Lamport signatures: Let \( f: X \rightarrow Y \) be a one-way function.

- Setup \( (1^n, 1^n) \): Sample \( X_i, b_i \in X \) \( \forall i \in [n], b_i \in \{0, 1\} \) and compute \( y_i, b_i \leftarrow f(X_i, b_i) \)

\[ \text{Set } S_k = \begin{array}{c} X_0 \quad X_1 \quad \cdots \quad X_{n-1} \\ X_0 \quad X_1 \quad \cdots \quad X_{n-1} \end{array} \quad \text{pk} = \begin{array}{c} y_0 \quad y_1 \quad \cdots \quad y_{n-1} \\ y_0 \quad y_1 \quad \cdots \quad y_{n-1} \end{array} \]

- Sign \((sk, m)\): Output \( X_{im}, \ldots, X_{in}, m\) 
- Verify \((sk, m, \sigma)\): Output \( 1 \) if \( \forall i \in [n] \), \( f(X_i, m_i) = y_i \), and 0 otherwise

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

\textbf{Proof}: Suppose A is a one-time signature adversary. We construct B \textcolor{red}{(will fix later!)} as follows:

1. Algorithm A receives challenge \( y \leftarrow f(x) \) where \( x \in X \) from challenge.
2. Choose \( i^* \in (n), b^* \in \{0, 1\} \) to program challenge. Sample \( X_i, b_i \in X \), \( y_i, b_i \leftarrow f(X_i, b_i) \) for \( (i, b) \neq (i^*, b^*) \).

\[ \text{Set } \text{pk} = (y_0, y_1, \ldots, y_{n-1}, y_{i^*}). \]
3. Send \( \text{pk} \) to B. If B makes signing query on \( m = m_0, \ldots, m_n \):
   - If \( m_i = b^* \), then abort.
   - Otherwise, reply with \( (X_{i^*}, \ldots, X_{in}, m) \).

4. After A outputs a forgery \( (m^*, c^*) \), if \( m^*_i \neq b^* \), then abort. Otherwise, output \( c^* \).

By construction, \( m_i = b^* \) with probability \( 1/n \). Thus \( \Pr[A \text{ succeed}] = \Pr[b^* \land \text{at least one success}] \).

\textbf{Limitations}: One-time only \[ \text{Long public keys, secret keys, and signatures} \]

- Compare with CRHF to get \( \text{poly}(n) \)-size parameters (independent of message length)
- Secret key can be derived from PRG (e.g., just 1 bit)
- Public key can also be shortened to \( 2n \) bits (special case of Winternitz construction below)

\textbf{Many combinatoric tricks to reduce signature size:}

- Winternitz signatures: use an iterated one-way function \( f: X \rightarrow X, f^{(d)} = f(f(\ldots f(x) \ldots)) \)

\[ \text{pk} = H(g_1, \ldots, g_d) \quad \text{key length} \]

\[ \Rightarrow P(m) \text{ maps onto } [d]^\ast \]

\[ \text{L gives out values} \]

\[ \text{corresponding to shaded nodes} \]

\[ \text{we say that } P(m) \leq P(m') \text{ if each component of } P(m) \text{ is smaller than } P(m') \]

\[ \Rightarrow \text{signature on } m \text{ can be used to obtain signature on } m' \]

\[ \Rightarrow \text{for security, just need a function } P \text{ where } P(m) \neq P(m') \]
Constructing $P(m)$:
- View $m$ ∈ $\{0,1\}^t$ as a number in base $d$: $s_1, ..., s_t$ ($d \sim t/\log d$)
- Compute $d (s_1 + \cdots + s_t)$ and write this in base $d$: $t_1, ..., t_{d'}$ ($d' \sim \log d$)
- Output $(s_1, ..., s_t, t_1, ..., t_{d'})$

Suppose $m \leq P(m)$ for some $m = m'$. This means that $s_i \leq s_i'$, ..., $s_t \leq s_t'$ (and at least 1 strict). Then, $(s_1 + \cdots + s_t) < (s_1' + \cdots + s_t')$. Thus, $d (s_1 + \cdots + s_t) < d (s_1' + \cdots + s_t')$ so there is at least one $t_i$ where $t_i > t_i'$, which is a contradiction.

**Benefit of Winteritz construction:** if messages are $O(\lambda)$ bits and $\log |X| = O(\lambda)$ bits, then
- Lamport signatures: $|pk| = O(\lambda^2)$, $|\sigma| = O(\lambda^2)$
- Winteritz: $|pk| = O(\lambda)$, $|\sigma| = O(\lambda/\log d)$

This is very significant in practice! Using CRHF as our $\text{CRHF}$ evaluation is very expensive.

Winteritz ($d=2$): $|pk| = 2$ bytes
- $|\sigma| = 8.5143$ KB
- $|\alpha| = 0.9$ KB

Winteritz ($d=16$): $|pk| = 21.1$ KB
- $|\sigma| = 0.9$ KB

**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:
  - $(m_i, vk_i)$, $\sigma \rightarrow (m_i, vk_{i+1})$, $\sigma_i \rightarrow (m_{i+1}, vk_{i+2})$, $\sigma_{i+1} \rightarrow (m_{i+2}, vk_{i+3}), \sigma_{i+2}, \ldots$
  - Signed using key $vk_i$ for $vk_{i+1}$
  - Signed using key $vk_{i+1}$ for $vk_{i+2}$

**Stateful many-time signatures from one-time signatures:**
- Idea: use a tree of one-time signatures:
  - Only $uk$ needed to verify signatures

```
   vk_0, sk_0
     /   \
    /     /
  vk_1, sk_1
      /   \
     /     /
  vk_2, sk_2
```

- Example: Signing message $m$ using $(vk_0, sk_0)$:
  - $\sigma_0$ ← Sign $(sk, vk_0 || vk_1)$
  - $\sigma_o$ ← Sign $(sk_0, vk_0 || vk_1)$
  - $\sigma_m$ ← Sign $(sk_m, m)$
  - Output $(vk_0 || vk_1, vk_0 || vk_1, \sigma_o, \sigma_m)$

**To verify, check**:
- Verify $(vk, \sigma_o)$
- Verify $(vk_0, \sigma_0)$
- Verify $(vk_0 || m)$

Only root $vk$ needed here, all other keys included in $\sigma$.
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 $2^t$ leaves and choose leaf at random for signing
- If we sign poly$(n)$ messages, there will not be a collision in the leaf with $1 - \negl(n)$ probability
- Problem: Signing key is exponential (need to store $O(2^t)$ signing keys)
  Solution: Derive signing keys from a PRF!

\[
(vk_i, sk_i) \leftarrow \text{KeyGen}(1^t; \text{PRF}(k, i))
\]

For many-time signature

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 $(sk_0, vk_0) \leftarrow \text{KeyGen}(1^t; \text{PRF}(k, 0))$

validation path from root to leaf and signature of leaf on message.
Every internal node still signs only one message.

Signature contains complete $(sk_0, vk_0) \leftarrow \text{KeyGen}(1^t; \text{PRF}(k, 10))$