Proving Properties of Incremental Merkle Trees

Mizuhito Ogawa[1], Eiichi Horita[2], and Satoshi Ono[3]

[1] Japan Advanced Institute of Science and Technology, Ishikawa, 923-1292 Japan
mizuhito@jaist.ac.jp
[2] NTT Information Sharing Platform Laboratories, Tokyo, 180-8585, Japan
horita.eiichi@lab.ntt.co.jp
[3] Kogakuin University, Tokyo, 163-8677 Japan
ono@cpd.kogakuin.ac.jp

Abstract. This paper proves two basic properties of the model of a single attack point-free event ordering system, developed by NTT. This model is based on an incremental construction of Merkle trees, and we show the correctness of (1) completion and (2) an incremental sanity check. These are mainly proved using the theorem prover MONA; especially, this paper gives the first proof of the correctness of the incremental sanity check.

Keywords: Merkle tree, theorem prover, temporal authentication.

1 Introduction

With the growth of the Internet, resilient temporal authentication for system failure and/or malicious attacks becomes important. The standard method is to use a *timestamp based on a public-key cryptosystem*. However, it has relatively short time span (most public keys are renewed each 5 years), and once the cryptography is compromised, all certificates become invalid.

With the aim for long-term validity (say, 20-30 years), NTT developed the event ordering system [5] based on a Merkle tree [8], which is a labeled binary tree such that a label of a node is recursively computed from labels of its child nodes using a hash function. Although an event ordering has relatively rough precision on a time scale, it relies only on the collision-resistance (and one-wayness) of a hash function, which is believed to be much harder than public key cryptosystems. Thus, this system is complementary to a timestamp system based on a public key cryptosystem; with its supplementary use, we can obtain both precision and long-term validity.

The event ordering system receives a hash value of a timestamp, and constructs a Merkle tree in an incremental manner. It issues a certificate immediately after a hash value is registered to a leaf label of a partially constructed Merkle tree. A newly registered hash value is recursively propagated in this bottom-up way, and a certificate is the known part of minimum information to compute the hash value at the root of a Merkle tree. When a whole Merkle tree has been constructed, the hash value at its root is released as a public witness.

Such an incremental construction has been proposed in literature [3, 2, 12, 9, 7], and these studies support

- for long-term validity, the systems relay on only collision-resistance (and one-wayness) of a hash function, and
- each transaction message is kept in $O(log\ n)$ wrt the number of events; thus, they are scalable.

Our system further enhances these systems:

- single attack point free.
- even if the system halts, an intermediate snapshop supports relative correctness of temporal authentication.

This paper proves two basic properties of the incremental construction of a Merkle tree: (1) correctness of completion and (2) correctness of an incremental sanity check. They are mainly proved using theorem prover MONA [1]; especially, this paper first prove (2) correctness of incremental sanity check.

Sections 2 and 3 explain what Merkle trees and MONA are, respectively. Section 4 briefly introduces the event ordering system; terminology for an incremental Merkle forest and our protocol design are explained. Section 6 gives the proof of correctness of completion of an incremental Merkle forest. The proof is performed both by manual induction and by MONA for comparison. Section 7 gives the proof of correctness of the incremental sanity check proposed in [5]. This property has been checked by experiments with large-scale data, but without the proof. Although the proof is not fully formal, the main lemmata are verified by MONA.

2 Merkle tree

$T = (V(T), E(T))$ is a *directed graph* if $E(T) \subseteq V(T) \times V(T)$. We call an element in $V(T)$ a *node*, and an element in $E(T)$ an *edge*. A *path* is a sequence $(t_0, \cdots, t_n)$ of nodes such that for each $1 \leq i \leq n$, $(t_{i-1}, t_i) \in E(T)$. A directed graph T is *acyclic* if there are no paths that visit the same node twice.

We say that an acyclic directed graph $T = (V(T), E(T))$ is a binary tree with the (unique) root denoted by $root(T)$ if

- $root(T) \in V(T)$,
- for each node $t \in V(T)$, there exists the unique path $(t_0, \cdots, t_n)$ such that $t_0 = root(T)$ and $t_n = t$, and
- for each node $t \in V(T)$, if $(\{t\} \times V(T)) \cap E(T) \neq \phi$, there are exactly two edges (t, t') and (t, t'') in $E(T)$ (we call t' and t'' the *child node*s of t).

To distinguish the child nodes of t, we will give the explicit ordering denoted by $t.0$ (left-child) and $t.1$ (right-child). $s \leq s'$ is equivalent to $\bar{v} \in \{0,1\}^*$ being a prefix of $\bar{w} \in \{0,1\}^*$ where $s = t.\bar{v}$ and $s' = t.\bar{w}$. We also say that $t.0$ is the *brother* of $t.1$, and vice versa. Note that the brother relation is symmetric, but

not reflexive, i.e., $t.0$ and $t.1$ are not brothers of themselves, respectively. We say that a node t is a *leaf* if t has no child nodes, and the set of leaves in T is denoted by $leaves(T)$.

The *position* of $t \in V(T)$ is the sequence of 0's and 1's such that the corresponding the sequence of choice of left- and right-child from the root $root(T)$ results the path to t.

In the following, $T = (V(T), E(T))$ is always a binary tree with a root. $T' = (V(T'), E(T'))$ is a subtree of $T = (V(T), E(T))$, if $V(T') \subseteq V(T)$ and there exists $t' \in V(T')$ such that T' is a binary tree with the root t' (i.e., $root(T') = t'$).

As convention, we will denote binary trees by $T, T_1, T_2, ...$, nodes by $s, t, u, v, ...$ and $t_0, t_1, ...$, sets of nodes by $X, Y, Z, ...$, the set of labels by L, and descriptions in MONA by `type writer fonts`.

Definition 1. *Let $g : L \times L \to L$ be a binary function where L is the set of labels. Let $T = (V(T), E(T))$ be a binary tree with the root $root(T)$, where $V(T)$ and $E(T)$ are the sets of nodes and edges, respectively. A* Merkle tree *$MT = (V(T), E(T), \alpha)$ is a L-labeled binary tree with a labeling function $\alpha : leaves(T) \to L$. The label for non-leaf nodes is recursively defined by $\alpha(t) = \alpha(g(t.0, t.1))$.*

We will often overload a tree T and a Merkle tree MT when it is clear from the context.

