Slightly Improved Merkle Tree Traversal for User Authentication Using Pseudorandomly-Generated Leaves

Douglas Stebila*

June 9, 2006

Abstract

Merkle trees can be used for a variety of cryptographic purposes, including digital signatures and user authentication. We describe a technique for pseudorandomly generating the private values corresponding to the leaves of a Merkle tree from a common seed. The technique allows for secure user authentication with fewer hash function evaluations required to generate the authentication data. Using the pseudorandomly-generated private values is as secure as using randomly-generated private values provided that the hash function cannot be inverted.

A *Merkle tree* [Mer90] on a set of nodes V is a complete binary tree with 2^H leaves, along with a function $\Phi : V \to \{0,1\}^k$ assigning values to each node of the tree. For each interior node n_{parent} with left and right child nodes n_{left} and n_{right},

$$\Phi\left(n_{\text{parent}}\right) = h\left(\Phi(n_{\text{left}})||\Phi(n_{\text{right}})\right) \quad ,$$

where $h : \{0,1\}^* \to \{0,1\}^k$ is a cryptographic hash function and $||$ denotes string concatenation. For each leaf node $leaf_i$, the value $\Phi(leaf_i)$ is

$$h\left(LEAFCALC(leaf_i)\right)$$

where $LEAFCALC(leaf_i)$ represents a call to an oracle to compute the private value of the leaf node $leaf_i$. Once the output values of the oracle $LEAFCALF$ is fixed, the Φ values for all nodes in the tree are determined.

Merkle trees, when combined with cryptographic hash functions, can be used for a variety of purposes, including digital signatures [DSS05] and user authentication.

In user authentication, a user wishes to repeatedly authenticate herself with a server. This can be accomplished by providing the server with the value $\Phi(n_{\text{root}})$ for the root of the Merkle tree. Each time the user wishes to authenticate to the

*Institute for Quantum Computing, University of Waterloo, Waterloo, ON, Canada, N2L 1W4; email dstebila@iqc.ca. Supported by NSERC and Sun Microsystems.

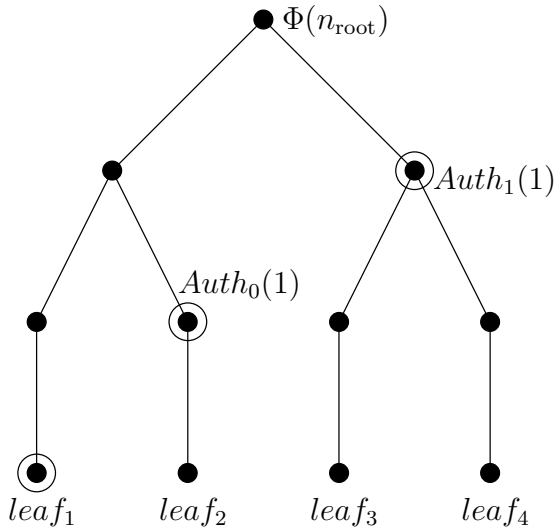

Figure 1: Merkle tree on 4 leaves showing the authentication path for $leaf_1$.

server, the user reveals the value of $LEAFCALC(leaf_i)$ for the next leaf node i of the tree, along with an *authentication path* of the remaining values of Φ from the tree so that the server can construct and verify $\Phi(n_{\text{root}})$. The values in the authentication path are defined as follows. Let $n_{j,k}$ denote the kth node from the left at height j in the tree; so the left-most leaf node is $n_{0,1}$. Let $Auth_j(i)$ be the value of Φ on the node that is the sibling to the node at height j in the path from $leaf_i$ to the root. For example, $Auth_0(i) = \Phi(n_{0,i'})$ where $leaf_i$ and $leaf_{i'}$ are siblings. The authentication path for $leaf_i$ is then

$$Auth(i) = (LEAFCALC(leaf_i), Auth_0(i), Auth_1(i), \ldots, Auth_h(i)) \quad .$$

The authentication path can be verified by checking if the published root value, $\Phi(n_{root})$, is equal to

$$h\left(\ldots h\left(LEAFCALC(leaf_i)||Auth_0(i)\right)\ldots||Auth_{H-1}(i)\right) \quad ,$$

where the order of concatenation is done correctly based on the structure of the tree.

