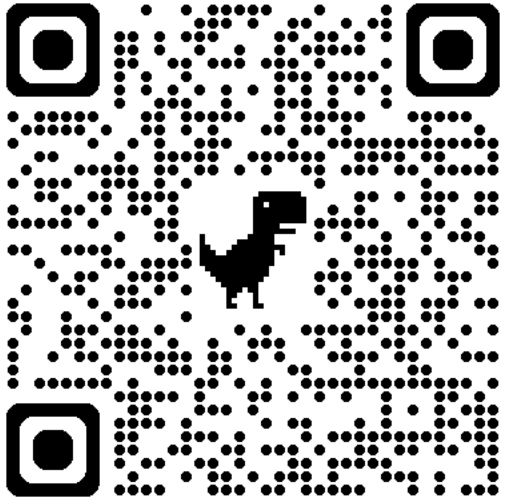
Key Transparency Client Implementation

Felix Linker

What is Implemented?

- Client state and updating view
- Complete Greatest-Version Search
- Limitations
  - No monitoring
  - Hard-coded ciphersuite
  - My code is not interoperable (discussed later)
- Code is proven memory safe (e.g., panic free)
  - Nice to have
  - Requirement for further analysis

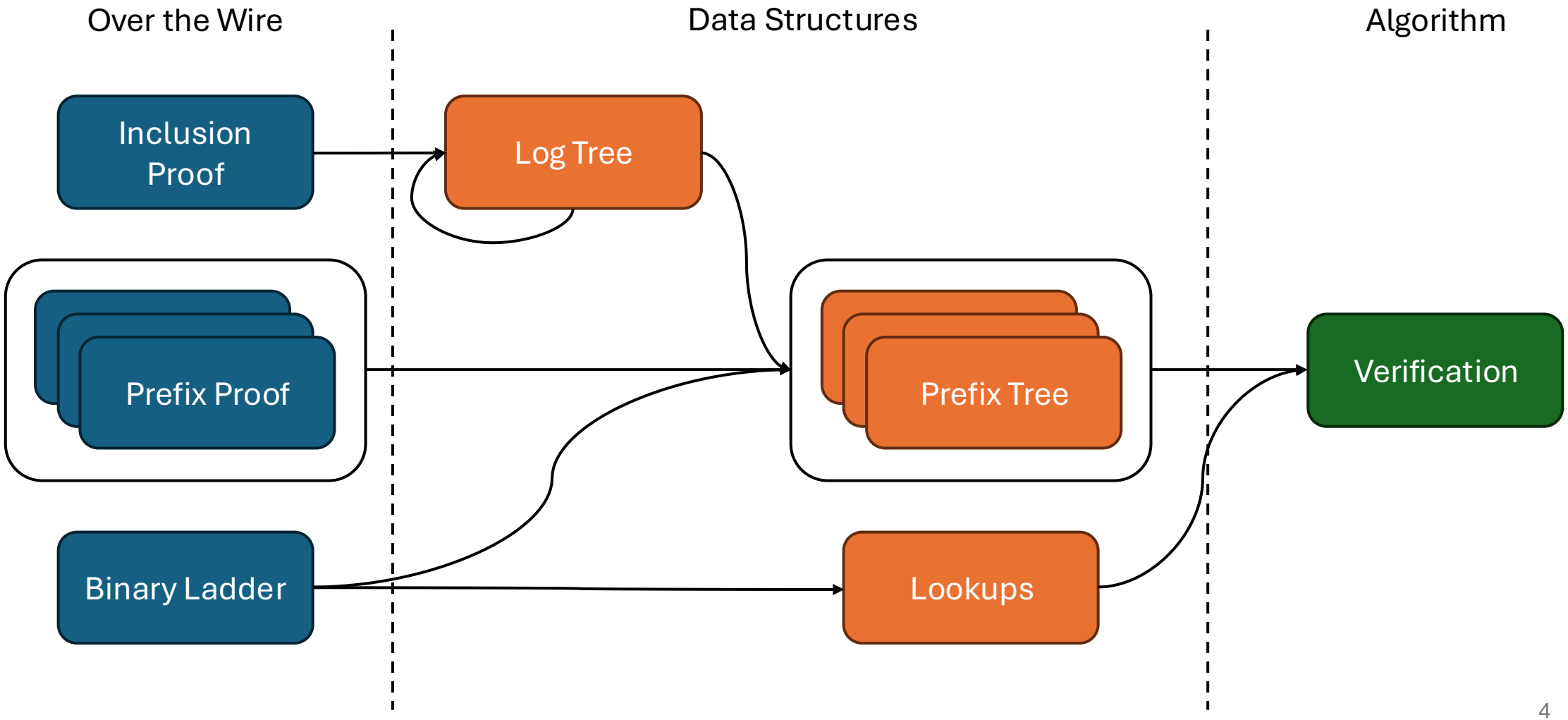


Source code: github.com/felixlinker/keytrans-verification/tree/felixlinker/client-state