Remark 1. Originally, a Merkle tree was defined such that each path to a leaf from $root(T)$ has the same length [8]. We generalize a *Merkle tree*, such that paths to leaves may have different lengths. This generalization makes proof of the target properties easier and expressible in $WS2S$.

In our design of the event-ordering system, we assume that *g is a collision-resistant one-way hash function.* Although theoretically it may be difficult to guarantee a collision-resistant and one-way function, in practice we can set an appropriate function.

An event sequence corresponds to the set of leaves of a Merkle tree (in which each path to a leaf has the same length); as default, we consider that time proceeds in a left-to-right manner. Thus, if the level (the length from the root to a leaf) of a Merkle tree T is n, the start leaf is $root(T).\underbrace{0 \cdots 0}_{n}$ and the end leaf is $root(T).\underbrace{1 \cdots 1}_{n}$. At each time unit, the referred leaf shifts to the next (i.e., right neighborhood) leaf. When an event occurs, it put a label (e.g., the hash value of the transaction to be certificated) at the currently referred leaf. The label of each node is computed recursively when the labels of its both children are computed. Thus, if a referred leaf reaches to the end leaf, the label of $root(T)$ is computed.

If g is a collision-resistant one-way hash function, bottom-up computation of hash values (i.e., hash values of both child nodes are concatenated by suitable injective binary operation, and a hash function computes the hash value of their

parent node) is easy; but topdown computation (i.e., from the label of a parent, guess labels of its child nodes) is infeasible. In other words, to interpolate the labels of children is expected to be practically impossible.

In the following definition, the authentication path at a node t is the minimum information to compute the label at $root(T)$ from the label of t, the left authentication path at a node t is the set of labels that were computed before t, and the right authentication path at a node t is the set of labels that will be computed after t. Note that our definition of an authentication path is not restricted to a leaf, but is also for a node.

Definition 2. *Let $t \in V(T)$ and let $(root(T), t_1, \cdots, t_{n-1}, t)$ be a path from $root(t)$ to t.*

- *The* authentication path *of t, denoted by $CA_T(t)$, is the set of brothers of $t_1, \cdots, t_{n-1}, t$.*
- *The* left authentication path *of t, denoted by $LA_T(t)$, is the intersection of $CA_T(t)$ and $\{root(T).0, t_1.0, \cdots, t_{n-1}.0\}$ (i.e., left brothers).*
- *The* right authentication path *of t, denoted by $RA_T(t)$, is the intersection of $CA_T(t)$ and $\{root(T).1, t_1.1, \cdots, t_{n-1}.1\}$ (i.e., right brothers).*
- *The* root path *of t, denoted by $path_T(t)$, is $\{root(T), t_1, \cdots, t_{n-1}, t\}$.*
- *The* path closure *of t, denoted by $pCls(t)$, is $path_T(t) \cup CA_T(t)$.*

We often omit T in $CA_T(t)$, $LA_T(t)$, $RA_T(t)$, $path_T(t)$, and $pCls_T(t)$ as $CA(t)$, $LA(t)$, $RA(t)$, $path(t)$, and $pCls(t)$, if T is clear from the context.

Remark 2. A left (resp. right) authentication path is called a *freshness* (resp. an *existential*) *token*.

3 Monadic Second Order Logic

3.1 $(W)S2S$

Monadic second order logic SnS is a logic on n-ary (possibly infinite) trees. We focus on $S2S$, a logic on binary trees, consisting of

- First order variable, $s, t, u, \cdots$
- Second order variable, $X, Y, Z, \cdots$
- Quantifiers, $\forall$, $\exists$
- Logical connectives, $\wedge$, $\vee$, $\neg$, $\Rightarrow$
- Set operations, $\in$, $\subseteq$, $\cup$, $\cap$, $\setminus$
- Function symbols, $root$, s_0, s_1
- Position relation, $<$, $\leq$

These are interpreted as logical operations on nodes of a binary tree. $root$ is the unique constant that represents the root of a binary tree. The order $s < t$ on nodes means that s is placed between $root$ and t, i.e., s is nearer to the root than t. Note that the satisfiability of an $S2S$-formula is decidable; that is, the

satisfiability of an $S2S$-formula is equivalent to the emptiness problem of a Büchi tree automata [11].

$WS2S$ (weak $S2S$) is the restricted logic of $S2S$ such that the range of set variables (second-order variables) runs on sets of finite trees. The satisfiability of a $WS2S$-formula corresponds to the emptiness problem of a tree automata.

3.2 MONA

```
# s is properly lefter than t
pred lefter(var1 s,t) = ex1 u: (u.0 <= s & u.1 <= t);
# u = glb(s,t)
pred glb(var1 s,t,u) =
  u <= s & u <= t & all1 v: ((v <= s & v <= t) => v <= u);
# s.1...1 = t
pred rightmost(var1 s,t) = s < t & all1 u: ((s <= u & u < t) => u.1 <= t);
# Each pair of nodes in A is incomparable
pred incomparable(var2 A) =
  all1 s,t : ((s in A & t in A) => (s = t | (~(s < t) & ~(t < s))));
# t is the last node in A
pred last(var1 t, var2 A) =
  t in A & all1 s: ((s in A & s ~= t) => lefter(s,t));
# t is the next node of s in A
pred next(var1 s,t, var2 A) =
  s in A & t in A & lefter(s,t) &
  all1 u : ((u in A & lefter(u,t)) => (u = s | lefter(u,s)));
# Y is the lower bound node set of X
pred lower_bound(var2 X,Y) =
  incomparable(Y) & all1 s: (s in X => ex1 t: (t in Y & t <= s));
# X is a (sub)bintree rooted at node s
pred bintree_at(var1 s, var2 X) =
  s in X & all1 t: ((s <= t => ((t notin X => (t.0 notin X & t.1 notin X)) &
                                (t in X => (t.0 in X <=> t.1 in X)))) &
                    (~(s <= t) => t notin X));
# Y is the subtree of X below s
pred below(var1 s, var2 X,Y) = all1 t: ((s <= t & t in X) <=> t in Y);
```