The process of generating authentication paths $Auth(i)$ for subsequent leaves is called *Merkle tree traversal*. Typically, the leaves are revealed in a predetermined order, for example from left to right, $i = 1, 2, \ldots, 2^H$. An asymptotically optimal algorithm for Merkle tree traversal was given by Szydlo [Szy04] which requires $2H$ hash function evaluations for each authentication path revealed, along with $O(1)$ calls to the oracle $LEAFCALC$. Szydlo's algorithm

allocates these $2H$ hash function evaluations on each iteration and guarantees that each authentication path will be ready by the iteration it needs to be revealed.

Suppose the $LEAFCALC$ values for the leaf nodes are generated pseudorandomly from a secret seed value s using the same hash function h as in the Merkle tree construction. If these values are generated left-to-right so that $LEAFCALC(leaf_i) = h^{i+1}(s)$, where h_c denotes the function h applied c times, then any party that observes the authentication path revealed for a leaf node $leaf_j$ can construct values of $LEAFCALF(leaf_k)$ and the corresponding authentications paths for any $k > j$.

However, suppose the $LEAFCALC$ values are generated pseudorandomly from right-to-left, so

$$LEAFCALC(leaf_i) = h^{2^H - i}(s) \ .$$

For leaf nodes $leaf_i$ that are left children of their parent nodes, the authentication value is

$$\begin{aligned} Auth_0(i) &= h(LEAFCALC(leaf_{i+1})) \\ &= h\left(h^{2^H-(i+1)}(s)\right) \\ &= h^{2^H-i}(s) \\ &= LEAFCALC(leaf_i) \ . \end{aligned}$$

This allows us to save a hash function evaluation for half of the authentication path calculations and also to shorten the authentication path.

Thus, for half of the authentication path calculations we need only $2H - 1$ hash function evaluations, and for the remaining calculations we still need $2H$ hash function evaluations. Overall, we need

$$(2H-1) \cdot 2^{H-1} + 2H \cdot 2^{H-1} = (2H - \frac{1}{2}) \cdot 2^H$$

hash function evaluations instead of $2H \cdot 2^H$.

A forger cannot generate an authentication path for $leaf_i$ given authentication paths for $leaf_1, \ldots, leaf_{i-1}$ without determining $LEAFCALC(leaf_i)$, and determining this value requires inverting the hash function h.

This technique is useful for authentication when a user repeatedly proves identity to the same party. However, it cannot be used in the various hash-based digital signature schemes (e.g., [DSS05]). In the hash-based digital signatures schemes, only a subset (corresponding to the message to be signed) of the bits of $LEAFCALC(leaf_i)$ are revealed in the authentication path. If, as in our technique above, $Auth_0(i) = LEAFCALC(leaf_i)$ for some nodes, all of the bits of $LEAFCALC(leaf_i)$ are revealed in the authentication path and any message can then be forged using the key from $leaf_i$. Moreover, for digital signature schemes, the size of $LEAFCALC(leaf_i)$ must be considerably larger than the output of the hash function.

References

[DSS05] C. Dods, N. P. Smart, and M. Stam. Hash-based digital signatures schemes. In N. P. Smart, ed., *Cryptography and Coding*, volume 3796 of *LNCS*, pp. 96–115. Springer-Verlag, 2005.

[Mer90] R. C. Merkle. A certified digital signature (subtitle: That antique paper from 1979). In G. Brassard, ed., *Advances in Cryptology – Proc. CRYPTO '89*, volume 435 of *LNCS*, pp. 218–238. Springer-Verlag, 1990.

[Szy04] M. Szydlo. Merkle tree traversal in log space and time. In C. Cachin and J. Camenisch, eds., *Advances in Cryptology – Proc. EUROCRYPT '04*, volume 3027 of *LNCS*, pp. 541–554. Springer-Verlag, 2004.