On Merkle Hash Trees

Johannes Grabmeier and Larry Lambe

January 3, 2018

1 Merkle Hash Trees

Let

$$h : \Omega^* \to \Omega^l$$

be a cryptographic hash function for an alphabet $\{0, 1\} \subseteq \Omega$ and a fixed natural number $0 < l \in \mathbb{N}$. $(\Omega^*, +, \epsilon)$ is the semigroup of words of Ω with concatenation $+$ and zero element ϵ. The results are often called hash values or message digest or simply digest.

Cryptographic hash functions share the following properties:

- *Pre-image resistance*: It is difficult to find a pre-image. (One-way-function)
- *Second pre-image resistance*: Given an $m_1 \in \Omega^*$ it is difficult to find a $m_2 \neq m_1 \in \Omega^*$ such that $h(m_1) = h(m_2)$.
- *Collision resistance*: It is difficult to find $m_1 \neq m_2 \in \Omega^*$ such that $h(m_1) = h(m_2)$.

Furthermore, we have a finite sequence

$$d := (d_0, \ldots, d_{n-1}) = (d_i)_{0 \leq i < n} \in \Omega^{*n}$$

of messages or documents for $0 \leq n \in \mathbb{N}$.

Definition 1. *A Merkle hash tree $t_h(d)$ for h and d is defined recursively to have its nodes labelled by*

- $M(d) := h(\epsilon)$, *if d is the empty list for $n = 0$ and the empty word $\epsilon \in \Omega^*$ (empty list),*
- $M((d_0)) := h(d_0)$ *for the case of $d = (d_0)$ being a one-element list, i.e. $n = 1$ (leaf) and finally*
- *for $n > 1$ we define $k \in \mathbb{N}$ to be the largest power of 2 smaller than n, i.e. $k < n \leq 2k$ and we set*

$$M((d)) := h(M((d_i)_{0 \leq i < k} + M((d_i)_{k \leq i < n})$$

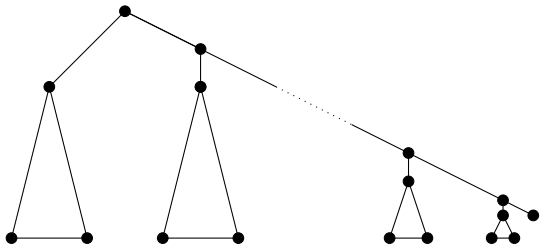
.

Note, that it is a binary tree.

Note, that the authors of [2] use SHA-256 for h an require additional concatenations of 00 resp. 01 from the left $M((d_0)) := h(00 + d_0)$ and $M((d)) := h(01 + M((d_i)_{0\leq i<k} + M((d_i)_{k\leq i<n})$. The purpose for that is not yet clear. They claim, that adding 00 from left to a leaf node and 01 for the other nodes is required to give second preimage resistance.

All the nodes of the Merkle binary tree are labelled by hashes. The tree has n leaves labelled by the hashes $(h(d_i))_{0\leq i<n}$. Merkle trees do not have to have the same depth for their leaves, i.e. distances from the root. But the number n of leaves determines the shape of the tree. If $n = 2^\kappa$ is a power of 2, then we have the perfect binary tree of depth κ. Otherwise, we assume that $k = 2^\kappa < n \leq 2^{\kappa+1}$, then the tree shape is recursively determined by the root node with the perfect binary tree of depth κ as left child and the binary tree with the shape of the binary tree with $n - k$ leaves as right child. This behaviour is iterated. The right subtree again has a left child being a perfect binary tree for a 2-power of leaves and a tree as right child for the remaining leaves. The structure of the left perfect binary tree is given by the binary expansion of $n = n_\kappa n_\kappa \dots n_1 n_0$.

