**Chapter 2**

**LITERATURE SURVEY**

This chapter will introduce the basic mathematical notions in cryptography, which are of great significance to this thesis. It will brief about digital signatures and one - time digital signatures, about hash and one - way functions, and of course, about the original Merkle digital signature, which is of crucial importance for Merkle tree traversal techniques.

**2.1 Hash and one - way functions**

**2.1.1 One - way functions**

A one - way function is a mathematical function that is significantly easier to compute in one direction (the forward direction) than in the opposite direction (the inverse direction). It might be possible, for example, to compute the function in the forward direction in seconds but to compute its inverse could take months or years, if at all possible.

**2.1.2 Hash functions**

The term hash apparently comes by way of analogy with its standard meaning in the physical world, to "chop and mix". The first use of the concept was in a memo from 1953, some ten years later the term hash came into use. Mathematically, a hash function (or hash algorithm) is a method of turning data into a number suitable to be handled by a computer. It provides a small digital “fingerprint” from any kind of data. The function substitutes or transposes the data to create that “fingerprint”, usually called hash value. This value is represented as a short string of random – looking letters and numbers (for example binary data written in hexadecimal notation).

**2.2 Digital signatures**

**2.2.1 The definition**

This chapter considers techniques designed to provide the digital counterpart to a handwritten signature. A digital signature of a message is a number dependent on some secret known only to the signer, and, additionally, on the content of the message being signed. Signatures must be verifiable; if a dispute arises whether a party signed a document (caused by either a lying signer, trying to repudiate the signature that the party created, or a fraudulent claimant), an unbiased third party should be able to resolve the matter equitably, without requiring access to the signer’s secret information

(Private Key).

* A digital signature is a data string, which associates a message (in digital form) with some originating entity.
* A digital signature generation algorithm is a method for producing a digital signature.
* A digital signature verification algorithm is a method for verifying that a digital signature is authentic (i.e., was indeed created by the specified entity).
* A digital signature scheme (or mechanism) consists of signature generation algorithm and an associated verification algorithm.
* A digital signature signing process (or procedure) consists of a digital signature generation algorithm, along with a method for formatting data into messages, which can be signed.
* A digital signature verification process (or procedure) consists of a verification algorithm, along with a method for recovering data from the message.

**2.2.2 One - time digital signatures**

One - time digital signature schemes are digital signature mechanisms, which can be used to sign, at most, one message; otherwise, signatures can be forged [MOV96]. A new public key is required for each message that is signed. The public information necessary to verify one - time signatures is often referred to as validation parameters. When one – time signatures are combined with techniques for authenticating validation parameters, multiple signatures are possible. Most, but not all, one – time digital signature schemes have the advantage that signature generation and verification are very efficient. One - time digital signature schemes are useful in applications such as chip cards, where low computational complexity is required.

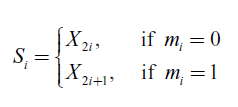
**2.3 The original Merkle digital signature**

**2.3.1 Introduction**

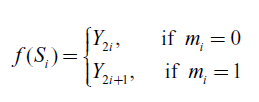
Merkle proposed a digital signature scheme that was based on both one - time signatures and hash functions, and that provides an infinite tree of one - time signatures. One - time signatures normally require the publishing of large amounts of data to authenticate many messages, since each signature can only be used once. Merkle's scheme solves the problem by implementing the signatures via a tree - like scheme. Every node has a k bit value and every interior node’s value is a hash function of the node values of its children. The public key that helps for authentication is placed as root of the tree. The one - time private keys are used for generation of the leaves and are authenticated with the help of the root. Although the number of messages that can be signed is limited by the size of the tree, the tree can be made arbitrarily large.

**2.3.2 The Lamport-Diffie one-time signature scheme**

Whitfield Diffie and Martin E. Hellman presented a new digital signature, based on hash functions. Such function *Y* = *f* (*X*) is selected. Each user U chooses 2*n* random values *X0 X1… X2n-1* and computes *Y0, Y1, Y2n-1* by *Yi* = *f (Xi)*. Then U publishes the vector *Y = (Y0 Y1 … Y2n-1)* in a public file under his name (i.e., in a newspaper or in a public file maintained by a trusted center). He can publish as many vectors as the number of signatures he is expected to sign. Now we come to signature generation [EB05]. Alice wants to sign an *n* - bit message *M* to Bob (*M = m0 m1… m*n-1). She then chooses one of his unused vectors from the public file and sends it to Bob. Bob verifies the existence of the vector in the public file. After that Alice and Bob mark the vector as used in the specific file. Alice computes the signature *S = S0 S1 S*n-1 by



and sends it to Bob. To verify the signature, Bob computes for all *i*- s

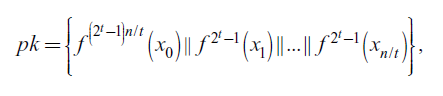


Talking about the security of the signature scheme we claim that if Bob can invert the hash function *f*, he can then forge Alice’s signature. Even if he is given a signature of some message using some vector, he still needs to invert the hash function *f* in order to forge a different message using the same vector. To make this easier to understand we give the following example:

