Behavior-driven Authenticated Data Structure

Kevin Atighehchi, Alexis Bonnecaze, Traian Muntean

I2M CNRS UMR 7373, ERISCS
YACC 2014, Porquerolles

June 12, 2014

Introduction

The term “authenticated dictionary“:

- Dictionary: organizes, manages a collection of data, and answers to queries on data.
- Authenticated: answers to queries are certified.

Many applications:

- Certificate revocation in public key infrastructures,
- Geographic information system querying,
- Third party data publication and validation on Internet.

This last application is of great interest for the Internet users (*e.g.* Content Distribution Networks (CDN) and authenticated Web site data).

Three-party authentication model

Generally, three actors are involved:

- A trusted *source*,
- A (potentially) untrusted *provider*,
- A set of *users*.

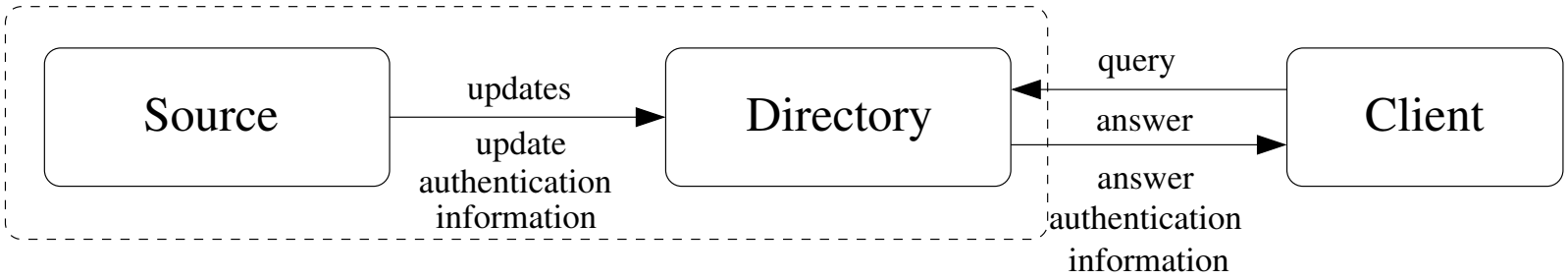


Figure: The three-party authentication model

Distribution of data and authenticated dictionaries

Most of authenticated dictionaries use Merkle trees, red-black trees or skip-lists as data structures.

- Well adapted as long as no distinction is made between data.
- Most Web traffic follows Zipf's law except for the traffic residue.

Useit.com traffic during a eight-week period (log-log scale):

Background

Authenticated dictionaries:

- A set S of pair elements of the form (Identifier, Content).
- Request of an element or membership query from the user.
- Cryptographic proof of membership/non-membership, content authenticity.
- Non-repudiation.

Data structures:

- Static.
- Dynamic: 2-3 Trees, B-Trees or red-black trees, non-deterministic skip-lists.
- Append/disjoin-only: variant of a Merkle tree with non-power of two leaves.

Authenticated dictionary based on frequency

On the Internet, some Web pages are consulted more frequently than others. In our scheme the size of authentication proof is smaller when the frequency of the query is higher.

Benefits:

- For the directory, reduction of the LAN/WAN interface bandwidth usage.
- The directory can cache proofs frequently queried for better usage of memory.
- For a given user, reduction of both the LAN/WAN interface bandwidth and the number of calculations to verify a proof.

Authenticated data structure construction

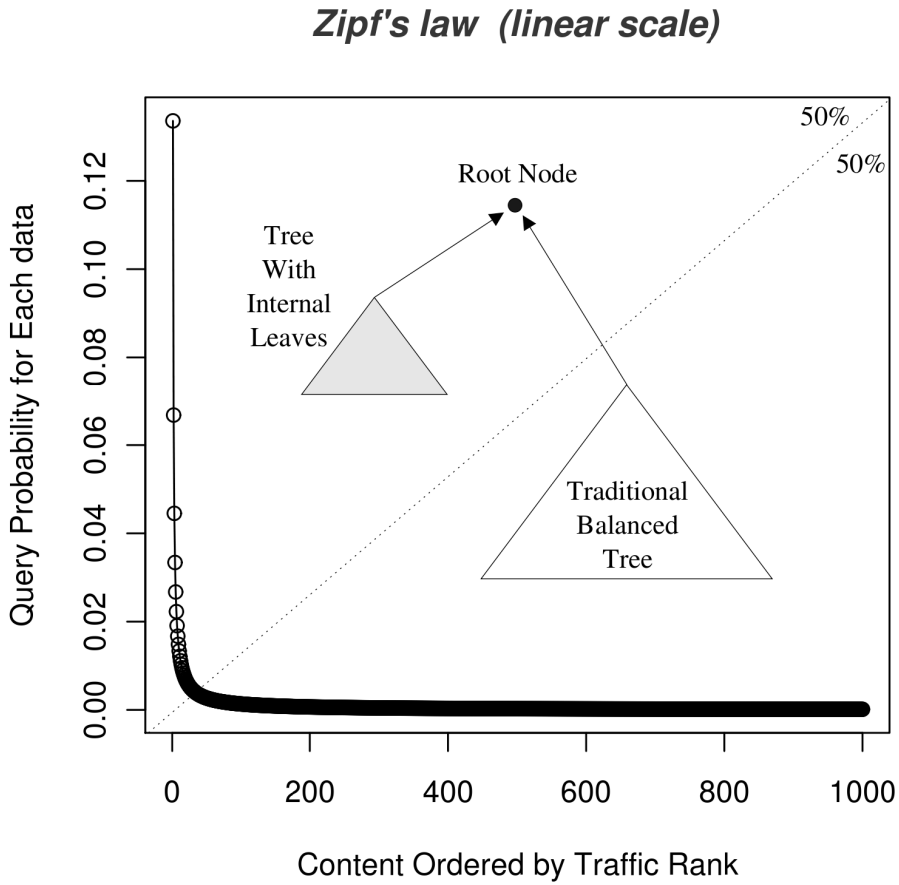
Our scheme relies on the following data structures.