Lemma 1. *The structure of Merkle hash trees is given by as many perfect subtrees as there are 1's in the binary expansion of the number n of leaves.*



1.1 Paths in Merkle Trees

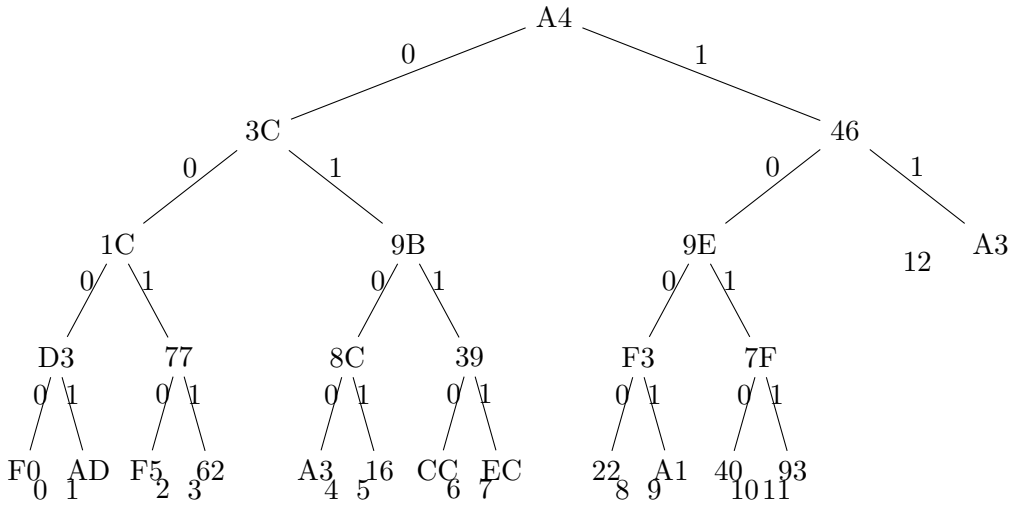
Labelling edges to left children with 0 and those to right children with 1 the path from the root to the i-th leave $0 \leq i < n$ is related to the binary expansion of i.

In other situations the binary path words have to be modified, e.g. if $n = 10$, the paths of leaf 8 and 9 in the right parts are 1000 and 1001 in full depth 4. The shortening of the two middle edges labelled 0 gives as the real paths 10 and 11 in depth 2. Note, as we also perform this operations, the shortened paths of the last leaf always consists of 1's only.

Lemma 2. *Let t a Merkle tree for the leaves $d = (d_i)_{0\leq i<n}$. The path from the root of t to the i-th leaf d_i, $0 \leq i < n$ is related to the binary expansion of i. We define the path $p(i)$ by setting 0, when it moves to the left child and 1 when it moves to the right child – from left to right. We have $p(i) \in \{0, 1\}^*$ of length equal or smaller to the maximal depth κ of the tree, given by $2^\kappa < n \leq 2^{\kappa+1}$.*

$p(i)$ can be constructed from the binary expansion of i. First fill with leading 0's to have the length κ. Then drop all those 0's at positions j, where the corresponding 2-power 2^{n_j} does not occur in the binary expansion of n.

Example 1. $n := 5000 = 1001110001000_{10} = 2^{12} + 2^9 + 2^8 + 2^7 + 2^3 = 4096 + 512 + 256 + 128 + 8$, $m - 1 := 4224 = 1000010000000_{10} = 2^{12} + 2^7 = 4096 + 128$. *Note, that the perfect subtree nodes of sizes 4096, 512, 256, 128 and 8 have paths 0, 1, 11, 111, 1111, and 11111, and the path to leaf number 4225 – counting starts with 1 – is 10010000000 as the 2 0's corresponding to 2048 and 1024 have to be dropped.*



1.2 Notation in Merkle Trees

For a node nd in a Merkle hash tree we denote by $nd.h$ its hash, by $nd.l$ its left child, by $nd.r$ its right child and by $nd.p$ its parent node. Note, that for a leaf node we have $nd.l = nd.r = ()$, the empty tree as well for the root node $nd.p = ()$.

