Verkle Trees: Ver(y Short Mer)kle Trees

John Kuszmaul
Mentored by Alin Tomescu
PRIMES Conference - 5/19/2019
Storing Files Remotely

Alice sends her files $F_0$, $F_1$, …, $F_n$. 

Alice

Dropbox
Storing Files Remotely

Alice

What is $F_i$?

Here you go: $F_i$.

Dropbox
Proving/Verifying Integrity (or Correctness)

Alice generates a digest $d$ of her files.

Alice sends her files $F_0$, $F_1$, ..., $F_n$.
Proving/Verifying Integrity

Alice verifies the proof $\pi_i$ against $d$ to make sure $F_i$ has not been modified.
Secure Hash Functions

Original File $F_i$

- Bob owes Alice $70k.$
- Hash Function
- $H(F_i) = 011010...110$
- 256 bits

Corrupted File $F'_i$

- Bob owes Alice $20.$
- Hash Function
- $H(F'_i) = 100111...101$
- 256 bits

Ideally, finding any two distinct files, $F_1, F_2$, s.t.

$H(F_1) = H(F_2)$

takes $2^{128}$ attempts.
A Simple Scheme for Verifying File Integrity

Alice hashes each of her files:

- \( H(F_0) \) to \( F_0 \)
- \( H(F_1) \) to \( F_1 \)
- \( H(F_2) \) to \( F_2 \)
- \( H(F_3) \) to \( F_3 \)
- \( H(F_4) \) to \( F_4 \)
- \( H(F_5) \) to \( F_5 \)
- \( H(F_6) \) to \( F_6 \)
- \( H(F_7) \) to \( F_7 \)
Alice computes and stores the hashes locally.

Alice sends her files $F_0, F_1, \ldots, F_n$.

Dropbox

Proving/Verifying Integrity: Simple Scheme
Alice computes $H(F_i)$ and checks that it equals stored $H(F_i)$. What is $F_i$?
Problem: Alice has to store $n$ hashes.

Alice’s digest must be constant-sized.
Solution: Merkle Trees

The root is the digest.

$h_{14} = H(h_{12}, h_{13})$

$h_{12} = H(h_8, h_9)$

$h_8 = H(h_0, h_1)$

$h_9 = H(h_2, h_3)$

$h_{13} = H(h_{10}, h_{11})$

$h_{10} = H(h_4, h_5)$

$h_{11} = H(h_6, h_7)$

$h_0 = H(F_0)$

$h_1 = H(F_1)$

$h_2 = H(F_2)$

$h_3 = H(F_3)$

$h_4 = H(F_4)$

$h_5 = H(F_5)$

$h_6 = H(F_6)$

$h_7 = H(F_7)$
Alice computes the Merkle tree and stores the root locally.

Alice sends her files $F_0$, $F_1$, $\ldots$, $F_n$.

Proving/Verifying Integrity: Merkle Tree
Proving/Verifying Integrity: Merkle Tree

What is $F_i$?

How does Dropbox respond with a proof?
Merkle Proofs

$h_{12} = H(h_8, h_9)$

$h_8 = H(h_0, h_1)$

$h_0 = H(F_0)$

$h_1 = H(F_1)$

$h_2 = H(F_2)$

$h_3 = H(F_3)$

$h_4 = H(F_4)$

$h_5 = H(F_5)$

$h_6 = H(F_6)$

$h_7 = H(F_7)$

$h_9 = H(h_2, h_3)$

$h_{10} = H(h_4, h_5)$

$h_{11} = H(h_6, h_7)$

$h_{12} = H(h_8, h_9)$

$h_{13} = H(h_{10}, h_{11})$

$h_{14} = H(h_{12}, h_{13})$

Dropbox sends these highlighted nodes.
Proving/Verifying Integrity: Merkle Tree

Alice

d = root

The Proof

F_3 \quad H(h_0, h_1) \quad H(F_2) \quad H(h_{10}, h_{11})

Dropbox
Verifying the Proof

Alice computes the root starting from $F_3$ with these highlighted proof.
Verifying the Proof

Alice hashes up the tree.

\[ h_{12} = H(h_8, h_9) \]

\[ h_8 = H(h_0, h_1) \]

\[ h_9 = H(h_2, h_3) \]

\[ h_2 = H(F_2) \]

\[ h_3 = H(F_3) \]

\[ h_{14} = H(h_{12}, h_{13}) \]

\[ h_{13} = H(h_{10}, h_{11}) \]

\[ h_{12} = H(h_{10}, h_{11}) \]

\[ h_{14} = H(h_{12}, h_{13}) \]

\[ h_9 = H(h_2, h_3) \]

\[ h_8 = H(h_0, h_1) \]

\[ h_2 = H(F_2) \]

\[ h_3 = H(F_3) \]

\[ F_3 \]
Verifying the Proof

h_{12} = H(h_8, h_9)

h_8 = H(h_0, h_1)

h_9 = H(h_2, h_3)

h_2 = H(F_2)

h_3 = H(F_3)

F_3

h_{13} = H(h_{10}, h_{11})

h_{14} = H(h_{12}, h_{13})

Alice hashes up the tree.
Verifying the Proof

Alice hashes up the tree.

