \mathbb{Z}_q, pk = g^{sk}$
  - Output (sk, pk)
- Sign(sk, m):
  - $k \leftarrow^{\$} Z_q, I = g^k$
  - $s = H(I, m) \cdot x + k \mod q$
  - Output $\sigma = (I, s)$
- Verify(pk, m, σ):
  - $(I, s) = \sigma$
  - Output $I \stackrel{?}{=} g^s y^{-H(I,m)}$

- Simple, easy to prove security by DL assumption in the ROM.
- Nice algebraic structure:
  - E.g., supports aggregation (multi keys/sigs can be combined to one).

Batch Ver: https://alinush.github.io/2024/05/31/Schnorr-signatures.html

Other common sigs

- E.g., ssh-keygen -t give you the following options: ECDSA, ED25519, RSA

```
SSH-KEYGEN(1)                   General Commands Manual                  SSH-KEYGEN(1)

NAME
     ssh-keygen — OpenSSH authentication key utility

SYNOPSIS
     ssh-keygen [-q] [-a rounds] [-b bits] [-C comment] [-f output_keyfile]
                [-m format] [-N new_passphrase] [-O option]
                [-t ecdsa | ecdsa-sk | ed25519 | ed25519-sk | rsa]
```

RSA signature

- RSA signatures are widely used still but not loved anymore:
  - RSA public key is long.
  - RSA is error prone to implement.

https://blog.trailofbits.com/2019/07/08/fuck-rsa/

Schnorr-derived schemes

- Schnorr is patented until 2008, which slows its adoption. ECDSA and EdDSA are much more widely used.
- ECDSA: A variant of Schnorr by NIST (2000). K is generated from RNG; Very **sensitive** to the quality of randomness.
  - Still widely used because it was standardized early and embedded into global infrastructure before its weaknesses were fully understood.

ECDSA is fragile

- Using a predictable value, or leaking even a few bits of k in each of several signatures, is enough to reveal the private key.

LadderLeak: Breaking ECDSA With Less Than One Bit Of Nonce Leakage

Diego F. Aranha
DIGIT, Aarhus University
Denmark
dfaranha@eng.au.dk

Felipe Rodrigues Novaes
University of Campinas
Brazil
ra135663@students.ic.unicamp.br

Akira Takahashi
DIGIT, Aarhus University
Denmark
takahashi@cs.au.dk

Mehdi Tibouchi
NTT Corporation
Japan
mehdi.tibouchi.br@hco.ntt.co.jp

Yuval Yarom
University of Adelaide and Data61
Australia
yval@cs.adelaide.edu.au

ACM CCS'20

https://blog.trailofbits.com/2020/06/11/ecdsa-handle-with-care/

Schnorr-derived schemes

- Ed25519: IETF standardized (2017) Schnorr-style sigs over the Edwards 25519 curve with **deterministic nonces**.
- How about the OG Schnorr?
  - Bitcoin: BIP-340 is the most complete real-world Schnorr standard today.

Next time, use Schnorr

- ssh-keygen -t **ed25519**

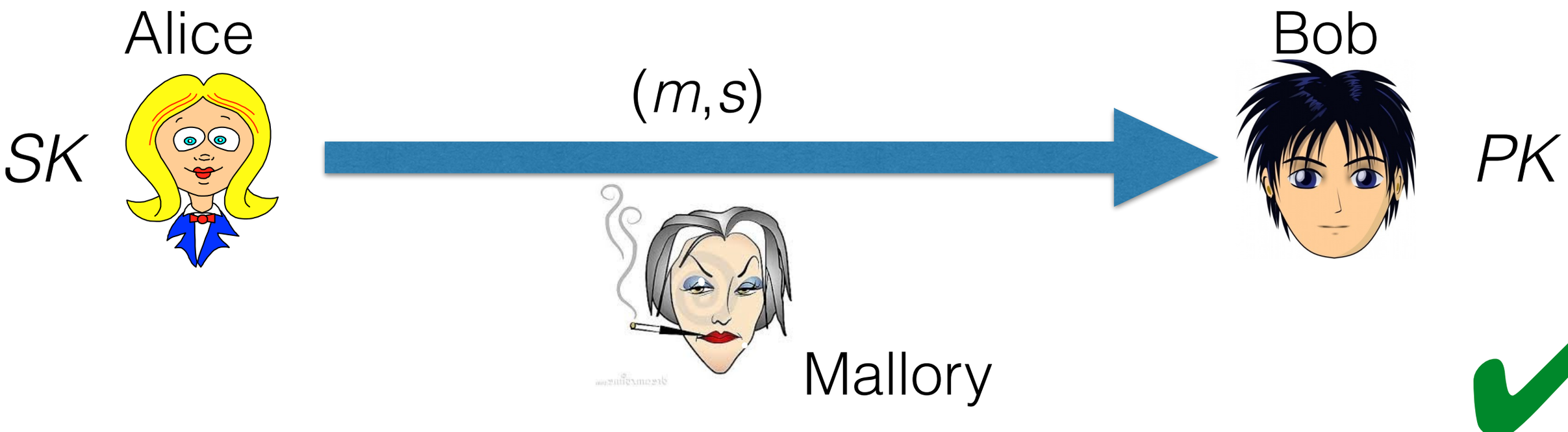
```
SSH-KEYGEN(1)                General Commands Manual                SSH-KEYGEN(1)

NAME
     ssh-keygen — OpenSSH authentication key utility

SYNOPSIS
     ssh-keygen [-q] [-a rounds] [-b bits] [-C comment] [-f output_keyfile]
                [-m format] [-N new_passphrase] [-O option]
                [-t ecdsa | ecdsa-sk | ed25519 | ed25519-sk | rsa]
```

Quiz: Digital signatures v.s. MAC

- MACed message v.s. a signed message
- Signed messages has
  - Publicly verifiability
  - Transferability
  - Non-repudiation
- Can be good or bad (e.g., deniability)



Digital signatures are legally binding by E-Sign Act (2000)

ELECTRONIC SIGNATURES IN GLOBAL AND NATIONAL COMMERCE ACT

The Consumer Consent Provision in Section 101(c)(1)(C)(ii)

FEDERAL TRADE COMMISSION
Bureau of Consumer Protection

DEPARTMENT OF COMMERCE
National Telecommunications and Information Administration

1. a name typed at the end of an e-mail message by the sender,
2. a digitized image of a handwritten signature that is attached to an electronic document,
3. a secret password or PIN to identify the sender to the recipient,
4. a mouse click, such as on an “I accept” button,
5. a sound, such as the sound created by pressing ‘9’ on a phone,
6. a cryptographic digital signature.