Setup

- Full binary tree of depth $n$
- Nodes at level $t$ are labeled by $t$-bit strings
- Leaves correspond to messages in $\{0, 1\}^n$
- Each node $x$ holds a key-pair $(pk_x, sk_x)$ for OT signature
- Publish $pk_\varepsilon$ as the public-key
Signing

\[
S_{sk}^0 (001) = pk_0, pk_1, S_{sk\varepsilon} (pk_0, pk_1)
\]

\[
pk_{00}, pk_{01}, S_{sk_0} (pk_{00}, pk_{01})
\]

\[
pk_{001}, pk_{000}, S_{sk_{00}} (pk_{000}, pk_{001})
\]

\[
S_{sk_{001}} (001)
\]
Security

Observations:

- Each key is used once (for its children)
- new document is always signed by at least one new key
Complexity of signing/verifying

- **Signing**: walk along the path from root to leaf $x$, and let each parent (one-time) sign on its children
- **Verification** is done in the natural way
- **Complexity** $O(n)$ applications of the one-time signature
- Longer messages can be hashed down to $n$-bit
- So complexity does not grow with the number of messages :)
Complexity of key-generation

Problem: key of exponential length!

- **Sol 1**: Generate keys on the fly
- **Warning**: This must be done in a **consistent way**! (why?)
- **Sol 2**: Use pseudorandom function $F_k$ and let
  $$(pk, sk)_x := F_k(x)$$
  where $k$ is the global secret key.
Take-Home Message

Some constructions are quite complicated and require several clever ideas.

Although the resulting construction is impractical it contains several concepts which are widely used in practice:

- Hash and Sign
- Key-Refreshing
- Chain of trust
- Use of pseudorandom function to reduce state