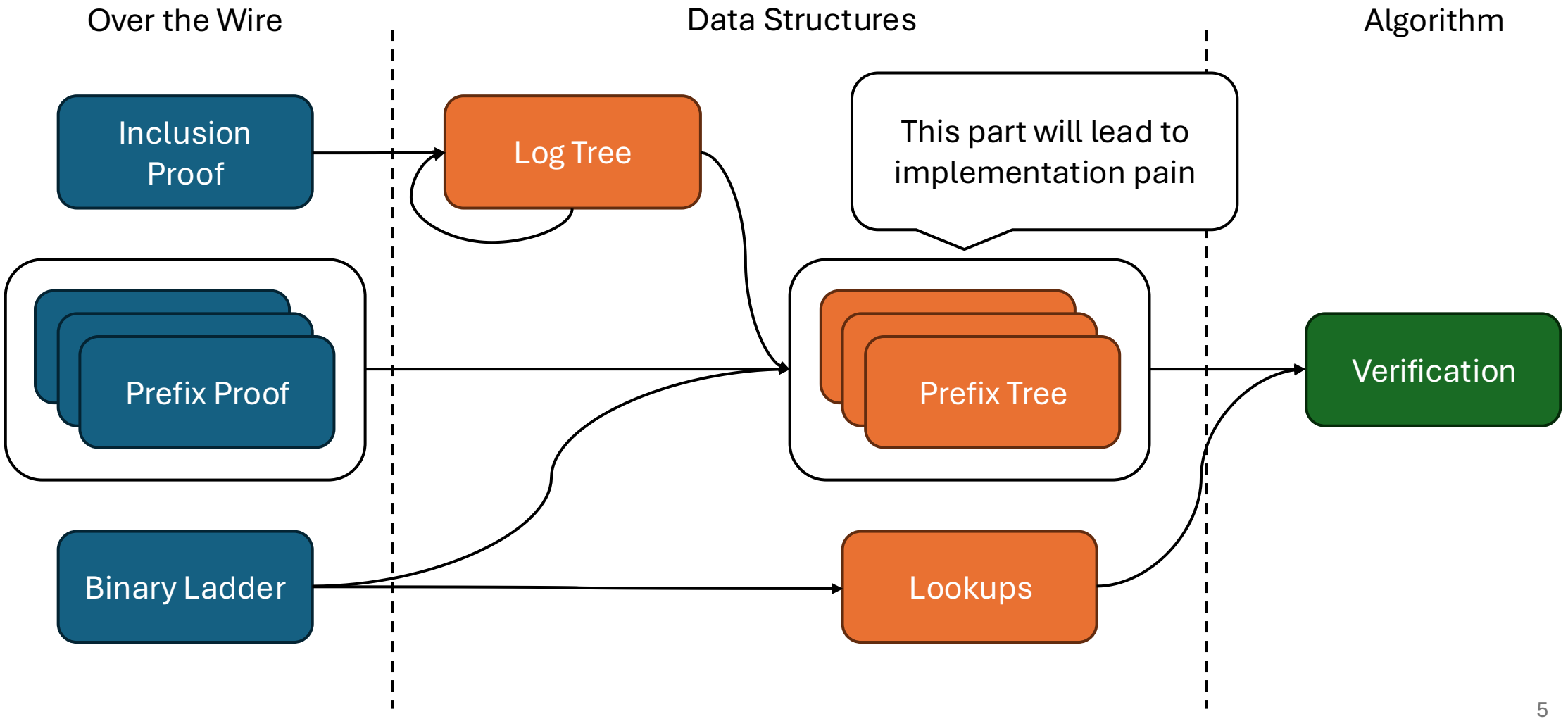
Outline

- Implementing certain parts of the draft was hard
  - Prefix proof structure is hard to parse
  - Data structures described implicitly
  - We’re setting us up for non-interoperability