1.3 Extension of Merkle Trees

Let t be a Merkle tree with n leaves and a further document d_{n+1} with $r := h(d_{n+1})$. If t is a perfect binary tree, then we can compute easily the new t by $t := (t, h(t.h+r), ((), r, ()))$. Otherwise, t is not perfect, then the left child must be a perfect binary tree with $k = 2^\kappa$ leaves such that $k < n \leq 2k$ and the right child can not be perfect. Following the sequence of right children as long as possible we get the leaf node for the last, i.e. n-th leaf. This is a perfect binary tree with $1 = 2^0$ leaves. Going back to parent nodes as long as this is the case, this tour has to stop, as the Merkle tree is not perfect. The resulting s is the smallest super node, which is not perfect, hence we can substitute it by the tree $p' := (s.l, h(s.l.h + r), ((), r, ()))$. From here on we move on to the root and substitute the next parent $s.p$ by $(s.p.l, h(s.p.l.h + p'.h), p')$.

2 Audit Proofs for Leaves

Definition 2. *For $n > 1$ let $k := 2^\kappa$ be the largest power of 2 such that $k < n$. The* Merkle audit path $p(m, d)$ *for the $m + 1$-st element d_m in the list $d = (d_i)_{0 \leq i < n}$ with*

$0 \leq m < n$ *is recursively defined to be*

- $p(0,(d_0)) := []$ *for a single leaf in a tree with a one-element input list;*
- $p(m,d) := p(m,(d_i)_{0\leq i<k}) : M((d_i)_{k\leq i<n})$ *for* $m < k$ *and*
- $p(m,d) := p(m-k,(d_i)_{k\leq i<n}) : M((d_i)_{0\leq i<k})$.

Here : *denotes the extension of a list by a further value.*

Note, that the audit path consists of the list of missing nodes required to compute the nodes leading from a leaf to the root of the tree. This can be used as a proof that the leaf exists in the tree by the properties of the cryptographic hash function h:

Theorem 1. *From* $p(m,d)$ *and* d_m *for* $d = (d_i)_{0\leq i<n}$ *and* $0 \leq m < n$ *the hash* $M(d)$ *can be computed by the following algorithm. Define* $k, \kappa \in \mathbb{N}$ *such that* $k := 2^\kappa < n \leq 2k = 2^{\kappa+1}$. *Let* $m' := (m_\kappa, m_{\kappa-1}, \ldots m_1, m_0)_2$ *be the binary expansion of* m, *prolonged by leading 0's, if the length is smaller than the length of the proof, by dropping final 0's, if the length is larger.*

- $y := d_m$.
- *for* p *in* $p(m,d)$ *and parallel for* b *in reverse* m' *repeat*
  - *if* $b = 1$ *then* $y := h(p,y)$
  - *else* $y := h(y,p)$
- $y = M(d)$.

Proof. If $n = 1$ then $m = 0$ and clearly $M((d_0)) = h(d_0)$. $(d)_{0\leq i<n})$ and the hash $M((d_i)_{k\leq i<n})$ can be computed. But then the root hash of $M(d)$ is the hash of the concatenation of $M((d_i)_{0\leq i<k})$ and $M((d_i)_{k\leq i<n})$ for the case $m < k$ and that of $M((d_i)_{k\leq i<n})$ and $M((d_i)_{0\leq i<k})$ by the construction of the Merkle hash tree $M(d)$. □

3 Merkle Consistency Proof

If we assume and want to guarantee that a Merkle hash tree can modified only by appending new leaves or further Merkle hash trees, it is of utmost importance, that the owner of a document d_{m-1} who had received the root hash $M((d_i)_{0\leq i<m})$ of the Merkle hash tree after appending the leaf $h(d_{m-1})$ not only gets a confirmative yes that the subtree for the documents $d' := (d_i)_{0\leq i<m}$ is still unchanged and a part of the tree after further leaf extensions by $(d_i)_{m\leq i<n})$, but gets a proof of that. If this tree still exists as a subtree node in the new tree for $d := (d_i)_{0\leq i<n}$ the hash of the subtree for d' has changed. Hence, the owner should be provided with a list of nodes which enables him to compute $M((d_i)_{0\leq i<n})$ as well as to compute $M((d_i)_{0\leq i<m})$, such that he compare the latter hash with originally given one for equality.

