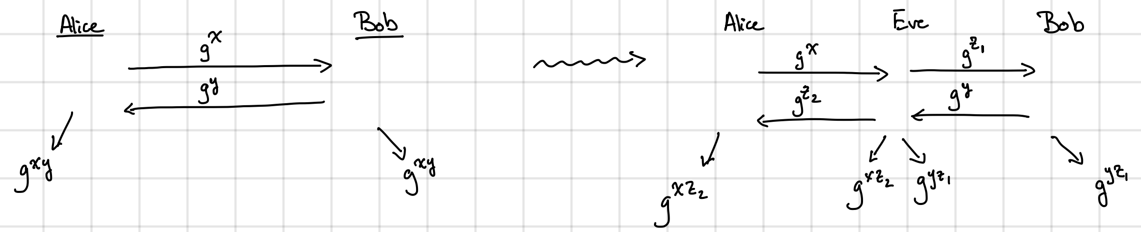
Diffie-Hellman key-exchange is an <u>anonymous</u> key-exchange protocol: neither side knows <u>who</u> they are talking to

↳ 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 <u>no</u> idea!

What we require: <u>authenticated</u> key-exchange (not anonymous) and relies on a root of trust (e.g., a certificate authority)

↳ On the web, one of the parties will <u>authenticate</u> themself by presenting a <u>certificate</u>

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

We will revisit when discussing the TLS protocol

Digital signature scheme: Consists of three algorithms:

- Setup$(1^\lambda) \to (vk, sk)$: Outputs a verification key vk and a signing key sk
- Sign$(sk, m) \to \sigma$: Takes the signing key sk and a message m and outputs a signature σ
- Verify$(vk, m, \sigma) \to 0/1$: Takes the verification key vk, a message m, and a signature σ, and outputs a bit $0/1$

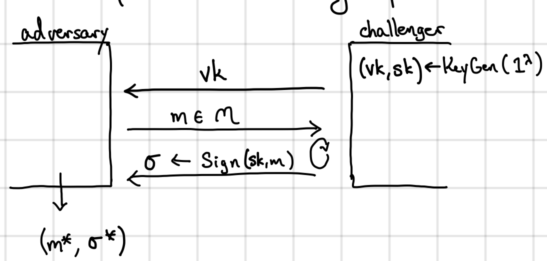
Two requirements:

- <u>Correctness</u>: 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 <u>always</u> verify]

- <u>Unforgeability</u>: 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 <u>new</u> message.

Exact analog of a MAC (slightly weaker unforgeability: require adversary to not be able to forge signature on <u>new</u> message)

↳ MAC security required that no forgery is possible on <u>any</u> 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 (later this semester)

↳ 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 <u>additive</u> group of integers modulo N. Let $\mathbb{Z}_N^*$ be the set of integers that are invertible (under <u>multiplication</u>) 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:

- <u>Factoring</u>: given $N = pq$ where p and q are sampled from a suitable distribution over primes, output p, q
- <u>Computing cube roots</u>: Sample random $x \xleftarrow{R} \mathbb{Z}_N^*$. Given $y = x^3 \pmod N$, compute $x \pmod N$.
  - ↳ This problem is <u>easy</u> 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 <u>as hard as</u> factoring N. In particular, if we know N and $\varphi(N)$, then we can write
    $$\begin{cases} N = pq \\ \varphi(N) = (p-1)(q-1) \end{cases} \quad \text{[both relations hold over the integers]}$$
    and solve this system of equations over the integers (and recover p, q)

Hardness of computing cube roots is the basis of the <u>RSA assumption</u>:

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

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

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

↳ Hardness of RSA relies on $\varphi(N)$ being hard to compute, and thus, on hardness of factoring
(Reverse direction factoring $\overset{?}{\Longrightarrow}$ RSA is <u>not</u> 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 ⟹ ECC generally preferred over RSA
- Both prime factors should have <u>similar</u> bit length (ECM algorithm factors in time that scales with <u>smaller</u> factor)

RSA problem gives an instantiation of more general notion called a <u>trapdoor permutation</u>:

$$F_{RSA} : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$$

$$F_{RSA}(x) := x^e \pmod N \quad \text{where } \gcd(N,e) = 1$$

Given $\varphi(N)$, we can compute $d = e^{-1} \pmod{\varphi(N)}$. Observe that given d, we can invert F_{RSA}:

$$F_{RSA}^{-1}(x) := x^d \pmod N.$$

Then, for all $x \in \mathbb{Z}_N^*$:

$$F_{RSA}^{-1}(F_{RSA}(x)) = (x^e)^d = x^{ed \pmod{\varphi(N)}} = x^1 = x \pmod N.$$

<u>Trapdoor permutations</u>: A trapdoor permutation (TDP) on a domain $\mathcal{X}$ consists of three algorithms:

- $\text{Setup}(1^\lambda) \to (pp, td)$: Outputs public parameters pp and a trapdoor td
- $F(pp, x) \to y$: On input the public parameters pp and input x, outputs $y \in \mathcal{X}$
- $F^{-1}(td, y) \to x$: On input the trapdoor td and input y, output $x \in \mathcal{X}$

Requirements:

- <u>Correctness</u>: for all pp output by Setup:
  - $F(pp, \cdot)$ implements a permutation on $\mathcal{X}$.
  - $F^{-1}(td, F(pp, x)) = x$ for all $x \in \mathcal{X}$.
- <u>Security</u>: $F(pp, \cdot)$ is a one-way function (to an adversary who does not see the trapdoor)

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

Let (F, F^{-1}) be a trapdoor permutation

- Verification key will be pp } to sign a message m, compute $\sigma \leftarrow F^{-1}(td, m)$
- Signing key will be td } to verify a signature, check $m \stackrel{?}{=} F(pp, \sigma)$

Correct because:

$$F(pp, \sigma) = F(pp, F^{-1}(td, m)) = m$$

Secure because F^{-1} is hard to compute without trapdoor (signing key) <u>FALSE!</u>

↳ This is <u>not</u> true! Security of TDP just says that F is one-way. One-wayness just says function is hard to invert on a <u>random</u> input. But in the case of signatures, the <u>message</u> 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 $vk = pp$, compute $F(pp, \sigma)$ for any $\sigma \in \mathcal{X}$

Output $m = F(pp, \sigma)$ and σ

↳ By construction, σ is a valid signature on the message m, and the adversary succeeds with advantage 1.

Textbook RSA signatures: [NEVER USE THIS!]

$\text{Setup}(1^\lambda)$: 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 <u>totally broken</u>!