Leaf Binding for a 2-Leaf Merkle Tree

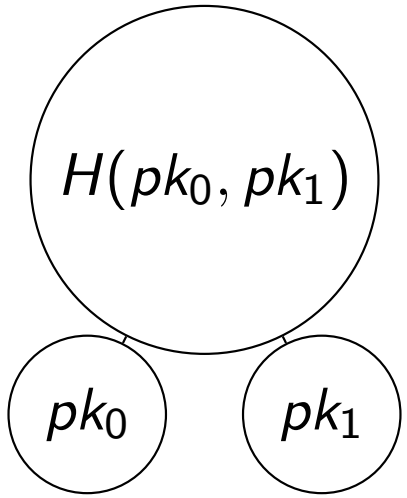
Risiraj Dey

April 10, 2026

Motivation

- Hash-based structures (e.g., Merkle trees) are widely used in modern cryptography
- Applications: digital signatures, blockchains, secure storage
- Security depends not only on hashing, but also on **structural correctness**
- What Binding means: *Does a signature uniquely correspond to a specific leaf?*
- Example: Some Passport $\rightarrow$ Some Country, Whose passport $\rightarrow$ Binding
- Goal: **formally verify leaf-binding property**

Merkle Tree (2 Leaves)



- Root = hash of two leaves
- Signature proves membership of one leaf

Model

- Hash outputs:

$$\texttt{HashVal := BitVec 256}$$

- Hash function:

$$H : \mathit{HashVal} \to \mathit{HashVal} \to \mathit{HashVal}$$

Assumption

$$H(a, b) = H(c, d) \Rightarrow a = c \wedge b = d$$

Signature Structure

- `idx`: selects branch
- `leaf_pk`: leaf key
- `sibling`: other leaf

Verification

if idx then $H(sibling, pk)$

else $H(pk, sibling)$

Verify

```
structure Sig where
  idx       : Bool
  leaf_pk   : HashVal
  leaf_sig  : HashVal
  sibling   : HashVal

def verify (leaf_verify : HashVal      HashVal
    HashVal      Prop)
  (root msg : HashVal) (sig : Sig) : Prop :=
  leaf_verify msg sig.leaf_sig sig.leaf_pk  ∧
  (if sig.idx then H sig.sibling sig.leaf_pk
              else H sig.leaf_pk sig.sibling) = root
```

Main Theorem

Leaf Binding

$$(\text{idx} = \text{false} \wedge pk = pk_0) \ \vee \ (\text{idx} = \text{true} \wedge pk = pk_1)$$

- Signature binds to exactly one leaf
- Prevents ambiguity

Proof Sketch (Lean)

```
cases idx with
| false =>
    H lpk sib = H pk0 pk1
        lpk = pk0
| true =>
    H sib lpk = H pk0 pk1
        lpk = pk1
```

- Case split on index
- Then, apply injectivity of H
  NOTE: Code is abbreviated for presentation purposes

"Compositional" Approach - Combining Both

```
theorem compositional
  (pk0 pk1 msg : HashVal)
  (sig : Sig)
  (hv : verify leaf_ver (tree_root pk0 pk1) msg sig) :
  ((sig.idx = false ∧ sig.leaf_pk = pk0) ∨
   (sig.idx = true  ∧ sig.leaf_pk = pk1)) ∧
  msg = sig.leaf_pk := by
  obtain ⟨h_leaf, h_tree⟩ := hv
  exact
    ⟨leaf_binding leaf_ver pk0 pk1 msg sig ⟨h_leaf, h_tree⟩,
     h_leaf⟩
```

Outcome

- Structural guarantee:

$$\text{Signature binds to one leaf}$$

- Functional guarantee:

$$msg = leaf\_pk$$

Combined

Unique binding + correct message

Instantiation

- Domain: 256-bit values

$$pk_0 = 0\mathrm{x}\ldots42, \quad pk_1 = 0\mathrm{x}\ldots63$$

- Distinct keys verified in Lean
- Signature constructed explicitly

Example (Lean) - Verify

```
-- Concrete keys
def pk0 : HashVal := 0x...42
def pk1 : HashVal := 0x...63

-- Valid signatures
def sig0 : Sig :=   false  , pk0, 0,  pk1, 0
def sig1 : Sig :=   true   , pk1, 0,  pk0, 0

-- Verification proofs
theorem sig0_ok :
  verify leaf_ver (tree_root pk0 pk1) pk0 sig0 :=
    rfl  , by simp [tree_root, sig0]

theorem sig1_ok :
  verify leaf_ver (tree_root pk0 pk1) pk1 sig1 :=
    rfl  , by simp [tree_root, sig1]
```

Binding

```
-- Binding
theorem sig0_binds :
  (sig0.idx = false  ∧  sig0.leaf_pk = pk0)  ∨
  (sig0.idx = true   ∧  sig0.leaf_pk = pk1) :=
  leaf_binding leaf_ver pk0 pk1 pk0 sig0 sig0_ok

theorem sig1_binds :
  (sig1.idx = false  ∧  sig1.leaf_pk = pk0)  ∨
  (sig1.idx = true   ∧  sig1.leaf_pk = pk1) :=
  leaf_binding leaf_ver pk0 pk1 pk1 sig1 sig1_ok
```

Combined Check

```
theorem sig0_full :
  ((sig0.idx = false ∧ sig0.leaf_pk = pk0) ∨
   (sig0.idx = true ∧ sig0.leaf_pk = pk1)) ∧
  pk0 = sig0.leaf_pk :=
  compositional pk0 pk1 pk0 sig0 sig0_ok

theorem sig1_full :
  ((sig1.idx = false ∧ sig1.leaf_pk = pk0) ∨
   (sig1.idx = true ∧ sig1.leaf_pk = pk1)) ∧
  pk1 = sig1.leaf_pk :=
  compositional pk0 pk1 pk1 sig1 sig1_ok
```

Execution Results

- Valid signatures:
  - Verification holds
  - Root matches
- Invalid signature:
  - Rejected via injectivity

Key Insight

Lean kernel verifies correctness

Thank You