**Example:** Let *X* =01001101 and *f*(*X*) be a function that just changes 0-s with 1-s and vice versa. Then *Y* =10110010. Imagine that Alice wants to sign two messages *M1* =1010 and *M2* =1101 with this very same private key *X*. Then, according to the definition of the scheme, Alice computes the corresponding signatures *S1* =1010 and *S2* =1011, and sends them to Bob. Knowing *M1*, *M2*, *S1* and *S2* it is extremely easy for Bob to forge a new message *M3* =1111 with its corresponding signature *S3* =1011, just combining the previous two.

**2.3.3 Winternitz improvement**

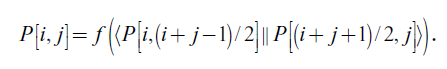
One generalization of the Lamport – Diffie scheme attributed by Merkle to Winternitz in is to apply the hash function *f* to the secret key iteratively a fixed number of times, resulting in the public key. Briefly the scheme works as follows- Suppose we wish to sign a *n*- bit message *M*. First the message is split into *n* / *t* blocks of size *t* bits. Let these parts be *M1, M2,……., Mn/t* . The secret key is sk ={x0, x1,….,xn/t}where xiis a *l*- bit value. The public key is



where *f 2*(*x*)= *f* (*f* (*x*)) is applying *f* iteratively.

**2.3.4 Merkle trees**

The greatest disadvantage of Lamport - Diffie scheme is the size of the public key. All the verifiers need an authenticated copy of this public key in order to verify the validity of any signature. In [MER79] Merkle proposed the use of binary trees to authenticate a large number of public keys with a single value, namely the root of the tree. That is how the definition of a Merkle tree comes into use. It is a complete binary tree with a *k*- bit value associated to each node such that the interior node value is a hash function of the node values of its children:



*P* is the assignment, which maps the set of nodes to the set of their strings of length *k*. In other words, for any interior node *nparent* and its two child nodes *nleft* and *nright*, the assignment *P* is required to satisfy:



The N values that need to be authenticated are placed at the N leaves of the tree. We may choose the leaf value arbitrarily, but usually it is a cryptographic hash function of the values that need to be authenticated. In this case these values are called leaf - preimages. A leaf can be verified with respect to a publicly known root value and its authentication path. We assume that the public keys of the Lamport - Diffie one- time signature scheme are stored at the leaf - preimages of the tree (one public key per leaf - preimage).

**2.3.5 Authentication paths**

Let us set *Authn* to be the value of the sibling of the node of height *h* on the path from the leaf to the root. Every leaf has height 0 and the hash function of two leaves has height 1, etc. In this manner the root has height *H* if the tree has 2H= *N* leaves. Then for every leaf the authentication path is the set { *Authn* | 0 ≤ *h* <*H* . So a leaf can be authenticated as follows: First we apply our hash function *f* to the leaf and its sibling *Auth0*, then we apply *f* to the result and *Auth1* , and so on all the way up to the root. If the calculated root value is the same as the published root value, then the leaf value is accepted as authentic. This operation requires (log2 *N*)invocations of the hash function *f*.

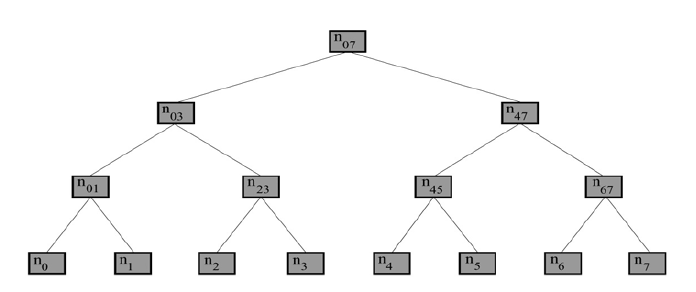


Figure 2.1: Merkle tree with 8 leaves.