Fig. 1. Library for proofs by MONA

MONA is a batch-style satisfiability checker for $WS2S$ [1]. Although complexity of the satisfiability is non-elementary, MONA is efficiently implemented and practically quite usable. MONA's syntax for $WS2S$ formulae consists of

- First order variable, `s`, `t`, `u`, $\cdots$
- Second order variable, `X`, `Y`, `Z`, $\cdots$
- *Variable declaration*, `var1`, `var2`

– Quantifiers, `all1`, `ex1`, `all2`, `ex2`
– Logical connectives, `&`, `|`, $\sim$, `=>`
– Set operations, `in`, `notin`, `sub`, `union`, `inter`, \
– Function symbols, `root`, `t.0`, `t.1`, `t^`
– Position relation, `<`, `<=`

The difference with $WS2S$ is:

– Quantifiers are explicitly classified for first- or second-order variables.
– Free variables used in a formula need the variable declarations `var1`, `var2` depending on whether they are first- or second-order free variables.
– Since negation (complement of tree automata) is an exponentially heavy operation, `notin` is prepared.
– `t^` is prepared for the ancestor node of `t`.

Note that `<=` is a prefix relation between positions and `=>` is a logical implication. The library used in the paper is shown in Fig. 1.

Example 1. The example below shows predicate definitions and a $WS2S$-formula that means *the closure operation is idempotent.*

```
ws2s;
var2 X,Y,Z;
pred closed(var2 Y) = all1 t: ((t.0 in Y & t.1 in Y) => t in Y);
pred closure(var2 X,Y) =
  closed(Y) & X sub Y & all2 Z : ((closed(Z) & X sub Z) => Y sub Z);
(closure(X,Y) & closure(Y,Z)) => closure(X,Z);
```

The predicate `closed(Y)` means that for each node `t`, if both children `t.0` (left child) and `t.1` (right child) are in `Y`, then `t` is in `Y`. The predicate `closure(X,Y)` means that `Y` is the minimum set such that `Y` is closed and includes `X`. The last line describes the formula to be checked. When these lines are saved as, say, `example.mona`, type "`mona example.mona`"; then it is computed to be `VALID`.

4 Scalable event-ordering system

4.1 Incremental Merkle forest

Definition 3. *Let T be a Merkle tree and let $t \in V(T)$. The* incremental Merkle forest $IMF_T(t)$ *is the union of binary sub-trees T' of T satisfying either*

– $s = root(T')$ *where s is the minimum node such that $s.1 \cdots 1 = t$, or*
– $s.0 = root(T')$ *where $s.1 \leq t$ and $s.\underbrace{1 \cdots 1}_{m} \neq t$ for $\forall m$.*

We will often omit T in $IMF_T(t)$ as $IMF(t)$ when T is clear from the context. Note that $IMF(t)$ is the set of subtrees in which the label (hash value) of each node is defined. In MONA, "*Z is the set of roots of subtrees in $IMF(t)$*" is described as `IMFroot(t,Z)` below.

```
pred defined(var1 s,t) = lefter(s,t) | rightmost(s,t) | s = t;
pred preIMF(var1 t, var2 Z) = all1 s: (s in Z => defined(s,t));
pred IMFroot(var1 t, var2 Z) =
  preIMF(t,Z) & all2 Y: (preIMF(t,Y) => lower_bound(Y,Z));
```

Remark 3. It is tempting to directly define $IMF(t, Z)$ as

```
pred IMF(var1 t, var2 Z) = all1 s: (s in Z <=> defined(s,t));
```

but, this makes `Z` in `IMF(t,Z)` run on infinite sets, i.e., beyond the scope of $WS2S$. For instance, $WS2S$ assumes that `Z` in `preIMF(t,Z)` runs on finite sets.

Note that the restriction to nodes in an incremental Merkle forest does not affect a left authentication path; however, it *does* affect a right authentication path. We say that for $s, t \in V(T)$, s is *lefter* than t (in T) if there exists $u \in V(T)$ such that $u.0 \leq s$ and $u.1 \leq t$ (which corresponds to `lefter(s,t` in Fig. 1).

Definition 4. *Let $s, t \in V(T)$ and let s be lefter than t. A The* relative right authentication path $RA_{T,t}(s)$ *of s wrt t is $RA_T(t) \cap IMF_T(t)$.*

We often omit T in $RA_{T,t}(s)$, if T is clear from the context. In MONA, $CA(t)$, $LA(t)$, $RA_t(s)$, and $pCls_T(t)$ are described as

```
# Authentication path
pred preCA(var1 t, var2 Y) = (all1 s : s.0 <= t => s.1 in Y) &
                             (all1 s : s.1 <= t => s.0 in Y);
pred CA(var1 t, var2 Y) =
  preCA(t,Y) & all2 Z : (preCA(t,Z) => Y sub Z);
# Left authentication path
pred preLA(var1 t, var2 Y) = all1 s : (s.1 <= t => s.0 in Y);
pred LA(var1 t, var2 Y) =
  preLA(t,Y) & all2 Z : (preLA(t,Z) => Y sub Z);
# Right authentication path
pred preRA(var1 s,t, var2 Y) =
  ex2 Z: (IMFroot(t,Z) &
          all1 u: ((u.0 <= s & ex1 v: (v in Z & v <= u.1))
                   => u.1 in Y));
pred RA(var1 s,t, var2 Y) =
  preRA(s,t,Y) & (all2 Z : preRA(s,t,Z) => Y sub Z);
# Path closure
pred pCls(var1 t, var2 X) =
  all1 s: (s in X <=> (s <= t | (ex2 Y: (CA(t,Y) & s in Y))));
```

Note that $PCls(t) = Cls(CA(t) \cup \{t\})$ where $Cls(X)$ is a *closure* operator is defined below.

Definition 5. *Let T be a Merkle tree. For $X \subseteq V(T)$, the* closure $Cls(X)$ *is the minimum set satisfying*

– $X \subseteq Cls(X)$, *and*
– *if both child nodes of* t *is in* $Cls(X)$, *then* t *is in* $Cls(X)$.

