So far in this course: assumption is that adversary is classical.
How do things change if adversaries are quantum? We won't go into detail but will state main results:

Grover's algorithm: Given black-box access to a function $f : [N] \to \{0,1\}$, Grover's algorithm finds an $x \in [N]$ such that $f(x) = 1$ by making $O(\sqrt{N})$ queries to f.

"Searching an unsorted database of size N in time $O(\sqrt{N})$."

- Classically: Searching an unstructured database of size N requires time $\Omega(N)$ — cannot do better than a linear scan.
- Quantum: Grover's algorithm is tight for unstructured search. Any quantum algorithm for the unstructured search problem requires making $\Omega(\sqrt{N})$ queries (to the function/database).

$\Rightarrow$ Quantum computes provide a quadratic speedup for unstructured search, and more broadly, function inversion.

Implications in cryptography: Consider a one-way function over a 128-bit domain. The task of inverting a one-way function is to find $x \in \{0,1\}^{128}$ such that $f(x) = y$ for some fixed target value f. Exhaustive search would take time $\approx 2^{128}$ on a classical computer, but using Grover's algorithm, can perform in time $\approx \sqrt{2^{128}} = 2^{64}$.

$\Rightarrow$ For symmetric cryptography, need to double key-sizes to maintain same level of security (unless there are new quantum attacks on the underlying construction itself.

$\Rightarrow$ Use AES-256 instead of AES-128 (not a significant change!)

Similar algorithm can be applied to obtain a quantum collision-finding algorithm that runs in time $\sqrt[3]{N}$ where N is the size of the domain (compare to $\sqrt{N}$ for the best classic algorithm)

↳ Instead of using SHA-256, use SHA-384 (not a significant change)

↳ The quantum algorithm require a large amount of space, so not clear that this is a significant threat, but even if it were, using hash functions with 384-bits of output suffices for security

Main takeaway: Symmetric cryptography mostly unaffected by quantum computers ~ generally just require a modest increase in key size

↳ e.g., symmetric encryption, MACs, authenticated encryption

Story more complicated for public-key primitives:

- Simon's algorithm and Shor's algorithm provide polynomial-time algorithms for solving discrete log (in any group with an efficiently-computable group operation) and for factoring
- Both algorithms rely on period finding (and more broadly, on solving the hidden subgroup problem)

Intuition for discrete log algorith (as a period finding problem):

- Let $(g, h = g^\alpha)$ be the discrete log instance in a group of prime order p
- Let $f : \mathbb{Z}_p \times \mathbb{Z}_p \to \mathbb{G}$ be the function
$$f(x,y) = g^x h^{-y}$$
- By construction,
$$f(x+\alpha, y+1) = g^{x+\alpha} h^{-y-1} = g^x h^{-y} g^\alpha h^{-1} = g^x h^{-y} = f(x,y)$$
- Thus, the element $(\alpha, -1)$ is the period of f, so using Shor's algorithm, we can efficient compute $(\alpha, -1)$ from (g,h), which yields the discrete log of h

Thus, if large scale quantum computers come online, we will need new cryptographic assumptions for our public-key primitives
↳ All the algebraic assumptions we have considered so far (e.g., discrete log, factoring) are broken

How realistic is this threat? - Lots of progress in building quantum computers recently by both academia and industry (e.g., see initiatives by Google, IBM, etc.)

- To run Shor's algorithm to factor a 2048-bit RSA modulus, estimated to need a quantum computer with ≈ 10000 logical qubits (analog of a bit in classical computers)
  ↳ With quantum error correction, this requires ≥ 10 million physical qubits to realize
  ↳ Today: machines with 10s of physical qubits, so still very far from being able to run Shor's algorithm
- Optimistic estimate: At least 20-30 years away

Should we be concerned? Quantum computers would break existing key-exchange and signature schemes

- Signatures: Future adversaries would be able to forge signatures under today's public keys, so if quantum computers come online, we can switch to and only use post-quantum schemes
- Key-Exchange: Future adversaries can break confidentiality of today's messages (i.e., we lose forward secrecy) — this is problematic in many scenarios (e.g., businesses want trade secrets to remain hidden for 50 years)

This course: will just focus on getting post-quantum signatures (will not discuss post-quantum key exchange)

[General approach for post-quantum cryptography: base hardness on assumptions believed to be hard on quantum computers (e.g., lattice-based cryptography, isogeny-based cryptography)]

For digital signatures, we will show that OWFs $\Rightarrow$ digital signatures
↳ Signatures can be based on symmetric primitives, so gives one approach to post-quantum signatures

Lamport signatures: Let $f: X \to Y$ be a one-way function.