Hence, the task is the following

- Given $1 \leq i \leq m \leq n$ and $d := (d_0, d_1, \ldots, d_{n-1})$ and the Merkle tree $t_h(d)$,
- compute a list of hashes $c(m, d)$ such that
- someone who knows $m, n, M(d')$ and $M(d)$
- can compute both $M(d')$ and $M(d)$ and from $c(m, d)$ only using h.

Definition 3. *For $1 \leq m \leq n$, $d := (d_i)_{0 \leq i < n}$, and $d' := (d_i)_{0 \leq i < m}$ let $k := 2^\kappa$ be the largest power of 2 such that $k < n$. The Merkle consistency proof $c(m, d)$ for the Merkle hash tree for d' with root hash $M(d'$ to be a subtree of the Merkle hash tree for d with root hash $M(d)$ is*

$$c(m, d) := c'(m, d, 1),$$

where c' is recursively defined to be

- *$c'(m, d', 1) := []$, if $m = n$ and hence $d = d'$, such that the subtree coincides with the tree;*
- *$c'(m, d', 0) := M(d)$, if $m = n$ and hence $d = d'$, such that the subtree coincides with the tree;*
- *$c'(m, d, b) := c'(m, (d_i)_{0 \leq i < k}, b) : M((d_i)_{k \leq i < n})$ for $m \leq k$ and $b \in \{0, 1\}$, note that the right subtree entries $(d_i)_{m \leq i < n}$ only exist in the current tree;*
- *$c'(m, d, b) := c'(m - k, (d_i)_{k \leq i < n}, 0) : M((d_i)_{0 \leq i < k})$ for $m > k$ and $b \in \{0, 1\}$, note that the left subtrees for $(d_i)_{0 \leq i < k}$ have to be identical in both trees. The number of nodes in the proof $c(m, d)$ is bounded by $\kappa + 1$.*

Here : denotes the extension of a list by a further value.

Theorem 2. *The list $c := c(m, d)$ can be used to computed $r' := M(d', m)$ and $r := M(d, m)$ by the following algorithm.*

- *If $m = n$ set lb to be the empty list. Compute the binary expansion of $m - 1$, extend to length κ by leading 0's, reserverse the order of the bits and drop the sequence of leading 1's. The result is called lb*
- *If m is a power of 2 or $m = n$ set $hx := r; hx' := r'$*
- *else set $hx := firstc; hx' := firstc, c := restc$.*
- *For u in c and for b in lb in parallel repeat*
  - *if $b = 1$ then $hx := h(u + hx); hx' := h(u + hx')$*
  - *else $hx := h(hx + u$*
- *$(hx' = r'$ and $(hx = r)$.*

Proof. • Let $m = n$, then we have nothing to prove, as the two trees coincide.

- Now we can assume that $m < n$. As usual let $k = 2^\kappa < n \leq 2k = 2^{\kappa+1}$. Two cases, namely $m \leq k$ and $k < m$.
- If $m \leq k$ then $h = h(r + last(c(m, d)))$ else
- case $m > k$ then $h(last(c(m, d)) + r)$.
- $c'(m, d, b) := c'(m - k, (d_i)_{k \leq i < n}, 0) : M((d_i)_{0 \leq i < k})$ for $m > k$ and

