A Note on Traversing Skew Merkle Trees

Marek Karpinski* Yakov Nekrich†

Abstract. We consider the problem of traversing *skew* (unbalanced) Merkle trees and design an algorithm for traversing a skew Merkle tree in time $O(\log n/h)$ and space $O(\log n(2^h/h))$. This algorithm can be of special use in situations when exact number of items to be identified is not known in advance.

1 Introduction

Merkle tree is a complete binary tree such that the values of internal nodes are one-way functions of the values of their children. Every leaf value in a Merkle tree can be verified with respect to a publicly known root and the *authentication path* of that leaf. An authentication path of a leaf consists of the siblings of all nodes on the path from this leaf to the root.

Merkle trees can be used in different cryptographic applications and in many such applications the so-called Merkle tree traversal, i.e. consecutive generation of authentication paths for all tree leaves, must be performed.

The problem of Merkle tree traversal for balanced trees, i.e. trees of logarithmic height , was considered in a number of papers, e.g. [S03], [JLMS03], [BKN04].

*Dept. of Computer Science, University of Bonn. Work partially supported by DFG grants , Max-Planck Research Prize, DIMACS and IST grant 14036 (RAND-APX). E-mail marek@cs.uni-bonn.de

†Dept. of Computer Science, University of Bonn. Work partially supported by IST grant 14036 (RAND-APX). E-mail yasha@cs.uni-bonn.de

In this note we introduce a notion of a *skew Merkle tree*. In the skew Merkle tree different leaves are allowed to have different depths. We show how Merkle tree traversal algorithms can be extended to the case of certain *skew Merkle trees* without additional time and memory resources.

We can imagine a skew Merkle tree as a traditional Merkle tree with some extra nodes appended to some of the leaves. In other words, in this paper we show that extra nodes can be appended to the balanced Merkle tree without changing the time and space bounds of the traversal algorithm.

2 Algorithm Description

In the argumentation below we will use the same notation as in [BKN04]. We denote by n the minimal number of items in a skew Merkle tree and $H = \log n$. Parameter h is the height of the subtree that is stored after every node computation (see [JLMS03] and [BKN04] for an explanation of this concept).

The key idea of our traversal method is that we do not have to re-compute subtrees on higher levels from a certain time. Therefore we can use the resources wasted in the case of balanced trees and append some additional nodes. In [BKN04] and [JLMS03] subtree levels were indexed by numbers between 1 and $L = H/h$. In this note we modify the definition of tree level and we also use zero or negative subtree levels to specify the levels of appended subtrees. We say that a node is on level k if its depth $d = H - kh$.

We say that the authentication path for a leaf j is output during the j-th round. We will also use the notion of *superround* . During the k-th superround the algorithm outputs authentication paths of all descendants of the k-th node of depth H. Please note that since the different leaves in a tree have different depths the number of authentication paths output during a superround can vary. Thus the algorithm consists of 2^H superrounds and each superround consists of a variable number of rounds.

For easiness of description we start by considering the case of $h = 1$ and show that for $h = 1$ almost $n \log n/2$ nodes can be added. The general case will be considered later.

Since we do not have to compute the subtree on the $H-1$-th level during the last 2^{H-1} superrounds, we have two spare computation units during each of these superrounds. We can use these units to attach two extra leaves to leaves $2^{H-1}+2, \ldots, 2^H$. In other words, during superrounds $2^{H-1}+1, \ldots, 2^H$

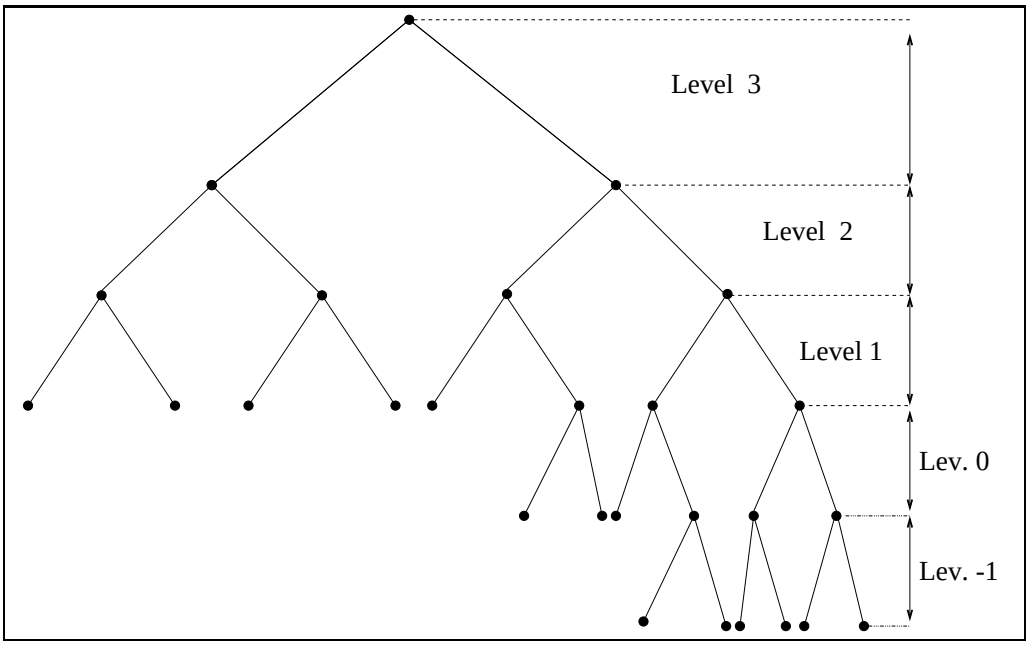

Figure 1: Example of a skew Merkle tree for $h = 1$ with two appended levels

we compute nodes on levels $0, 1, 2, \ldots, H-2$ instead of computing nodes on levels $1, 2, \ldots, H-1$ as we did before. This results in $2^H - 2 - 2^{H-1} = 2^{H-1} - 2$ new leaves.

