Thus far, only considered passive adversaries, but in reality, adversaries can be malicious

no man-in-the-middle protection

- Adversary can impersonate server (e.g., phishing) and then try to authenticate as client (but cannot interact with client during auth.)
- All protocols thus far are vulnerable [all consist of client sending token that server checks, which can be extracted by active adversary]
- For active security, we use challenge-response

Signature-based challenge-response
- Server stores a verification key vk for digital signature scheme
- Client holds signing key sk

random message m

client (sk) $\qquad\qquad$ server (vk)

$$m \xleftarrow{R} \mathcal{M}$$

$$\xleftarrow{\qquad\qquad\qquad}$$

$$\xrightarrow{\ \sigma \leftarrow \mathrm{Sign}(sk, m)\ }$$

$\downarrow$

check that $\mathrm{Verify}(vk, m, \sigma) = 1$

Server asks client to sign a random message
↳ Client's signature indicates proof of possession of sk associated with vk
↳ Active adversary that interacts with the client before interacting with the prover cannot forge signatures

Provides active security but signatures are long (~ 384 bits)

Signature-based challenge response: client "demonstrates knowledge" of signing key
↳ we will generalize this to "proving" arbitrary statements

Identification protocols such as Schnorr's protocol can be readily compiled into a digital signature scheme
↳ This will give us a signature scheme from the discrete log assumption

Key idea: Replace the verifier's challenge with a hash function $H: \{0,1\}^* \to \mathbb{Z}_p$ [outputs must be random-looking]

Namely, instead of sampling $c \xleftarrow{R} \mathbb{Z}_p$, we sample $c \leftarrow H(g, h, u)$. ← prover can now compute this quantity on its own!

The signature is then the protocol transcript (u, c, z) which anyone can check

Issue: Where does the message go? In the hash function!

Signatures from discrete log in random oracle model (Schnorr)

- Setup: $x \xleftarrow{R} \mathbb{Z}_p$

  $vk: (g, h = g^x)$ $\qquad$ $sk: x$

- $\mathrm{Sign}(sk, m)$: $r \xleftarrow{R} \mathbb{Z}_p$

  $u \leftarrow g^r$ $\qquad$ $c \leftarrow H(g, h, u, m)$ $\qquad$ $z \leftarrow r + cx$

  $\sigma = (u, z)$

  } signature is a proof of knowledge of discrete log of h (with challenge derived from the message m)

  as we will see, this is in fact a "zero-knowledge" proof

- $\mathrm{Verify}(vk, m, \sigma)$: write $\sigma = (u, z)$, $vk = h$, compute $c \leftarrow H(g, h, u, m)$ and accept if $g^z = u \cdot h^c$

Security essentially follows from security of Schnorr's identification protocol (together with Fiat-Shamir)
↳ forged signature on a new message m is a proof of knowledge of the discrete log (can be extracted from adversary)

Length of Schnorr's signature: vk: $(g, h = g^x)$ $\quad \sigma: (g^r, \underbrace{c = H(g, h, g^r, m)}, \; z = r + cx)$ Verification checks that $g^z = g^r h^c$
sk: x

can be computed given other components, so do not need to include $\Rightarrow |\sigma| = 2 \cdot |\mathbb{G}|$ [512 bits if $|\mathbb{G}| = 2^{256}$]

But, can do better... observe that challenge c only needs to be 128-bits (the knowledge error of Schnorr is $1/|C|$ where C is the set of possible challenges), so we can sample a 128-bit challenge rather than 256-bit challenge. Thus, instead of sending (g^r, z), instead send (c, z) and compute $g^r = g^z / h^c$ and that $c = H(g, h, g^r, m)$. Then resulting signatures are 384 bits

128 bit challenge
+
256 bit group element

Important note: Schnorr signatures are randomized, and security relies on having good randomness
↳ What happens if randomness is reused for two different signatures?
Then, we have

$$\left.\begin{aligned} \sigma_1 &= (g^r, \; c_1 = H(g, h, g^r, m_1), \; z_1 = r + c_1 x) \\ \sigma_2 &= (g^r, \; c_2 = H(g, h, g^r, m_2), \; z_2 = r + c_2 x) \end{aligned}\right\} \; z_1 - z_2 = (c_1 - c_2)x \implies x = (c_1 - c_2)^{-1}(z_1 - z_2)$$

This is precisely the set of relations the knowledge extractor uses to recover the discrete log x (i.e., the signing key)!

Deterministic Schnorr: We want to replace the random value $r \xleftarrow{R} \mathbb{Z}_p$ with one that is deterministic, but which does not compromise security
↳ Derive randomness from message using a PRF. In particular, signing key includes a secret PRF key k, and signing algorithm computes $r \leftarrow F(k, m)$ and $\sigma \leftarrow \mathrm{Sign}(sk, m; r)$.
↳ Avoids randomness reuse/misuse vulnerabilities.

In practice, we use a variant of Schnorr's signature scheme called $\overbrace{\text{DSA / ECDSA}}^{\text{digital signature algorithm / elliptic-curve DSA}}$

TLS protocol

↳ larger signatures (2 group elements – 512 bits) and proof only in "generic group" model [but we use it because Schnorr was patented ... until 2008]

ECDSA signatures (over a group $\mathbb{G}$ of prime order p):
- Setup: $x \xleftarrow{R} \mathbb{Z}_p$
  vk: $(g, h = g^x)$ $\quad$ sk: x
- Sign(sk, m): $\alpha \xleftarrow{R} \mathbb{Z}_p$
  $u \leftarrow g^\alpha$ $\quad r \leftarrow f(u) \in \mathbb{Z}_p$ (deterministic function specified by ECDSA)
  $s \leftarrow (H(m) + r \cdot x)/\alpha \in \mathbb{Z}_p$
  $\sigma = (r, s)$

[Specifically, $f(u)$ parses $u = (\hat{x}, \hat{y}) \in \mathbb{F}_q^2$ where $\mathbb{F}_q$ is the base field over which the elliptic curve is defined, and outputs $\hat{x} \pmod p$, where $\hat{x}$ is viewed as a value in $[0, q)$]

- Verify (vk, m, σ): write $\sigma = (r, s)$, compute $u \leftarrow g^{H(m)/s} h^{r/s}$, accept if $r = f(u)$
  vk = h

Correctness: $u = g^{H(m)/s} h^{r/s} = g^{[H(m) + rx]/s} = g^{[H(m)+rx]/[H(m)+rx]\alpha^{-1}} = g^\alpha$ and $r = f(g^\alpha)$
Security analysis non-trivial: requires either strong assumptions or modeling $\mathbb{G}$ as an "ideal" group
Signature size: $\sigma = (r, s) \in \mathbb{Z}_p^2$ — for 128-bit security, $p \sim 2^{256}$ so $|\sigma| = 512$ bits (can use P-256 or Curve25519)