Verkle Trees: 
Ver(y Short Mer)kle Trees

John Kuszmaul
Mentored by Alin Tomescu
PRIMES Computer Science Conference - 10/13/18
Storing Files Remotely

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

Alice

What is $F_i$?

Here you go: $F_i$`
Alice generates a digest $d$ of her files. Alice sends her files $F_0$, $F_1$, ..., $F_n$. Dropbox
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:

\[
\begin{align*}
H(F_0) & \rightarrow F_0 \\
H(F_1) & \rightarrow F_1 \\
H(F_2) & \rightarrow F_2 \\
H(F_3) & \rightarrow F_3 \\
H(F_4) & \rightarrow F_4 \\
H(F_5) & \rightarrow F_5 \\
H(F_6) & \rightarrow F_6 \\
H(F_7) & \rightarrow F_7
\end{align*}
\]
Alice computes and stores the hashes locally. Alice sends her files $F_0$, $F_1$, …, $F_n$.

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

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

The root is the digest.

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

\[ h_8 = \text{H}(h_0, h_1) \]

\[ h_9 = \text{H}(h_2, h_3) \]

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

\[ h_{10} = \text{H}(h_4, h_5) \]

\[ h_{11} = \text{H}(h_6, h_7) \]
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

Alice sends her files $F_0, F_1, \ldots, F_n$. 
Proving/Verifying Integrity: Merkle Tree

What is $F_i$?

How does Dropbox respond with a proof?
Merkle Proofs

Dropbox sends these highlighted nodes.

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

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

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

$h_8 = H(h_0, h_1)$

$h_9 = H(h_2, h_3)$

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

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

$h_0 = H(F_0)$

$h_1 = H(F_1)$

$h_2 = H(F_2)$

$h_3 = H(F_3)$

$h_4 = H(F_4)$

$h_5 = H(F_5)$

$h_6 = H(F_6)$

$h_7 = H(F_7)$

$F_0$

$F_1$

$F_2$

$F_3$

$F_4$

$F_5$

$F_6$

$F_7$
Proving/Verifying Integrity: Merkle Tree

The Proof

\[
F_3, \quad H(h_0, h_1), \quad H(F_2), \quad H(h_{10}, h_{11})
\]
Verifying the Proof

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

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

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

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_{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.

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

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

$h_8 = H(h_0, h_1)$

$h_9 = H(h_2, h_3)$

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

$h_{12} = H(F_2)$

$h_{13} = H(F_3)$

$F_3$
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 checks if the Merkle Root = \( d \)

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

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

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

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

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

\[ h_2 = H(F_2) \]

\[ h_3 = H(F_3) \]

\[ F_3 \]

F_3 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></th>
<th>Merkle Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construct Tree</td>
<td>O(n)</td>
</tr>
<tr>
<td>Proof size</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Update File</td>
<td>O(log n)</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 $\Rightarrow$ Merkle proofs too large.

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

Merkle Proof: $\sim 1$ 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

$h_0 = H(F_0, F_1, F_2)$
$h_1 = H(F_3, F_4, F_5)$
$h_2 = H(F_6, F_7, F_8)$

$h_3 = H(h_0, h_1, h_2)$
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
- (e.g., $q = 1024 \Rightarrow \log_2(q) = 10x$ less bandwidth)

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)

Commitment (C) is the digest.

Each file has a constant-sized proof (π).

$F_0, \pi_0$, $F_1, \pi_1$, $F_2, \pi_2$, $F_3, \pi_3$, $F_4, \pi_4$, $F_5, \pi_5$, $F_6, \pi_6$, $F_7, \pi_7$, $F_8, \pi_8$
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 $\log_2 n$ Constant-Sized $\pi$’s.

Alice verifies:
1. VC Proof from $F_2$ to $C_1$: $\pi_2$
2. VC Proof from $C_1$ to $C_4$: $\pi_9$
Verkle Trees let us trade off proof-size vs. construction time.
My Contribution

- I proved complexity bounds for Verkle Trees.
- I implemented Verkle Trees in C++.
- I am measuring performance.
Acknowledgements

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