□

3.1 Example of Consistency Proofs

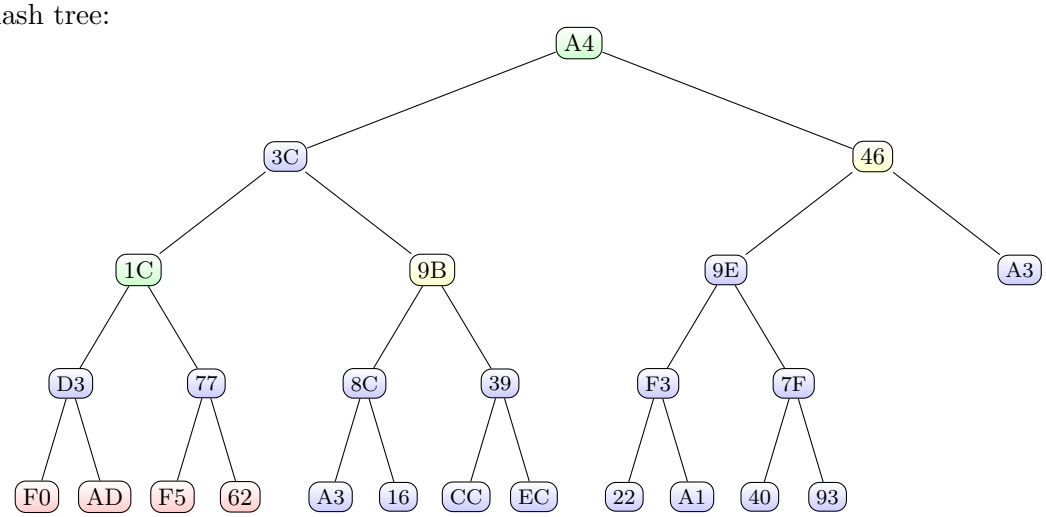
We define a list of 13 documents as the binary expansion of

$$(702934750923487502394785029348 75^i)_{0 \leq i < 13}.$$

After using the 512-bit padding of SHA-1 its hash codes by SHA-1 are the following 7 40-digits hexadecimal numbers,

```
F07F977DE683FCDB71347AF1E63FC039EF0F0074 AD7D0CFE4CF22065EEC00AE5F9B3742A3F40C387
F5C3F4F492F985D1D035F053FF5E725B3C647D2C 6207E607E533B76F87EE21C14F96619C4D0A592D
A351D63CD815906118C03FE4415B05D984AC4F91 162CEDDB7F3EDEF374CBDD8EBD674375E0202B7A
CC3B66372ED5AAD9C9060ED9E120BDA3EEB60086 162CEDDB7F3EDEF374CBDD8EBD674375E0202B7A
CC3B66372ED5AAD9C9060ED9E120BDA3EEB60086 EC68CC77D54C6311A45E2865B00707D364A20840
22D0C51B72B100CB1D89962F32603FEC2C48379E A1D3A54273CF89D9461B467B3878DE36AFEB309D
40D14D4ADC9C2F76D74DBC489A643433A9D7C022 9355EE62AB5EE1F70E7182763B288BF7943AEAA7
                    A3DE7E425CAB20F9A53F23B443A94CFA92DC1237
```

which we abbreviate by their two leftmost 'digits' in the following picture of their Merkle hash tree:



3.1.1 Proof for the First 4 Leaves

The binary expansion of $m - 1 = 4 - 1 = 3$ of length $4 = \lceil \log_2(16) \rceil$ is 0011_2, reverse order is 1100, dropping the leading 1's yields the path 00 from node 1C to the root A4. The consistency proof for the Merkle hash tree for the $m = 4$ leaves F0, AD, F5 and 62

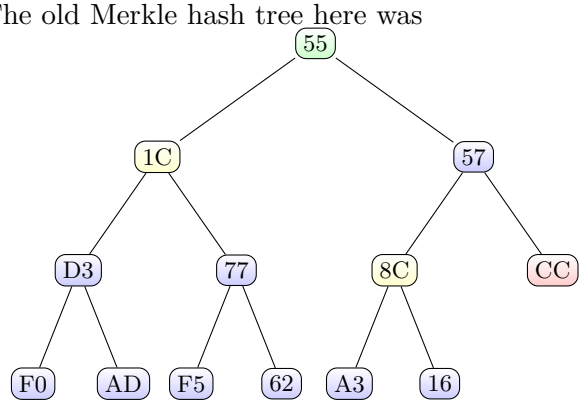
is the hash sequence (9B, 46). The owner of the document with 62, who knows the root hash 1C of this earlier Merkle tree – which happens to be identical as the hash of the corresponding node within the actual Merkle tree, can use this information to compute the current hash root as follows:

