Public Key Certificates using a Merkle Tree
Summary of an Idea
peter-thoemmes.org research

© Peter Thoemmes
Weinbergstrasse 3a
D-54441 Ockfen, Germany

December 9, 2011

Abstract

This paper is a summary of the idea having public key certificates using a Merkle Tree. Ralph C. Merkle invented the Merkle Puzzles in 1974 as basic idea to the later (1976) developed Diffe-Hellmann Key Agreement Method (Ralph C. Merkle, Whitfield Diffe and Martin E. Hellman). Later in 1979 Merkle invented the Merkle Trees, and this is a summary about how to use a Merkle Tree to create and verify public key certificates. This paper is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY, without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Contents

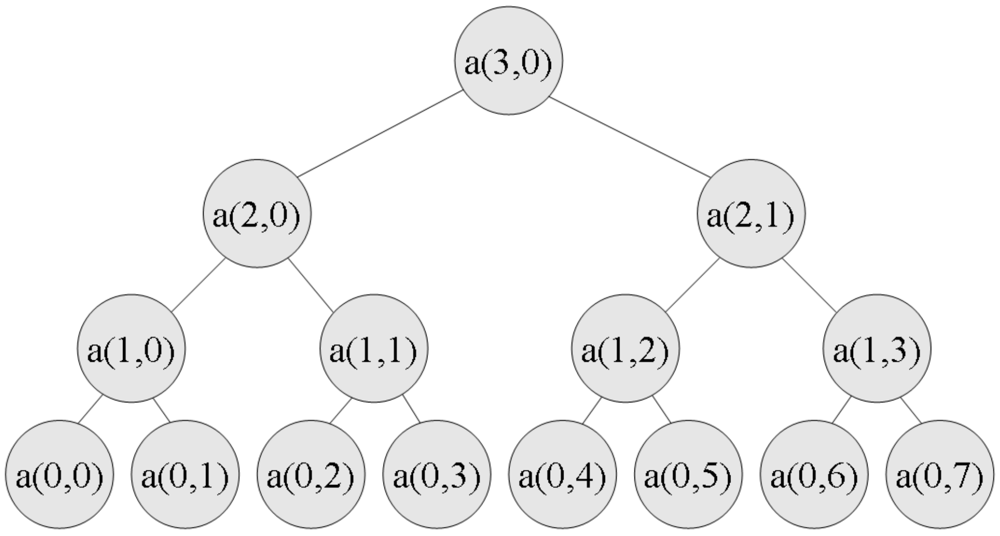
1 Building a Merkle Tree for 2^n Certificates

Having a look at Figure 1, you see a Merkle Tree having 8 leaves. You see that the number of leaves is always a power of 2. If you count the number of levels n under the tree's root, which is 3 in our tree, then you will find following equation:

$N = 2^n$ with $n \in \mathbb{N}^+$

N: Number of leaves of the tree.
n: Number of tree levels under the tree's root.

Figure 1: Merkle Tree for N = 8



So now we generate N asymmetric key pairs and assign those to the tree's leaves by hashing each pair's public key part concatenated (||) with the owner's name (FQDN, Fully Qualified Domain Name):

Level 0 (leaves):

$name_0 : priv_0, pub_0 \rightarrow a(0,0) := hash(pub_0||name_0)$
$name_1 : priv_1, pub_1 \rightarrow a(0,1) := hash(pub_1||name_1)$
...
$name_7 : priv_7, pub_7 \rightarrow a(0,7) := hash(pub_7||name_7)$

Then we create the next levels of nodes. By concatenating the values of the 2 nodes underneath and then hashing them, we get each node's value:

Level 1:

$$a(1,0) := hash(a(0,0)||a(0,1))$$
$$a(1,1) := hash(a(0,2)||a(0,3))$$
$$a(1,2) := hash(a(0,4)||a(0,5))$$
$$a(1,3) := hash(a(0,6)||a(0,7))$$

Level 2:

$$a(2,0) := hash(a(1,0)||a(1,1))$$
$$a(2,1) := hash(a(1,2)||a(1,3))$$

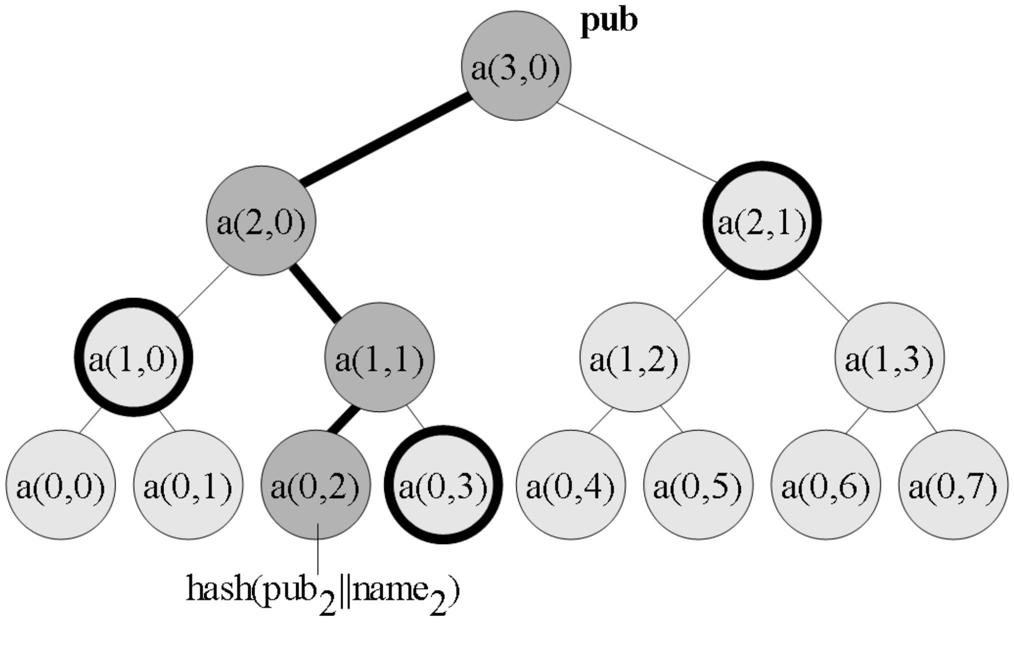
Level 3 (root):