$h_{12} = H(h_8, h_9)$

$h_8 = H(h_0, h_1)$

$h_9 = H(h_2, h_3)$

$h_{13} = H(h_{10}, h_{11})$

$h_2 = H(F_2)$

$h_3 = H(F_3)$

$h_{14} = H(h_{12}, h_{13})$

$F_3$
Verifying the Proof

Alice checks if the Merkle Root = d

- $h_{14} = H(h_{12}, h_{13})$
- $h_{12} = H(h_8, h_9)$
- $h_8 = H(h_0, h_1)$
- $h_9 = H(h_2, h_3)$
- $h_{13} = H(h_{10}, h_{11})$
- $h_2 = H(F_2)$
- $h_3 = H(F_3)$
- $F_3$

F₃ has not been modified!

Time to stop using Dropbox!
Everyone loves Merkle Trees!

- They’re beautiful.
- They’re efficient.

\[
n = \text{number of leaves (files)}
\]

<table>
<thead>
<tr>
<th>Merkle Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construct Tree</td>
</tr>
<tr>
<td>Proof size</td>
</tr>
<tr>
<td>Update File</td>
</tr>
</tbody>
</table>
Problem: Many small files \(\Rightarrow\) Merkle proofs too large.
Problem: Many small files $\Rightarrow$ Merkle proofs too large.

- Suppose Alice has one billion $\approx 2^{30}$ files.
Problem: Many small files ⇒ Merkle proofs too large.

- Suppose Alice has one billion ≈ $2^{30}$ files.

Merkle Proof: ~ 1 KB (in addition to the file itself)
Possible Solution: q-ary Merkle Tree

Example: 3-ary tree

\[ h_3 = H(h_0, h_1, h_2) \]

\[ h_0 = H(F_0, F_1, F_2) \]
\[ h_1 = H(F_3, F_4, F_5) \]
\[ h_2 = H(F_6, F_7, F_8) \]
Problem: The Proof Becomes Bigger, $O(q \log_q n)$

Example: 3-ary tree

$h_3 = H(h_0, h_1, h_2)$

$h_0 = H(F_0, F_1, F_2)$

$h_1 = H(F_3, F_4, F_5)$

$h_2 = H(F_6, F_7, F_8)$

$F_0$  $F_1$  $F_2$  $F_3$  $F_4$  $F_5$  $F_6$  $F_7$  $F_8$
Our Work: Verkle Trees reduce the proof size

- We pick a $q$.
- We reduce the proof size from $\log_2 n$ to $\log_q n = \log_2 n / \log_2 q$.
- Factor of $\log_2 q$ less bandwidth!
- At the cost of $q$ times more computation to construct.
- Proof verification is $\log_2 q$ times faster.

Wow, that’s big!
Does this matter?  (Hint: Yes)

- Merkle hash trees are everywhere in cryptography:
  - Consensus Protocols
  - Public-Key Directories
  - Cryptocurrencies
  - Encrypted Web Applications
  - Secure File Systems
Vector Commitment (VC) Schemes by Catalano and Fiore (2013)

Each file has a constant-sized proof (\(\pi\)). Commitment (C) is the digest.
VC Schemes are Computationally Impractical

<table>
<thead>
<tr>
<th>Scheme/op</th>
<th>Construct</th>
<th>Proof size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merkle</td>
<td>$O(n)$</td>
<td>$O(\log_2 n)$</td>
</tr>
<tr>
<td>VC scheme</td>
<td>$O(n^2)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
Our Solution: Replace Hash Functions with VC Schemes

This is the Verkle Tree.
We now have a Verkle Tree!

We get to choose the branching factor, $q$, to be whatever we want!

The root commitment is the digest.
Alice Receives $\log_q n$ Constant-Sized $\pi$'s.

Alice verifies:
1. VC Proof from $F_2$ to $C_1$: $\pi_2$
2. VC Proof from $C_1$ to $C_4$: $\pi_9$
Comparison

<table>
<thead>
<tr>
<th>Scheme/op</th>
<th>Construct</th>
<th>Update file</th>
<th>Proof size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merkle</td>
<td>$O(n)$</td>
<td>$O(\log_2 n)$</td>
<td>$O(\log_2 n)$</td>
</tr>
<tr>
<td>q-ary Merkle</td>
<td>$O(n)$</td>
<td>$O(q \log_q n)$</td>
<td>$O(q \log_q n)$</td>
</tr>
<tr>
<td>VC scheme</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>q-ary Verkle</td>
<td>$O(qn)$</td>
<td>$O(q \log_q n)$</td>
<td>$O(\log_q n)$</td>
</tr>
</tbody>
</table>

Verkle Trees let us trade off proof-size vs. construction time.
My Contribution

- I proved complexity bounds for Verkle Trees.
- I implemented and optimized Verkle Trees in C++.
- Benchmarked implementations.
Prefix Tree

![Graph showing the relationship between construction time and tree size](image)

- **Y-axis:** Construction Time (seconds)
- **X-axis:** Tree Size (Leaves)

The graph plots the construction time in seconds against the tree size, which is measured in the number of leaves. The data points indicate a linear relationship, suggesting that the construction time increases as the tree size grows.
Prefix Tree vs. History Tree
Prefix Tree vs. History Tree vs. Parallelized History

![Graph showing the comparison of construction time (seconds) for Prefix, History, and History (2 Cores) with varying tree sizes (Leaves).]
On a Linear Scale:
Acknowledgements

- Thank you Alin!
- Thank you PRIMES!
- Thank you Mom and Dad!