$$hx := 1C, hx := h(1C, 9B) = 3C; hx := h(3C, 46) = A4.$$

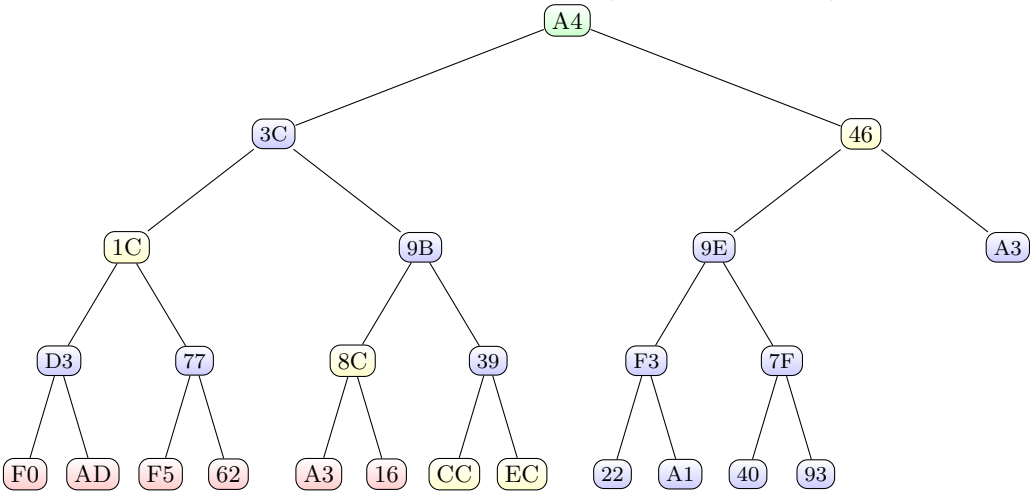
Note, that in both hash computation, hx has to be the first arguments as the path is 00.

3.1.2 Proof for the First 7 Leaves

The old Merkle hash tree here was



Note, that now the old root hash 55 differs from the corresponding node hash 3C in the current tree. The binary expansion of $m - 1 = 7 - 1 = 6$ of length $4 = \lceil \log_2(16) \rceil$ is 0110_2, reverse order is 0110, no leading 1's to drop, hence this is the path from leaf CC to the root A4. The consistency proof for the Merkle hash tree for the $m = 4$ leaves F0, AD, F5, 62, A3, 16 and CC is the hash sequence (CC,EC,8C,1C,46).



The owner of the document with CC, who knows the root hash 55 of this earlier Merkle tree can use this information to compute the current hash root as follows. As $m = 7$ neither is a power of 2 nor is equal to $n = 13$, this time he has to start with the

first entry in the proof list:

$$hx := CC, hx := h(CC, EC) = 39; hx := h(8C, 39) = 9B, hx := h(1C, 9B) = 3C, hx := h(3C, 46).$$

Note, that in the 4 hash computation, hx has to be first (0) resp. second argument (1) according to the path 0110.

Moreover, he also can verify, that the first 7 leaves still unchanged are in the current tree, as the computation

$$hx' := CC, hx' := h(8C, CC) = 57, hx' := h(1C, 57) = 55,$$

and he could recompute the old root hash. But note, computations only are allowed, if the path value is 1.

References

[1] Marc Clifton: *Understanding Merkle Trees – Why use them, who uses them, and how to use them*, `https://crypto.stackexchange.com/questions/1011/why-doesnt-preimage-resistance-imply-the-second-preimage-resistance?rq=1`

[2] B. Laurie, A. Langley, E. Kasper: *Certifate Transparency*, `http://www.rfc-editor.org/info/rfc6962`