In MONA, "$Y = Cls(X)$" is described as `closure(X,Y)` (see Example 1).

For notational convenience, we define $LS(t) = LA(t) \cup \{t\}$ and $LSR_t(s) = LS(s) \cup RA_t(s)$, where $s, t \in V(T)$ such that s is lefter than t. In MONA, they are described as

```
pred LS(var1 t, var2 X) = all2 Y: (LA(t,Y) => X = Y union {t});
pred LSR(var1 s,t, var2 Z) =
  ex2 X,Y: (LS(s,X) & RA(s,t,Y) & Z = X union Y);
```

4.2 Incremental scheme for optimal slice replication

Let $A = \{t_1, \cdots, t_k\}$ be a set of leaf nodes of T where one user requests to register events. An incremental Merkle forest $IMF(t_k)$ is also called a *temporal slice at* t_k. A *spatial slice of* A is the union of path closures of nodes in A (i.e., $\cup_{t_i \in A}\ pCls(t_i)$), and an *optimal slice* of A is the intersection of the temporal slice at t_k and the spatial slice of A (i.e., $(\cup_{t_i \in A}\ pCls(t_i)) \cap IMF(t_k)$).

A *path slice of* t_i *at* t_j (for $i < j$) is the intersection of the root path of t_i and the temporal slice at t_j, i.e., the fragment of the root path of t_i in which each hash value is known at t_j.

Fig. 2 shows the spatial/temporal/optimal slices of of $A = \{t_1, t_2, t_3, t_4, t_5\}$. The area surrounded by the dotted line is the spatial slice of A, the area surrounded by the thin line is the temporal slice at t_5, and their intersection is the optimal slice of A. The set of circled nodes is the left authentication path $LA(t_4)$ at t_4, the set of two boxed node is the right authentication path $RA(t_4)$, and $RA_{t_5}(t_4)$ consists of the node boxed with the line. The thick line that stems from t_4 shows the path slice of t_4 at t_5.

The protocol of our event-ordering system proceeds with the following transaction at each request from a user. The detailed algorithm is described in [5].

– For the request at t_1, return a pair $(\phi, LS(t_1))$.
– For a request at t_i with $1 < i \leq k$, return a pair $(RA_{t_i}(t_{i-1}), LS(t_i))$.

Theorem 1 guarantees that one can recover the optimal slice of A only from $LSR_{t_{i+1}}(t_i)$'s (for $1 \leq i < k$) and $LS(t_k)$, which are obtained by transactions of the protocol. This is called *completion*. Each message at a transaction is logarithmically small, and this gives an efficient optimal slice replication.

The event ordering system is designed to use this protocol between an auditor and a server, as well as an user and a server. One auditor is assumed to periodically register events $a_1, a_2, \cdots$. If these are sufficiently frequent, there will be an auditor's request a_j between one user's requests t_i and t_{i+1}. In such situation, the left authentication path $LS(a_j)$ has an overlap with the path slice of t_i at t_{i+1}. Then, the auditor can confirm that t_i occurs before a_j by comparing a hash value of an overlapping node at the user side and that at the auditor side.

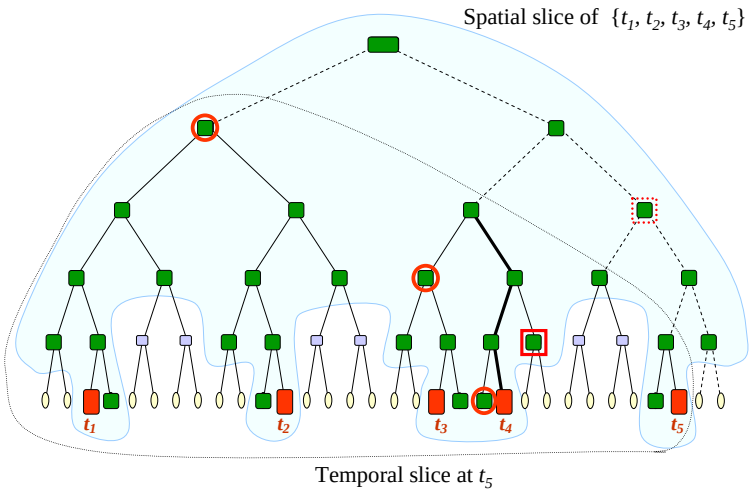

Fig. 2. Incremental Merkle forest

The systems studied in [3, 2, 12, 9, 7] could have this ability, assuming that information on right authentication paths is obtained from a server. Note that our system design can perform the same thing *without* information from a server, because each participant keeps its own optimal slice replication. Thus, our system is safe from a server clash.

The optimal slice replication also enables participants to check each other without making inquiries to a server. We also assume that there are multiple auditors; this enables them to detect a malicious auditor even if a server halts. This setting guarantees single attack point free.

Theorem 2 guarantees the correctness of an efficient *incremental sanity check*, i.e., consistency among labels of nodes in $\{LSR_{t_{i+1}}(t_i) \mid 1 \leq i < k\} \cup \{LS(t_k)\}$ can be incrementally verified by weak consistency between each pair of neighbors $\{LSR_{t_{i+1}}(t_i), LS(t_{i+1})\}$ for $1 \leq i < k$.

During an incremental optimal slice replication, hash values may be computed at different moments even for the same node. The consistency among multiple definitions enables us (including a server itself) early detection of server errors and/or malicious attacks.

The proof of Theorem 1 (in Section 6) is fully performed by MONA, because it can be described in terms of nodes in a binary tree T. However, the proof of Theorem 2 (in Section 7) is only partially performed by MONA; the use of MONA is restricted to proofs of the main lemmata, which are essential for inductive steps in the full proof. MONA is fully automatic, thus its scope and ability are restricted. The main limitations here are:

- MONA lacks induction, and
- MONA lacks a description for equality between labels.