(n = length of message ($m = \{0,1\}^n$))

- $\text{Setup}(1^\lambda, 1^n)$: Sample $x_{i,b} \xleftarrow{R} X \ \forall i \in [n], b \in \{0,1\}$ and compute $y_{i,b} \leftarrow f(x_{i,b}) \ \forall i \in [2n], b \in \{0,1\}$

  Set

$$sk = \begin{array}{|c|c|c|c|} \hline x_{1,0} & x_{2,0} & \cdots & x_{n,0} \\ \hline x_{1,1} & x_{2,1} & \cdots & x_{n,1} \\ \hline \end{array} \qquad pk = \begin{array}{|c|c|c|c|} \hline y_{1,0} & y_{2,0} & \cdots & y_{n,0} \\ \hline y_{1,1} & y_{2,1} & \cdots & y_{n,1} \\ \hline \end{array}$$

- $\text{Sign}(sk, m \in \{0,1\}^n)$: Output $(x_{1,m_1}, \ldots, x_{n,m_n})$
- $\text{Verify}(vk, m, \sigma)$: Output 1 if $\forall i \in [n], f(x_{i,m_i}) = y_{i,m_i}$ and 0 otherwise.

Theorem. If f is one-way, then Lamport signatures are secure one-time signatures (i.e., where adversary can only make 1 signing query).

Proof. Suppose A is a one-time signature adversary. We construct B for f as follows:

1. Algorithm A receives challenge $y = f(x)$ where $x \xleftarrow{R} X$ from challenger.
2. Choose $i^* \xleftarrow{R} [n]$, $b^* \xleftarrow{R} \{0,1\}$ to program challenge. Sample $x_{i,b} \xleftarrow{R} X$, $y_{i,b} \leftarrow f(x_{i,b})$ for $(i,b) \neq (i^*, b^*)$.
   Set $pk = (y_{1,0}, y_{1,1}, \ldots, y_{n,0}, y_{n,1})$.
3. Send pk to B. If B makes signing query on $m = m_1, \ldots, m_n$:
   - If $m_{i^*} = b^*$, then abort.
   - Otherwise, reply with $(x_{1,m_1}, \ldots, x_{n,m_n})$.
4. After A outputs a forgery (m^*, σ^*), if $m^*_{i^*} \neq b^*$, then abort. Otherwise, output σ_{i^*}.

By construction, $m^*_{i^*} = b^*$ with probability $1/2n$. Thus if A succeed with probability ε, B succeeds with prob. $\varepsilon/2n$.

Limitations: One-time only [will fix later!] (two signatures allow recovering secret key!)

Long public keys, secret keys, and signatures
- Compose with CRHF to get $\text{poly}(\lambda)$-size parameters (independent of message length)
- Secret key can be derived from PRG (e.g., just λ bits)
- Public key can also be shortened to 2λ bits

Many combinatoric tricks to reduce signature size

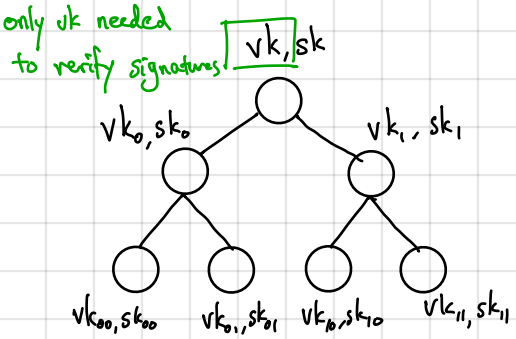
One-time signatures are very fast (only needs symmetric cryptography)
- Very useful in streaming setting: each packet in stream should be signed, but expensive to do so
  - Instead: include pk for one-time signature in first packet
    sign first packet using standard signature algorithm (public key)
    each packet includes OTS public key for next packet:

$$(m_0, vk_1), \sigma \to (m_1, vk_2), \sigma_1 \to (m_2, vk_3), \sigma_2, \ldots$$