By the same argument we do not compute the subtree on the $H-2$-th level during superrounds $2^{H-1} + 2^{H-2} + 1, \ldots, 2^H$. Therefore we can attach two more leaves to each of the last 2^{H-1} extra leaves. This accounts for another $2^{H-1}2^1 - 2 - 2^{H-1}$ nodes. In the same way an arbirtary number m of levels can be added. Repeating the same for levels $H-3, H-4, \ldots, 1, \ldots, m-H$ we see, that we can add in total $(2^{H-2}2^1 - 2) + (2^{H-3}2^2 - 2) + \ldots + (2^{H-i-1}2^i - 2) + \ldots = m2^{H-1} - 2m$ leaves. And the last expression equals to $nm/2 - 2m$. However this method puts additional memory requirements, because the rightmost path in the skew Merkle tree has to be stored. Therefore m additional memory units must be stored.

For the general case $h \geq 1$ our method is almost the same as described above.

Namely, we add levels $0, -1, -2, \ldots$ to the Merkle tree. Level 0 can be added after $2^H - 2^{(L-1)h} + 2^h - 1$ superrounds level -1 can be added after $2^H - 2^{(L-2)h} + 2^h - 1$ superrounds and level $-i$ can be added after

$2^H - 2^{(L-i)h} + 2^h - 1$ superrounds.

Since subtrees appended at levels $0, -1, \ldots$ have 2^h leaves, superrounds $2^H - 2^{(L-1)h} + 2^h, \ldots, 2^H - 2^{(L-2)h} + 2^h - 1$ consist of 2^H rounds, superrounds $2^H - 2^{(L-2)h} + 2^h, \ldots, 2^H - 2^{(L-3)h} + 2^h - 1$ consist of 2^{2h} rounds and so on. Thus the total number of rounds is $\sum_{i=1}^{m/h-1}(2^{(L-i-1)h}2^{ih} - 2^h + 1) = L2^{(L-1)h} - L2^h + L$. Therefore the total number of extra nodes that could be appended to a Merkle tree is $(nm/h2^h) - 2^h m/h + m/h$

We sum up the above argument in a theorem:

Theorem 1 *There exists an efficient skew Merkle tree traversal algorithm that works in $O(\log n)$ time and $O(\log n) + m$ space. This algorithm can be used to authenticate up to $n + mn - m$ items. There exists an efficient skew Merkle tree traversal algorithm that works in $O(\log n/h)$ time and $O((2^h/h)\log n) + m$ space. This algorithm can be used to authenticate $n + nm - (m/h)2^h + (m/h)$ items.*

An interesting question remains on the optimal trade-off for traversing *hash chains* (c.f.,e.g., [CJ02], [J02]) . Unlike the Merkle tree, in hash chains all the values result from applying the hash function to the same initial seed value. That is, a hash chain is a sequence of values $h_0, h_1, \ldots, h_n$ so that $h_i = H(h_{i-1})$, where H is a hash function. The problem of a hash sequence traversal, i.e. of generating the hash values in reverse order $h_{n-1}, h_{n-2}, \ldots, h_0$ is considered by Coppersmith and Jakobsson [CJ02] and Jakobsson [J02]. In [CJ02] it was shown that hash chains can be traversed with $O(\log n)$ hash function applications per value and $O(\log n)$ memory cells.

We leave it as an open question whether our approach can be also applied for the general *hash chains*.

Acknowledgements

We thank Markus Jakobsson and Michael Szydlo for helpful discussions.

References

[BKN04] P. Berman, M. Karpinski, Y. Nekrich, *Optimal Trade-Off for Merkle Tree Traversal* , Technical Report TR-85255, University of Bonn, Available at `ftp://theory.cs.uni-bonn.de/pub/reports/cs-reports/2004/`

[CJ02] D. Coppersmith, M. Jakobsson, "Almost Optimal Hash Sequence Traversal", Financial Cryptography, 2002, 102-119

[J02] M. Jakobsson, *Fractal Hash Sequence Representation and Traversal*, Proc. of the ISIT, 2002 p.437; Full version available at `http://citeseer.ist.psu.edu/jakobsson02fractal.html`

[JLMS03] M. Jakobsson, T. Leighton, S. Micali and M. Szydlo, *Fractal Merkle Tree Representation and Traversal*, RSA Cryptographers Track, RSA Security Conference, 2003.

[S03] M. Szydlo, *Merkle Tree Traversal in Log Space and Time*, to appear in Eurocrypt 2004 Available at `http://www.szydlo.com/logspacetime.ps.gz`