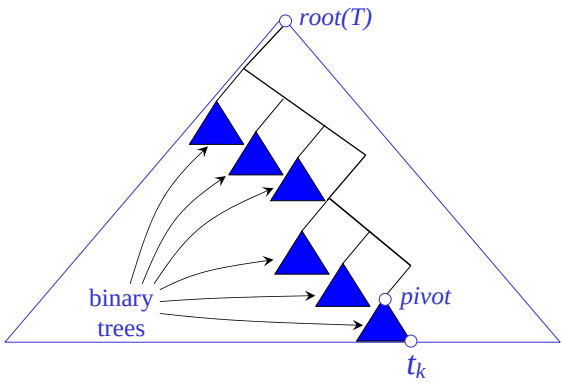

Fig. 3. $IMF(t_k)$ as a pivoted forest

5 Characterization as a pivoted forest

Although the characterization given in this section is more than that needed in later sections (what we need in the proof of Lemma 8 is the fact that the union and the intersection of a forest of binary trees are again a forest of binary trees), this will clarify the perspective.

Definition 6. *Let T be a Merkle tree. A node $t \in V(T)$ is a* pivot *if either*

- $t = root(T)$, *or*
- t *is the left child of a node.*

A forest $X \subseteq V(T)$ of binary trees is a pivoted forest (wrt a pivot t) *if $X = \cup_{s \in LS(t)} X_s$ where X_s is a binary tree with $root(X_s) = s$.*

In MONA, "X is a pivoted forest wrt t" is given as `pivoted_forest(t, X)`.

```
pred pivoted_forest(var1 t, var2 X) =
  all2 Y: (LS(t,Y) =>
          (lower_bound(X,Y) & Y sub X &
           all1 s: (s in Y =>
                   (all2 Z: (below(s,X,Z) => bintree_at(s,Z))))));
```

Let $A = \{t_1, \cdots, t_k\}$ be a set of leaf nodes of T such that t_i is lefter than t_{i+1}. We first show that an incremental Merkle forest $IMF(t_k)$ is a pivoted forest, as described in Fig. 3.

Lemma 1. *An incremental Merkle forest $IMF(t)$ is a pivoted tree where its pivot is the root of the rightmost component of $IMF(t)$.*

In MONA, Lemma 1 is described below and verified as `VALID`.

```
(IMFroot(t,X) & last(s,X)) => pivoted_forest(s,X);
```

Second, $Cls_T(LSR_{t_{i+1}}(t_i))$ for $1 \leq i < k$, $Cls_T(LS(t_k))$, and their union are also pivoted forests.

Lemma 2. *1. $Cls_T(LS(s))$ is a pivoted forest wrt u where u is the minimum node with $s = u.1 \cdots 1$.*
2. Let $s, t \in V(T)$ such that s is lefter than t and let $v = glb(s,t)$. Then, $Cls_T(LSR(s,t))$ is a pivoted forest wrt u where u is
– the minimum node with $t = u.1 \cdots 1$ if $t = v.1 \cdots 1$, and
– $v.0$ otherwise.

To describe Lemma 2, we prepare predicates that describe

– "$X = Cls_T(LS(s))$" as `LSclosure(s,X)`,
– "$X = Cls_T(LSR_t(s))$" as `LSRclosure(s,t,X)`,
– "*u is a pivot of $Cls_T(LS(s))$*" as `LSpivot(s,u)`, and
– "*u is a pivot of $Cls_T(LSR_t(s,u))$*" as `LSRpivot(s,t,u)`.

```
pred LSclosure(var1 t, var2 X) = ex2 Y: (LS(t,Y) & closure(Y,X));
pred LSRclosure(var1 s,t, var2 X) =
                              ex2 Y: (LSR(s,t,Y) & closure(Y,X));
pred LSpivot(var1 s,t) =
          (t = s | rightmost(t,s)) &
          (all1 u: ((u = s | rightmost(u,s)) => t <= u));
pred LSRpivot(var1 s,t,u) =
  all1 v: (((glb(s,t,v) & rightmost(v,t)) => LSpivot(t,u)) &
           ((glb(s,t,v) & ~rightmost(v,t)) => u = v.0));
```

In MONA, the statement of Lemma 2 is described as

```
(LS(s,X) & closure(X,Y) & LSpivot(s,t)) => pivoted_forest(t,Y);
(lefter(s,t) & LSR(s,t,X) & LSRpivot(s,t,u) & closure(X,Y))
                                        => pivoted_forest(u,Y);
```

and is verified as `VALID`.

Lemma 3. *Let X, Y be pivoted forests wrt to pivots s, t, respectively. If $s \in Y$, then $X \cup Y$ (resp. $X \cap Y$) is a pivoted forest wrt t (resp. s).*

In MONA, this statement is described as

```
(pivoted_forest(s,X) & pivoted_forest(t,Y) & s in Y) =>
  ((pivoted_forest(t, X union Y) & pivoted_forest(s, X inter Y)));
```

and is verified as `VALID`.

Lemma 4. *If s is lefter than t, the pivot of $Cls(LSR_t(s))$ is in $Cls(LS(t))$.*

This is described as

```
(lefter(s,t) & LSRpivot(s,t,u) & LSclosure(t,X)) => t in X;
```

is verified as `VALID` by MONA. Thus, next Corollary is immediate.

Corollary 1. *$(\cup_{1 \leq i < k} Cls_T(LSR_{t_{i+1}}(t_i))) \cup Cls_T(LS(t_k))$ is a pivoted forest wrt u where u is the minimum node satisfying $u.1 \cdots 1 = t_k$.*

6 Completion

6.1 Completion in incremental Merkle forest

Intuitively, completion is a process to collect all nodes in an incremental Merkle forest such that their hash values can be computed only from issued certificates. Its correctness is, whether an optimal slice at the moment can be computed (Theorem 1) at a user side well a a server side.