Information Flow



Information Flow



Prefix Tree Construction

Current

```
enum {
  reserved(0),
  inclusion(1),
  nonInclusionLeaf(2),
  nonInclusionParent(3),
  (255)
} PrefixSearchResultType;

struct {
  opaque vrf_output[VRF.Nh];
  opaque commitment[Hash.Nh];
} PrefixLeaf;

struct {
  PrefixSearchResultType result_type;
  select (PrefixSearchResult.result_type) {
    case nonInclusionLeaf:
      PrefixLeaf leaf;
  };
  uint8 depth;
} PrefixSearchResult;

struct {
  PrefixSearchResult results<0..2^8-1>;
  HashValue elements<0..2^16-1>;
} PrefixProof;
```

My Proposal

```
struct {
  uint8 depth;
  // either nodeValue defined or (commit and vrf_output)
  optional<opaque> node_value[Hash.Nh]; // use length 0 for zero-bytes
  optional<opaque> commitment[Hash.Nh];
  optional<opaque> vrf_output[VRF.Nh];
} PrefixLeaf;

struct {
  PrefixLeaf leafs<0..2^16-1>;
} PrefixProof;
```

Prefix Tree Construction

Current

- Merge in results from binary ladder
- Create empty tree
- Insert all leaves at the respective depth
- DFS from left to right and fill empty children with hashes

My Proposal

- DFS from left to right and fill with hashes

Data Structures Described Implicitly

```
struct {
  uint64 timestamps<0..2^8-1>;
  PrefixProof prefix_proofs<0..2^8-1>;
  HashValue prefix_roots<0..2^8-1>;

  InclusionProof inclusion;
} CombinedTreeProof;
```

The `timestamps` field contains the timestamps of specific log entries, and the `prefix_proofs` field contains search proofs from the prefix trees of specific log entries. There is no explicit indication as to which log entry the elements correspond to, as they are provided in the order that the algorithm the user is executing would request them. The elements of the `prefix_roots` field are, in left-to-right order, the prefix tree root hashes for any log entries whose timestamp was provided in `timestamps` but a search proof was not provided in `prefix_proofs`.

Data Structures Described Implicitly

```
struct {
  uint64 timestamps<0..2^8-1>;
  PrefixProof prefix_proofs<0..2^8-1>;
  HashValue prefix_roots<0..2^8-1>;

  InclusionProof inclusion;
} CombinedTreeProof;
```

Which ones?

Which ones?

Technically underspecified

The `timestamps` field contains the timestamps of specific log entries, and the `prefix_proofs` field contains search proofs from the prefix trees of specific log entries. There is no explicit indication as to which log entry the elements correspond to, as they are provided in the order that the algorithm the user is executing would request them. The elements of the `prefix_roots` field are, in left-to-right order, the prefix tree root hashes for any log entries whose timestamp was provided in `timestamps` but a search proof was not provided in `prefix_proofs`.

Which ones?

Data Structures Described Implicitly

```
struct {
  uint64 timestamps<0..2^8-1>;
  PrefixProof prefix_proofs<0..2^8-1>;
  HashValue prefix_roots<0..2^8-1>;

  InclusionPro
} CombinedTree
```

Which ones?

The protocol draft should be more explicit about the contents of data structures

Which ones?

Technically underspecified

The `timestamps` … `prefix_proofs` f… entries, and the …ees of specific log entries. There is… …he elements correspond to, as… …orithm the user is executing would request them. The elements of the `prefix_roots` field are, in left-to-right order, the prefix tree root hashes for any log entries whose timestamp was provided in `timestamps` but a search proof was not provided in `prefix_proofs`.

Which ones?

Interoperability

- I tested interoperability with Guy's server
- Result: Not interoperable
  - I use TLS encoding
  - Guy uses protobuf

We should suggest default

2.1. Presentation Language

This document uses the TLS presentation langu[age] to describe the structure of protocol messages, but does not require the use of a specific transport protocol. As such, implementations do not necessarily need to transmit messages according to the TLS format and can choose whichever encoding method best suits their application. However, cryptographic computations MUST be done with the TLS presentation language format to ensure the protocol's security properties are maintained.

Summary

- I suggest three changes
  - Simpler prefix tree serialization; decoupled from binary ladder
  - More concrete description of data structures
  - Specify wire format