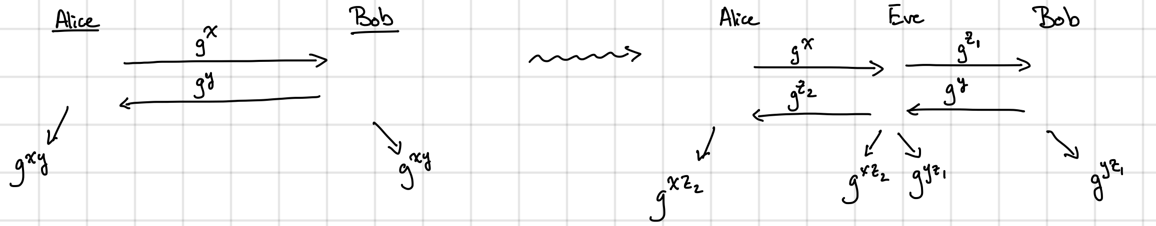
Diffie-Hellman key-exchange is an anonymous key-exchange protocol: neither side knows who they are talking to
↳ vulnerable to a "man-in-the-middle" attack



Observe Eve can now decrypt all of the messages between Alice and Bob and Alice + Bob have no idea!

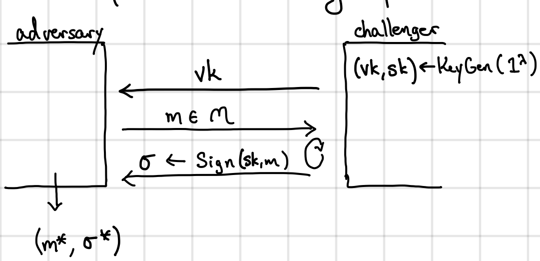
What we require: authenticated key-exchange (not anonymous) and relies on a root of trust (e.g., a certificate authority)
↳ On the web, one of the parties will authenticate themself by presenting a certificate

To build authenticated key-exchange, we require more ingredients — namely, an integrity mechanism [e.g., a way to bind a message to a sender — a "public-key MAC" or digital signature]

We will revisit when discussing the TLS protocol

Digital signature scheme: Consists of three algorithms:
- $\text{Setup}(1^\lambda) \to (vk, sk)$: Outputs a verification key vk and a signing key sk
- $\text{Sign}(sk, m) \to \sigma$: Takes the signing key sk and a message m and outputs a signature σ
- $\text{Verify}(vk, m, \sigma) \to 0/1$: Takes the verification key vk, a message m, and a signature σ, and outputs a bit $0/1$

Two requirements:
- Correctness: For all messages $m \in \mathcal{M}$, $(vk, sk) \leftarrow \text{KeyGen}(1^\lambda)$, then
$$\Pr[\text{Verify}(vk, m, \text{Sign}(sk, m)) = 1] = 1.$$
[Honestly-generated signatures always verify]
- Unforgeability: Very similar to MAC security. For all efficient adversaries $\mathcal{A}$, $\text{SigAdv}[\mathcal{A}] = \Pr[W = 1] = \text{negl}(\lambda)$, where W is the output of the following experiment:



Let $m_1, \dots, m_Q$ be the signing queries the adversary submits to the challenger. Then, $W = 1$ if and only if:
$$\text{Verify}(vk, m^*, \sigma^*) = 1 \quad \text{and} \quad m^* \notin \{m_1, \dots, m_Q\}$$
Adversary cannot produce a valid signature on a new message.

Exact analog of a MAC (slightly weaker unforgeability: require adversary to not be able to forge signature on new message)
↳ MAC security required that no forgery is possible on any message [needed for authenticated encryption]

digital signature algorithm — elliptic-curve DSA } standards (widely used on the web — e.g., TLS)

It is possible to build digital signatures from discrete log based assumptions (DSA, ECDSA)
↳ But construction not intuitive until we see zero knowledge proofs
↳ We will first construct from RSA (trapdoor permutations)

We will now introduce some facts on composite-order groups:

Let $N = pq$ be a product of two primes p, q. Then, $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$ is the additive group of integers modulo N. Let $\mathbb{Z}_N^*$ be the set of integers that are invertible (under multiplication) modulo N.

$$x \in \mathbb{Z}_N^* \text{ if and only if } \gcd(x, N) = 1$$

Since $N = pq$ and p, q are prime, $\gcd(x, N) = 1$ unless x is a multiple of p or q:

$$|\mathbb{Z}_N^*| = N - p - q + 1 = pq - p - q + 1 = (p-1)(q-1) = \varphi(N)$$

Euler's phi function (Euler's totient function)

Recall Lagrange's Theorem:

for all $x \in \mathbb{Z}_N^*$: $x^{\varphi(N)} = 1 \pmod N$ [called Euler's theorem, but special case of Lagrange's theorem]

important: "ring of exponents" operate modulo $\varphi(N) = (p-1)(q-1)$

Hard problems in composite-order groups:

- Factoring: given $N = pq$ where p and q are sampled from a suitable distribution over primes, output p, q
- Computing cube roots: Sample random $x \xleftarrow{R} \mathbb{Z}_N^*$. Given $y = x^3 \pmod N$, compute $x \pmod N$.
  - ↳ This problem is easy in $\mathbb{Z}_p^*$ (when $3 \nmid p-1$). Namely, compute $3^{-1} \pmod{p-1}$, say using Euclid's algorithm, and then compute $y^{3^{-1}} \pmod p = (x^3)^{3^{-1}} \pmod p = x \pmod p$.
  - ↳ Why does this procedure not work in $\mathbb{Z}_N^*$. Above procedure relies on computing $3^{-1} \pmod{|\mathbb{Z}_N^*|} = 3^{-1} \pmod{\varphi(N)}$. But we do not know $\varphi(N)$ and computing $\varphi(N)$ is as hard as factoring N. In particular, if we know N and $\varphi(N)$, then we can write

$$\begin{cases} N = pq \\ \varphi(N) = (p-1)(q-1) \end{cases}$$

[both relations hold over the integers]

and solve this system of equations over the integers (and recover p, q)