Theorem 1. *(Theorem 1 in [5]) Let $A = \{t_1, t_2, \cdots, t_k\}$ be leaves in a Merkle tree T such that t_i is lefter than t_{i+1} for $1 \leq i < k$. Then,*

$$(\cup_{1 \leq i \leq k}\ pCls(t_i))\ \cap\ IMF(t_k) = Cls(\cup_{1 \leq i < k}\ LSR_{t_{i+1}}(t_i)\ \cup\ LS(t_k)).$$

In the system, completion can be done efficiently by a right-to-left incremental closure operations. For notational convenience, we define a *path closure slice* $pClsSlc_t(s) = pCls(s)\ \cap\ IMF(t)$ where s is lefter than t. In MONA, $pClsSlc_t(s)$ is described as `pClsSlc(s,t,X)`.

```
pred pClsSlc(var1 s,t, var2 X) =
  ex2 Y,Z: (pCls(s,Y) & IMFroot(t,Z) &
             all1 u: (u in X <=> (u in Y & ex1 v: (v in Z & v <= u))));
```

By the distributive law

$$(\cup_{1 \leq i \leq k}\ pCls(t_i))\ \cap\ IMF(t_k) = \cup_{1 \leq i \leq k}\ (pCls(t_i)\ \cap\ IMF(t_k)),$$

the completion is enough to compute $pClsSlc_{t_k}(t_i)$ for $1 \leq i \leq k$. Then, the completion algorithm (guaranteed by Lemma 5) is:

1. Compute $pClsSlc_{t_k}(t_k)$, which is $Cls(LS(t_k))$.
2. When $pClsSlc_{t_k}(t_{i+1})$ (for $1 \leq i < k$) is computed, compute $pClsSlc_{t_k}(t_i)$, which is contained in $Cls(pClsSlc_{t_k}(t_{i+1})\ \cup\ LSR_{t_{i+1}}(t_i))$.

Note that during computation, each step requires only logarithmic time.

Lemma 5. *Let $A = \{t_1, t_2, \cdots, t_k\}$ be leaves in a Merkle tree T such that t_i is lefter than t_{i+1} for $1 \leq i < k$. Then,*

- $pClsSlc_{t_k}(t_k)\ =\ Cls(LS(t_k))$.
- $pClsSlc_{t_k}(t_i)\ \subseteq\ Cls(pClsSlc_{t_k}(t_{i+1})\ \cup\ LSR_{t_{i+1}}(t_i))$.

Section 6.2 will show a manual proof of Theorem 1, and Section 6.3 will show a formal proof by MONA for comparison of proofs by human and machine.

6.2 Proving Theorem 1 by induction

Let $A = \{t_1, t_2, \cdots, t_k\}$ such that for t_i is lefter than t_{i+1} for $1 \leq i < k$.

Proof of Theorem 1 by induction on k. By induction on k. If $k = 1$, obvious. For $k > 1$, by induction hypothesis,

$$(\cup_{2 \leq i \leq k}\ pCls(t_i))\ \cap\ IMF(t_k) = Cls(\cup_{2 \leq i < k}\ LSR_{t_{i+1}}(t_i)\ \cup\ LS(t_k)).$$

Let $u = glb(t_1, t_2)$. We denote the left child node of u by $u.0$ and the right child node by $u.1$, respectively. Since t_1 is lefter than t_2, $u.0 \leq t_1$ and $u.1 \leq t_2$. Since $u.0 \in LS(t_2)$, $u.0 \in IMF(t_k)$. Thus, $u.0 \in pClsSlc_{t_k}(t_1)$.

Let $X_1 = \{t \in pClsSlc_{t_k}(t_1) \mid u.0 \leq t\}$ and $X_2 = pClsSlc_{t_k}(t_1) \setminus X_1$. Then, $X_1 \subseteq Cls(LSR_{t_2}(t_1))$ and $X_2 \subseteq pClsSlc_{t_k}(u.0) \subseteq pClsSlc_{t_k}(t_2)$. Therefore

$$(\cup_{1 \leq i \leq k}\ pCls(t_i))\ \cap\ IMF(t_k) \subseteq Cls(\cup_{1 \leq i < k}\ LSR_{t_{i+1}}(t_i)\ \cup\ LS(t_k)).$$

The opposite direction is obvious. ■

The proof of Lemma 5 can be performed similarly to that of Theorem 1.

6.3 Proving Theorem 1 by MONA

Let $A = \{t_1, t_2, \cdots, t_k\}$ such that t_i is lefter than t_{i+1} for $1 \leq i < k$. Define:

- "$X = \cup_{1 \leq i < k}\ LSR_{t_{i+1}}(t_i)\ \cup\ LS(t_k)$" is denoted by `LSRunion(A,X)`.
- "$X = \cup_{1 \leq j \leq k}\ pCls(t_j)$" is denoted by `spatial_slice(A,X)`.
- "$X = (\cup_{1 \leq j \leq k}\ pCls(t_j))\ \cap\ IMF(t_k)$" is denoted by `opt_slice(A,X)`.

```
pred LSRunion(var2 A,X) =
  all1 s: (s in X <=>
      ex1 t: ((ex1 u: ex2 Y: (next(t,u,A) & LSR(t,u,Y) & s in Y)) |
              (ex2 Z: (last(t,A) & LS(t,Z) & s in Z))));
pred spatial_slice(var2 A,X) =
  all1 s: (s in X <=>
           ex1 t: ex2 Y: (t in A & pCls(t,Y) & s in Y));
pred opt_slice(var2 A,X) =
  ex1 t: ex2 Y,Z: (last(t,A) & IMFroot(t,Y) & spatial_slice(A,Z) &
                   all1 u: (u in X <=>
                            (u in Z & ex1 s: (s in Y & s <= u))));
```

The statement of Theorem 1 is described as

```
(incomparable(A) & opt_slice(A,X) & LSRunion(A,Y) & closure(Y,Z))
                                                    => X = Z;
```

and is verified as `VALID` by MONA. The statement of Lemma 5 is described as

```
(LS(t,X) & pClsSlc(t,t,Y) & closure(X,Z)) => Y = Z;
(lefter(s,t) & (lefter(t,u) | t = u) & LSR(s,t,X) &
 pClsSlc(s,u,Y) & pClsSlc(t,u,Z) & closure(X union Z,C)) => Y sub C;
```

and also verified as `VALID`.

7 Incremental sanity check

7.1 Consistency

Let $A = \{t_1, t_2, \cdots, t_k\}$ such that for each pair (t_i, t_{i+1}) with $1 \leq i < k$, t_i is lefter than t_{i+1}. Upon completion, there may be nodes in a Merkle tree such that their labels are computed from different $LSR_{t_{i+1}}(t_i)$'s. If multiple computations of the label of a node coincide, this will be an indication of no system failures and/or no malicious attacks. This check of a server can be also performed by users and auditors, as well as self check by a server itself. This is called a *sanity check*; however, the naive way will be too expensive. We will show that an *incremental sanity check* that verifies weak consistency between each pair of neighbors $(LSR_{t_{i+1}}(t_i), LS(t_{i+1}))$ is enough.