$$a(3,0) := hash(a(2,0)||a(2,1))$$

2 Creating a Public Key Certificate

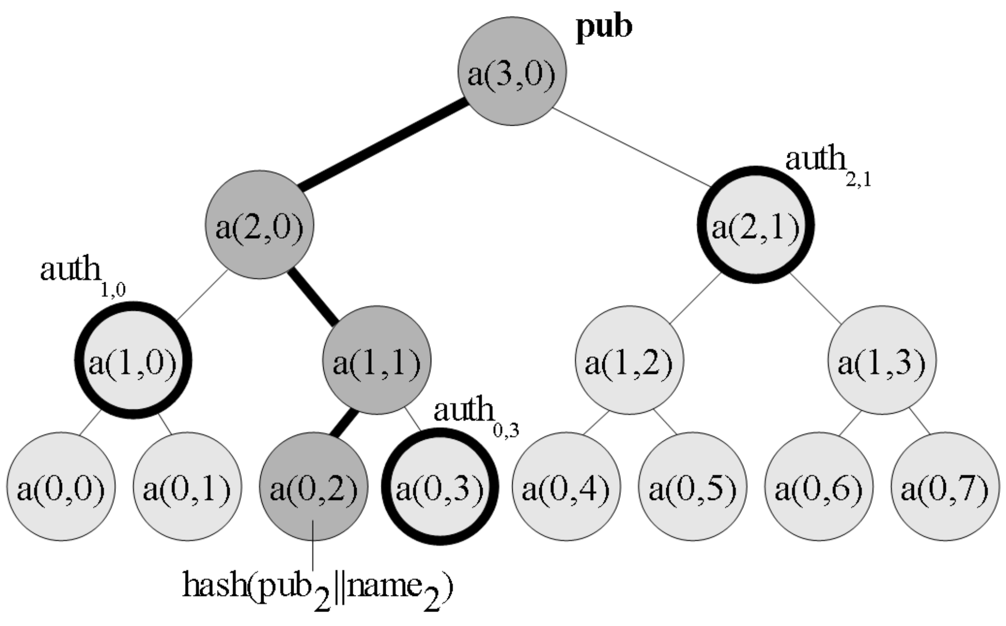
To sign a certificate, consisting of a public key and a name (FQDN), both belonging to one leave of a Merkle Tree, we define two things first: the Merkle Tree's overall public key **pub** and the **path** to the hash of our the certificate (which is the hash over the concatenation of our public key and our name). Please see Figure 2, which shows the path for signing the certificate made of pub_2 and $name_2$:

Figure 2: Path to pub_2 and $name_2$ in the Merkle Tree



Due to the symmetry, each node along the path has a brother. Merkle's authentication makes use of those brothers, as you see when looking at $auth_{0,3}$, $auth_{1,0}$ and $auth_{2,1}$ in Figure 3.

Figure 3: Merkle Tree Authentication



So there will always be n brothers along a path of a Merkle Tree and those will be used to sign our certificate by simply concatenating them to the owner's public key and name as shown here for our example:

$cert_2 := pub_2||name_2||auth_{0,3}||auth_{1,0}||auth_{2,1}$

3 Validating a Public Key Certificate

To validate a certificate consisting of a public key, a name (FQDN) and a Merkle authentication, the overall public key **pub** of the related Merkle Tree **needs to be deployed safely to the receiver first**. This is somehow equal to the installation of a root CA's public signature decryption key, when talking about SSL public key certificates in an SSL PKI (Public Key Infrastructure). Without having this starting point safely deployed once, things don't work here. The important thing with **pub** is, that it needs to be authentic. It is not important to hide it, but it is important to make it resistant against manipulation. So if a receiver, owning the valid overall public key **pub** of the Merkle Tree, receives a public key certificate, he validates it by calculating the value of **pub** from that certificate. If this calculation is equal to the installed **pub**, the certificate is valid, and so the public key inside can be seen as belonging to the name (FQDN) inside:

$cert_2 := pub_2||name_2||auth_{0,3}||auth_{1,0}||auth_{2,1}$

$$a(2,0) = hash(pub_2||name_2)$$
$$a(1,1) = hash(a(0,2)||auth_{0,3})$$
$$a(2,0) = hash(a(1,1)||auth_{1,0})$$
$$a(3,0) = hash(a(2,0)||auth_{2,1})$$

$a(3,0) \overset{!}{=} pub \rightarrow$ VALID $\rightarrow pub_2$ and $name_2$ are valid and belong together!