- Two dynamic binary trees A_1 and A_2:
  - A_1: ranks the hashed identifiers $(u_i)_{i=1\ldots n}$ in ascending order and allows us to search a given u_i and to retrieve its corresponding content, or frequency.
  - A_2: arrange frequencies $(f_i)_{i=1\ldots n}$ in decreasing order and allows the rank of a given frequency to be retrieved.
- Authentication proofs are constructed using the third data structure, denoted T.

Authenticated data structure construction (Ctd)

T is formed of the following two subtrees:

- T_1: the left child, is a height-balanced tree with internal leaves, used for the most frequent data.
- T_2: right child of T, is a Merkle-like tree, used for very low frequent data.



Authenticated data structure construction (Ctd)

The source and the directory construct the following ordered sets:

- L^u ranks elements by ascending order of hashed identifier

$$L^u = \{(u_1, f_{\Pi(1)}), \ldots, (u_n, f_{\Pi(n)}), (+\infty, 0)\}.$$

- L^f ranks elements by descending order of frequency

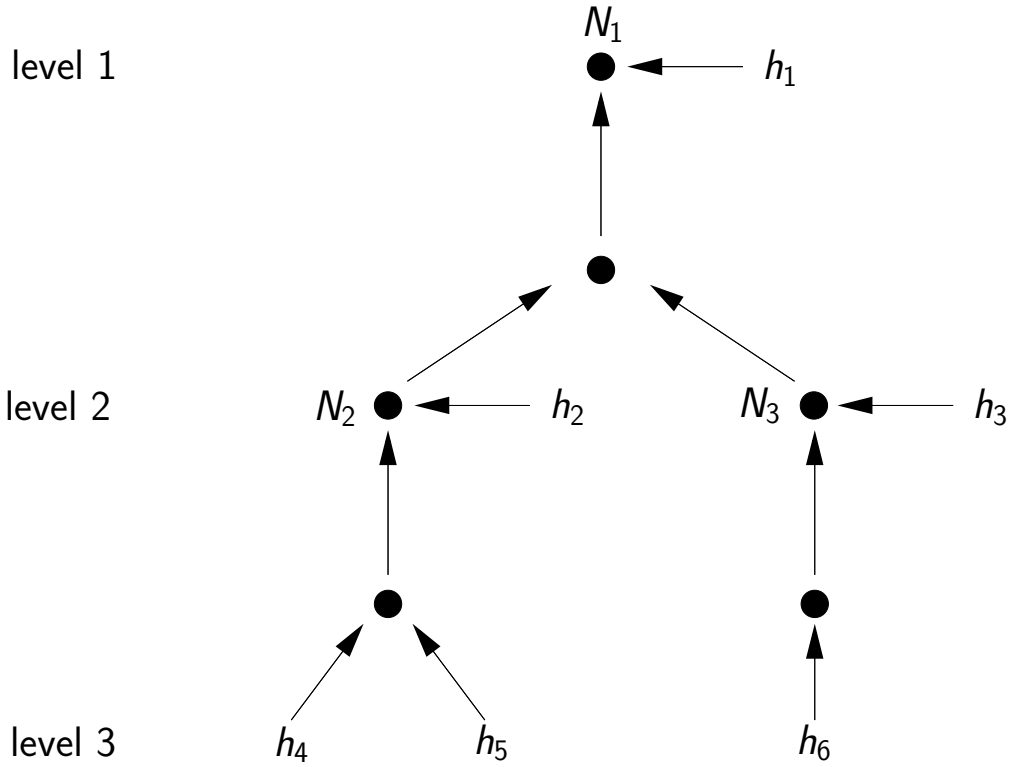
$$L^f = \{(u_{\Pi^{-1}(1)}, f_1), \ldots, (u_{\Pi^{-1}(n)}, f_n), (u_{\Pi^{-1}(n)}, f_n), (+\infty, 0)\}.$$

From these lists, the source calculates the tree T. Calculation of a leaf h_i is done using a pair-wise chaining, as follows:

- $h_{\Pi(1)} = H(-\infty, u_1, c_1)$,
- $h_{\Pi(i)} = H(u_{i-1}, u_i, c_i)$ where $i \in [2, \ldots, n]$,
- $h_{\Pi(+\infty)} = H(u_n, +\infty, 0)$.