σ: signed using many-time signature
σ_1: signed using secret key for vk_1
σ_2: signed using secret key for vk_2
} chaining signatures

Stateful many-time signatures from one-time signatures:

Idea: use a tree of one-time signatures:



- every node is associated with a key-pair for an OTS scheme
- each signing key used to sign verification keys of its children
- signing key for leaf nodes used to sign messages
- each leaf can only be used to sign one message — need to keep track of which nodes have been used (stateful signature)

Example: Signing message m using (vk_{00}, sk_{00}):

- $\sigma_0 \leftarrow \text{Sign}(sk, vk_0 \| vk_1)$
- $\sigma_{00} \leftarrow \text{Sign}(sk_0, vk_{00} \| vk_{01})$
- $\sigma_m \leftarrow \text{Sign}(sk_{00}, m)$
- Output $(vk_0 \| vk_1, vk_{00} \| vk_{01}, \sigma_0, \sigma_{00}, \sigma_m)$

To verify, check
- $\text{Verify}(vk, vk_0 \| vk_1, \sigma_0) = 1$
- $\text{Verify}(vk_0, vk_{00} \| vk_{01}, \sigma_{00}) = 1$
- $\text{Verify}(vk_{00}, m) = 1$

Only root vk needed here, all other keys included in σ

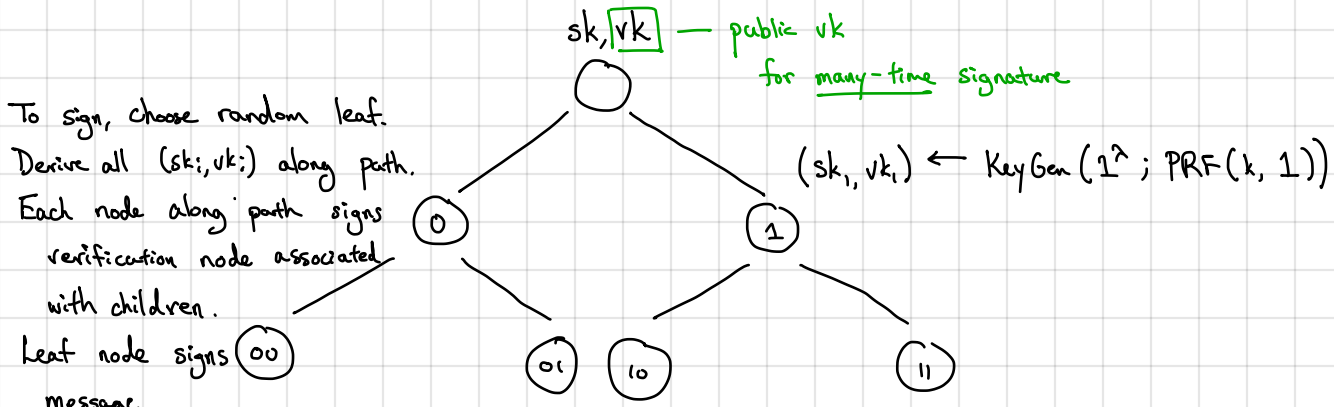
Security (Intuition):
- Keys for internal nodes only used to sign single message (verification keys of children)
- As long as leaf node never reused, then leaves are also only used once
- Security now reduces to one-time security of signature scheme

How to remove state?
- Consider a tree with 2^λ leaves and choose leaf at random for signing
- If we sign $\text{poly}(\lambda)$ messages, there will not be a collision in the leaf with $1 - \text{negl}(\lambda)$ probability
- Problem: Signing key is exponential (need to store $O(2^\lambda)$ signing keys)

Solution: Derive signing keys from a PRF!

$(vk_i, sk_i) \leftarrow \text{KeyGen}(1^\lambda ; \text{PRF}(k, i))$

(randomness to key-generation algorithm; node index; part of signing key)



$(sk_1, vk_1) \leftarrow \text{KeyGen}(1^\lambda ; \text{PRF}(k, 1))$

$(sk_{10}, vk_{10}) \leftarrow \text{KeyGen}(1^\lambda ; \text{PRF}(k, 10))$

To sign, choose random leaf.
Derive all (sk_i, vk_i) along path.
Each node along path signs verification node associated with children.
Leaf node signs message.
Signature contains complete validation path from root to leaf and signature of leaf on message.
Every internal node still signs only one message.