To formalize the sanity check, we need to distinguish generated labels at each transaction; we associate a labeling (partial) function $\alpha_i : leaves(T) \to L$ to each pair $(LSR_{t_{i+1}}(t_i), LS(t_i))$. Note that during the sanity check, $g : L \times L \to L$ is fixed.

Definition 7. *Let $U_i \subseteq V(T)$ be a set of incomparable nodes in a Merkle tree T and let $\alpha_i : U_i \to L$ be a labeling (partial) function such that α_i is extended by $\alpha_i(t) = \alpha_i(g(t.0, t.1))$ for $t \in cCLS_T(U_i)$.*

- $\{(U_i, \alpha_i)\}$ *is* weakly consistent *if $t \in cCLS_T(U_i) \cap cCLS_T(U_j)$ implies $\alpha_i(t) = \alpha_j(t)$.*
- $\{(U_i, \alpha_i)\}$ *is* consistent *if for each $t \in Cls_T(\cup U_i)$, $\alpha(t)$ is well-defined where*

$$\alpha(t) = \begin{cases} \alpha_i(t) & \text{when } t \in leaves(U_i) \\ \alpha(g(t.0, t.1)) & \text{when } t \notin leaves(\cup U_i) \end{cases}$$

Note that $\alpha(t)$ may have multiple definitions, i.e., t may be a leaf node of some U_i and simultaneously t may be a non-leaf node of some U_j.

Theorem 2. *If $(LSR_{t_{i+1}}(t_i), \alpha_i)$ and $(LS(t_{i+1}), \alpha_{i+1})$ are weakly consistent for $1 \leq i < k$, $\{(LSR_{t_{i+1}}(t_i), \alpha_i) \mid 1 \leq i < k\} \cup \{(LS(t_k), \alpha_k)\}$ is consistent.*

Note that weak consistency between $(LSR_{t_{i+1}}(t_i), \alpha_i)$ and $(LS(t_{i+1}), \alpha_{i+1})$ can be checked quite efficiently. That is, by Lemma 2, 3, and 4, the set of minimum nodes in $Cls_T(LSR_{t_{i+1}}(t_i)) \cap Cls_T(LS(t_{i+1}))$ is $LS(u)$ where u is the pivot of $Cls_T(LS(t_{i+1}))$. In practice, we assume a collision-resistant one-way hash function g; thus, it is enough to check whether each hash value by α_i at a node in $LS(u)$ coincides with that by α_{i+1}.

7.2 Proving weak consistency

For the former half of the proof of Theorem 2, we will prove that *if $(LSR_{t_{i+1}}(t_i), \alpha_i)$ and $(LS(t_{i+1}), \alpha_{i+1})$ are weakly consistent for each i with $1 \leq i < k$, then $\{(LSR_{t_{i+1}}(t_i), \alpha_i) \mid 1 \leq i < k\} \cup \{(LSR_{t_k}(t_k), \alpha_k)\}$ is weakly consistent.*

Lemma 6. *Let $s, t, u, v, w \in V(T)$ such that s is lefter than t, t is lefter than or equal to u, u is lefter than or equal to v, and v is lefter than w. Then,*

1. $LSR_t(s) \cap LSR_w(v) \subseteq LS(u)$, *and*
2. $Cls_T(LSR_t(s)) \cap Cls_T(LSR_w(v)) \subseteq Cls_T(LS(u))$.

In MONA, the statement of Lemma 6 is described as

```
(lefter(s,t) & (t = u | lefter(t,u)) & (u = v | lefter(u,v)) &
 lefter(v,w) & LSR(s,t,X) & LS(u,Y) & LSR(v,w,Z))
                                              => X inter Z sub Y;
(lefter(s,t) & (t = u | lefter(t,u)) & (u = v | lefter(u,v)) &
 lefter(v,w) & LSRclosure(s,t,X) & LSclosure(u,Y)
                      & LSRclosure(v,w,Z)) => X inter Z sub Y;
```

and is verified as `VALID`.

Lemma 7. *If $(LSR_{t_{i+1}}(t_i), \alpha_i)$ and $(LS(t_{i+1}), \alpha_{i+1})$ are weakly consistent for each i with $1 \leq i < k$, then $\{(LSR_{t_{i+1}}(t_i), \alpha_i) \mid 1 \leq i < k\} \cup \{(LS(t_k), \alpha_k)\}$ is weakly consistent.*

Proof. By induction on k. If $k = 1$, obvious. Assume $k > 1$ and the statement holds for $k - 1$. Let $X = (\cup_{1 \leq i < k-1} Cls_T(LSR_{t_{i+1}}(t_i)) \cup Cls_T(LS(t_{k-1}))$.

It is enough to consider the intersection $X_1 = X \cap Cls_T(LSR_{t_k}(t_{k-1}))$ and $X_2 = X \cap Cls_T(LS(t_k))$.

From Lemma 6, $X_1, X_2 \subseteq Cls_T(LS(t_{k-1}))$. Since $(LSR_{t_k}(t_{k-1}), \alpha_{k-1})$ and $(LS(t_k), \alpha_k)$ are weakly consistent, Lemma is proved. ■

Note that MONA cannot verify Lemma 7, because it cannot describe the equality between labels.

7.3 Proving consistency

For the latter half of the proof of Theorem 2, we will prove that *if*

$$\{(LSR_{t_{i+1}}(t_i), \alpha_i) \mid 1 \leq i < k\} \cup \{(LS(t_k), \alpha_k)\}$$

is weakly consistent, they are consistent. This complete the proof of Theorem 2.

Lemma 8. *If $\{(LSR_{t_{i+1}}(t_i), \alpha_i) \mid 1 \leq i < k\} \cup \{(LS(t_k), \alpha_k)\}$ is weakly consistent, they are consistent.*

For notational convenience, we define

$$LSR_A(t_i) = \begin{cases} LSR_{t_{i+1}}(t_i) & \text{for } 1 \leq i < k \\ LS(t_k) & \text{for } i = k \end{cases}$$

for $A = \{t_1, \cdots, t_k\}$.