Authenticated data structure construction (Ctd)

The tree T_1 is constructed using the most frequent data (until the median frequency).



The remaining data serve to construct a Merkle-like tree, denoted T_2.

Algorithms

To manage the dictionary and communicate with a client, the following algorithms are employed:

- Proof of existence.
- Proof of non-existence.
- Verification.
- Updating:
  - Insertion.
  - Modification.
  - Reordering and deletion.

Proof construction and verification algorithms

- Proof of existence of u_i: Insert in a list u_{i-1} and the sibling nodes (or sibling leaves) of its parent (or ancestor) nodes from the base level to the root node.
- Proof of non-existence of u: Search the lowest j such that $u_j > u > u_{j-1}$ and construct the proof of existence of u_j.
- The verification consists to:
  - Recursively hash these listed values.
  - Check the consistency of the computed root node with the signed one.

Updating algorithms

Insertion of a new pair and modification of a content

Insertion of a new *pair element* $(Id,\ c)$ where $H(Id) \notin (u_i)_{i=1\ldots n}$:

- Two new leaves are computed:
  - One leaf is inserted in T_2.
  - An existing leaf is updated to restore the pair-wise chaining.
- Internal nodes corresponding to paths from each of these two leaves to the root node of T are recomputed.

The content of an existing element (Id, c) is changed to c':

- One leaf is updated.
- The nodes of the path from the updated leaf up to the root node of T are recomputed.

Updating algorithms

Reordering of elements and deletion

Reordering elements in T. Example: The leaf belongs to T_1 and will stay in T_1

- A simple solution is to move the corresponding leave at the correct position and shift a consequent part of leaves.
  - Example: Move the leaf of the element of frequency f_m $(m > i)$ such that $f_i < f'_m < f_{i+1}$. .
  - Cost in $\mathrm{O}(n \log n)$.
- A better solution is to break the order of elements in T_1 and to satisfy the following relaxed order property:

$$\forall\ i = 1 \ldots h-1,\ \forall\ Id_x \in S_i,\ Id_y \in S_{i+1}\ f(Id_x) \geq f(Id_y).$$

Example ("min-max" choice criteria):
A leaf authenticating an element $e = (Id, C(Id))$ belonging to level i must move up to level j $(i > j)$.

1. This leaf is inserted at level j at the position of the leaf having the lowest frequency in this level.
2. This last element is moved down to level $j + 1$ at the position of the leaf having the lowest frequency, and so on.
3. The leaf having the lowest frequency at level $i - 1$ is moved up to the former position of e at level i.
4. Finally, nodes which are on the path of the leaves that have been moved are recomputed back up the root of the tree.

Complexity analysis

Authentication proof size and verification run time

The operation costs are expressed in terms of number of hash operations.

Theorem (Worst case proof size)

Considering a number of elements $n \geq 1$, the authentication proof is of length 3 in the best case, and of length in $O(\log n)$ in the worst case.

Numerical results:

Dictionary size	Merkle-like structure	Our system	Improvement
10^3	9.97	8.05	19.5%
$5 \cdot 10^4$	15.61	12.25	22.5%
$5 \cdot 10^5$	18.93	14.73	22.5%
10^6	19.93	15.46	22.5%

Table: Average proof size and verification cost results

Complexity analysis (Ctd)

Modification, insertion, frequency changes and deletion

Theorem (Modification of a content or insertion of a new element of frequency $f = 0$)

The number of hash evaluations to update T is in $\mathrm{O}(\log n)$ where n is the overall number of elements in the dictionary.

Theorem (A single frequency change or a deletion an element)

By using the "min-max" choice criteria, the number of hash computations to update T is in $\mathrm{O}(\log^2 n)$.

A solution to authenticate HTTP responses

The server returns either the requested page together with a 200 success response or a 404 error message and a proof of authenticity of the content of that page (or possibly a proof of non existence):

- u_i= digest of a url (Uniform Resource Locator),
- c_i= content digest of the corresponding page.

For a dynamic site, some works have to be done to allow the use of our dictionary:

- Individual authentication for each static object of the page (multiple proofs=consequent overhead).
- A hash scheme to generate the same hash value for the set of static fields of interest in the considered page.

Conclusion

Our proposition is an authenticated dictionary with the following features:

- A data structure with two components, each of which being nearly optimal for a portion of the distribution.
- Compared to the use of a Merkle tree, smaller proof sizes (average gain of more than 20%) when the requests distribution follows Zipf law.
- Comparable proof sizes otherwise.

Improvements ? Length-limited Huffman trees or variants of dynamic Huffman trees. An average gain of 30% for the proof size is expected.

Future work: a complete scheme to authenticate http responses.

Thanks for your attention!