Hardness of computing cube roots is the basis of the RSA assumption:

RSA assumption: Take $p, q \leftarrow \underbrace{\text{Primes}(1^\lambda)}_{\text{distribution over prime numbers.}}$, and set $N = pq$. Then, for all efficient adversaries A,

$$\Pr[x \xleftarrow{R} \mathbb{Z}_N^* ; y \leftarrow A(N, x) : y^3 = x] = \text{negl}(\lambda)$$

more generally, can replace 3 with any e where $\gcd(e, \varphi(N)) = 1$

common choices:

$e = 3$

$e = 65537$

↳ Hardness of RSA relies on $\varphi(N)$ being hard to compute, and thus, on hardness of factoring (Reverse direction factoring $\overset{?}{\Longrightarrow}$ RSA is not known)

Hardness of factoring / RSA assumption:

- Best attack based on general number field sieve (GNFS) — runs in time $\sim 2^{\tilde{O}(\sqrt[3]{\log N})}$ (same algorithm used to break discrete log over $\mathbb{Z}_p^*$)
- For 112-bits of security, use RSA-2048 (N is product of two 1024-bit primes)
  128-bits of security, use RSA-3072

  large key-sizes and computational cost $\Rightarrow$ ECC generally preferred over RSA
- Both prime factors should have similar bit length (ECM algorithm factors in time that scales with smaller factor)

Naive approach (common "textbook" approach) to build signatures:

Setup: Sample (N, e, d) where $N = pq$ and $ed = 1 \pmod{\varphi(N)}$

Output $vk = (N, e)$ and $sk = d$

$\text{Sign}(\overset{=d}{sk}, m)$: Output $\sigma \leftarrow m^d \pmod N$

$\text{Verify}(\overset{=e}{vk}, m, \sigma)$: Output 1 if $\sigma^e = m \pmod N$

Looks tempting (and simple)... but totally broken!

Correctness: Suppose $\sigma = m^d$. Then $\sigma^e = (m^d)^e = m^{ed \bmod \varphi(N)}$
$= m \pmod N$

Security: Signature on m is an e^{th} root of m — security should follow from RSA
FALSE!

↳ This is not true! RSA says that computing e^{th} root of random $x \in \mathbb{Z}_N^*$ is hard, not that it is hard for all inputs $x \in \mathbb{Z}_N^*$. But in the case of signatures, the message is the input. This is not only not random, but in fact, adversarially chosen!

↳ Very easy to attack. Consider the 0-query adversary:

Given verification key $vk = (N, e)$, take any $\sigma \in \mathbb{Z}_N^*$ and compute $m = \sigma^e \pmod N$.

By construction, σ is a valid signature on m

Signatures from RSA (the full domain hash):

In order to appeal to RSA, we need the signature to be an e^{th} root of a random value

Idea: hash the message first and sign the hash value (often called "hash-and-sign")

↳ Another benefit: Allows signing long messages (much larger than $\mathbb{Z}_N^*$)

RSA-FDH signatures:

Setup: Sample modulus N, e, d such that $ed = 1 \pmod{\varphi(N)}$ — typically $e = 3$ or $e = 65537$

Some (partial) attacks can exploit very small public exponent ($e=3$)

Output $vk = (N, e)$ and $sk = (N, d)$

$\text{Sign}(sk, m)$: $\sigma \leftarrow H(m)^d$ [Here, we are assuming that H maps into $\mathbb{Z}_N^*$]

$\text{Verify}(vk, m, \sigma)$: output 1 if $H(m) = \sigma^e$ and 0 otherwise

from $\{0,1\}^*$ to $\mathbb{Z}_N^*$

Theorem. Under the RSA assumption and modeling H as an ideal hash function (i.e., "random oracle") then RSA-FDH is a secure digital signature scheme.

Proof Idea: Signature is deterministic, so to succeed, adversary has to forge on an unqueried message m.

Signature on m is e^{th} root of $H(m)$

↳ Adversary has to compute e^{th} of $H(m)$, which is a random value (since H is modeled as a random oracle)

Computing e^{th} root of random target is hard under RSA

Reduction also needs to answer signing queries — relies on "programming" the random oracle

Standard: PKCS1 v1.5 (typically used for signing certificates)

↳ Standard cryptographic hash functions hash into a 256-bit space (e.g., SHA-256), but FDH requires full domain

↳ PKCS 1 v1.5 is a way to pad hashed message before signing:

$$\underbrace{\texttt{00 01}}_{16\text{ bits}}\;\underbrace{\texttt{FF FF} \cdots \texttt{FF FF}}_{\text{pad}}\;\underbrace{\texttt{00 DI}}_{\substack{\text{digest info} \\ \text{(e.g., which hash function} \\ \text{was used)}}}\;\underbrace{H(m)}_{\text{message hash (e.g., computed using SHA-256)}}$$

↳ Padding important to protect against chosen message attacks (e.g., preprocess to find messages m_1, m_2, m_3 where $H(m_1) = H(m_2) \cdot H(m_3)$)
(but this is not a full-domain hash and cannot prove security under RSA — can make stronger assumption...)