Proof. Let $X = (\cup_{1 \leq i < k} Cls_T(LSR_{t_{i+1}}(t_i)) \cup Cls_T(LS(t_k))$. From Corollary 1, X is a pivoted forest; thus, for each $t \in X$, $t.0 \in X$ if and only if $t.1 \in X$.

For each $t \in X$, we will prove that the labeling function $\alpha : V(X) \to L$ is well-defined by induction on the size of $X \cap T_t$ where $T_t = \{s \in V(T) \mid t \leq s\}$. If $|X \cap T_t| = 1$, this means $t \in LSR_A(t_i)$ or $t \notin Cls_T(LSR_A(t_i))$ for each i. Since $LSR_A(t_i)$'s are weakly consistent, $\alpha(t)$ is well-defined.

Assume $|X \cap T_t| > 1$. Since $X \cap T_t$ is a forest of binary trees, $t.0, t.1 \in X \cap T_t$. If $t \in Cls_T(LSR_A(t_i))$, either $t.0,\ t.1 \in Cls_T(LSR_A(t_i))$ or $t \in LSR_A(t_i)$.

Since $|X \cap T_{t.0}|,\ |X \cap T_{t.1}| < |X \cap T_t|$, induction hypothesis implies that $\alpha(t.0)$ and $\alpha(t.1)$ are well-defined. Let $I_0 = \{i \mid t.0, t.1 \in Cls_T(LSR_A(t_i))\}$ and $I_1 = \{i \mid t \in LSR_A(t_i)\}$. Since $|X \cap T_t| > 1$, $I_0 \neq \phi$.

Let $j \in I_0$; then $\alpha_j(t) = \alpha(g(t.0, t.1))$. Weakly consistency of $LSR_A(t_i)$'s implies that $\alpha_j(t) = \alpha_i(t)$ for each $i \in I_1$. Thus, $\alpha(t) = \alpha_j(t)$ is well-defined. ■

Theorem 2 is immediate from Lemma 7 and 8. Note that MONA cannot verify Lemma 8, because it cannot describe the equality between labels.

8 Conclusion

This paper proved two basic properties

1. correctness of completion
2. correctness of incremental sanity check

of an incremental Merkle forest, which is used in the event ordering system [5] developed by NTT. Especially, this paper is the first to prove (2) the correctness of an incremental sanity check.

During the proofs, we mainly used the automata-based theorem prover MONA [1]. Although MONA can treat only decidable properties, this does not mean that its use is easy. We need to find suitable formalization and key lemmata, which are essential in the whole proof and still provable by MONA. For instance, during the use of MONA, we have also simplified the manual proof of Theorem 1 (the original proof, by induction on the homogeneous depth of a Merkle tree, takes more than 1 page in two-column style).

Another notable example of $WSnS$ is an optimal reduction strategy of a strongly sequential term rewriting system [6]. This is known to be intricate; however it was clearly re-described in terms of $WSnS$ [4].

A drawback is that an automata-based prover does not give a deductive proof. Thus, incomplete descriptions may be easily neglected; instead, we often found them by test data. On the other hand, it is extremely powerful for detecting oversights and gaps in a proof draft, which are often found in tentative proof goals. At the moment, support for theorem prover is not enough; but, we feel it is possible for theorem provers to be an assistance even for constructing a new proof.

For future work, we are planning:

– full formal proof of Theorem 2 by combining MONA and an induction-based prover *Isabelle/HOL* [10].
– proofs for more detailed properties of the event ordering system.

Acknowledgments

This research is partially supported by Special Coordination Funds for Promoting Science and Technology and Scientific Research on Priority Area (No.16016241) by Ministry of Education, Culture, Sports, Science and Technology, PRESTO by Japan Science and Technology Agency, and Kayamori Foundation of Informational Science Advancement.

References

1. MONA project. `http://www.brics.dk/mona/`.
2. C. Adams and et al. RFC3161, internet X.509 public key infrastructure time-stamp protocol (TSP). Technical report, IETF, 2001.
3. A. Buldas, H. Lipmaa, and B. Schoenmakers. Optimally efficient accountable time-stamping. In *Proc. 3rd International Workshop on Practice and Theory in Public Key Cryptography (PKC 2000)*, pages 293–305. Springer-Verlag, 2000. Lecture Notes in Computer Science, Vol.1751.
4. H. Comon. Sequentiality, monadic second-order logic and tree automata. *Information and Computation*, 157(1 & 2):25–51, 2000. Previously presented in *Proc. 10th IEEE Symposium on Logic in Computer Science*, pages 508–517, 1995.
5. E. Horita, S. Ono, and H. Ishimoto. Implementation mechanisms of scalable event-ordering system without single point of attack. Technical report, IEICE SIG-ISEC, 11 2004. *in Japanese.*
6. G. Huet and J.-J. Lévy. Computations in orthogonal rewriting systems I,II. In *Computational Logic: Essays in Honor of Alan Robinson*, pages 395–443. MIT Press, 1991. Previous version: Report 359, INRIA, 1979.
7. M. Jakobsson, F.T. Leighton, S. Micali, and M. Szydlo. Fractal Merkle tree representation and traversal. In *Proc. Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003*, pages 314–326. Springer-Verlag, 2003. Lecture Notes in Computer Science, Vol.2612.
8. R.C. Merkle. *Secrecy, Authentication, and Public Key Systems.* UMI Research Press, 1982. also appears as Ph.D thesis at Stanford University, 1979.
9. S. Michael. Merkle tree traversal in log space and time. In *Proc. International Conference on the Theory and Applications of Cryptographic Techniques, Advances in Cryptology - EUROCRYPT 2004*, pages 541–554. Springer-Verlag, 2004. Lecture Notes in Computer Science, Vol.3027.
10. T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL, A proof assistant for higher-order logic.* Springer-Verlag, 2002. Lecture Notes in Computer Science, Vol.2283.
11. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 4, pages 133–192. Elsevier, 1990.
12. J. Villemson. *Size-efficient interval time stamps.* PhD thesis, University of Tartu, Estonia, 2002.