Sara Rosander
sararosa@kth.se

Kryptolab – One-Way Hash Function och MAC

2.1 Generering av Message Digest och MAC

Uppgift 1

I denna uppgift så undersöktes de olika hash-algoritmerna för One Way Hash, dvs SHA256, SHA1 och MD5. De olika algoritmerna applicerades på en klartext i form av min KTH-mail: ”sararosa@kth.se”. När det olika algoritmerna applicerades på klartexten så genererades olika hashvärden för samma klartext, med olika längd. Under utförandet uppmärksammades det att SHA256 genererade det hashvärdet med mest antal bitar. Efter en kontroll i dokumentationen för det olika algoritmerna så upptäcktes det att SHA256 ska generera en fast storlek på hashvärdet som är 256 bitar, SHA1 ska generera ett fast värde på 160 bitar och MD5 ett fast värde på 128 bitar. Detta stämde överens med längderna på de olika värdena som genererades.

Uppgift 2

Se nedan det olika hashvärdena som beräknades när algoritmerna applicerades på klartexten:

MD5(kthmail)= 258c3c8e60cea86109676d5f06ac5ad1

SHA1(kthmail)= 5d7687f87854288de0fb1142169069ea5d7d4161

SHA256(kthmail)= 9f8bc443c17bab18572b08a8a0c85fb59a97581dd0fbed0e9589b015ebb3c7ea

2.2 Keyed Hash och HMAC

Uppgift 3

I uppgiften utfördes tester med hjälp av HMAC, dvs. jag genererade hashvärden med hjälp av en nyckel och hashalgoritmerna. Algoritmerna och nyckeln applicerades på den tidigare filen med min KTH-mail. Jag testade olika längder på nyckeln, jag startade med nyckeln ”abcdefg” och tog sedan kortare nycklar och även mycket längre nycklar med samma antal bitar som hashvärdet från respektive algoritm. Men oavsett vilken längd på nyckeln jag tog så genererades ett hashvärde med lika många bitar för alla algoritmerna. Om en nyckel skulle vara kortare än själva blockstorleken som hash-algoritmerna opererar på så appliceras padding för att få den önskade storleken på blocken, skulle däremot nyckeln vara längre än blockstorleken så kortas nyckeln ner med hjälp av hashing. Genom detta så produceras samma nyckellängd oavsett hur många bitar nyckeln består av. Så nyckeln ska kunna vara vilken längd som helst.

Uppgift 4

Se nedan för de olika hasvärdena som beräknades i HMAC-uppgiften:

HMAC-MD5(kthmail)= d9e2c3d4a969567691fa20a3515ea0aa

HMAC-SHA1(kthmail)= 2d49a57e9c0cb3d5e07d57e493fd1eb57b9f6307

HMAC-SHA256(kthmail)= 26e0acbadf95d8cae9d9e40c83c4edc26be4623d61fdb52a03cc58d1df40bc47

2.3 Slumpmässigheten i One-way Hash

Uppgift 5

I denna uppgift så testades slumpmässigheten i SHA256 och MD5. Den första biten i ”sararosa@kth.se” ändrades från 0 till 1. Skillnaden mellan de olika värdena mättes sedan med ett Java-program. Vi kan se att skillnaden blir ganska stor mellan hashvärdena om man endast ändrar en bit. I båda fallen så kan vi se att mer

än hälften av bitarna ändrades. Detta tyder på en bra slumpmässighet. Slumpmässighet är viktigt för en hashfunktion annars finns det en chans att man kan göra förutsägelser om inputen med avseende på outputen och i slutändan finns det då risk att algoritmen kan knäckas. Se nedan för resultatet för skillnaden mellan de olika strängarna:

SHA256
Ursprungligt värde: 9f8bc443c17bab18572b08a8a0c85fb59a97581dd0fbed0e9589b015ebb3c7ea
Nytt värde: d2e6545b72469d5b05fa1c4c6584a87fc9f88124a2bf7fdaeb6dea3ca5706d8c

Diff: 190bitar

-> 66 bitar är likadana

MD5
Ursprungligt värde: 258c3c8e60cea86109676d5f06ac5ad1
Nytt värde: b3de44d5b3b524127664576d38ab1927

Diff: 77bitar

-> 51 bitar är likadana.

2.4 Stark vs Svag Kollisionsresistans

Uppgift 6

I denna uppgift skulle det undersökas hur lång tid det tog att knäcka ”weak collision property” med hjälp av brute force. Kollisionsresistans är en egenskap som en kryptografiskt hashfunktion har. En hashfunktion är kollisionsbeständig om det är svårt att hitta två inputs som har samma hashvärde. Skillnaden mellan svag och stark kollisionsresistans är att med svag kollisionsresistans är vi ”bunden” till ett visst val av input, t.ex. i denna labb ska vi hitta en sträng med samma hashvärde som en viss utvald sträng. Medans i stark kollsionsresistans är man mer fri att välja vilka två inputs man ska ha.

För att undersöka den svaga kollisionsresistansen skapades ett eget program i Java. För att generera ett hashvärde används Javas Security bibliotek och metoden nedan skapades för att generera de olika hashvärdena för de olika strängarna. Delar av denna kod är tagen från exemplet som tillhandahölls i laborationen.

```
pulic byte[] makeDigest(String inputText){
    try {
        MessageDigest md = MessageDigest.getInstance(digestAlgorithm);
        inputBytes = inputText.getBytes(textEncoding);
        md.update(inputBytes);
        digest = md.digest();

    } catch (NoSuchAlgorithmException e) {
        System.out.println("Algorithm \"" + digestAlgorithm  + "\" is not available");
    } catch (Exception e) {
        System.out.println("Exception " + e);
    }

    return digest;
}
```

Själva brute-force-attacken genomfördes genom att lägga till värdet för en räknare som räknar upp varje gång ett nytt hashvärde räknas ut. Detta värde för räknaren omvandlades till bytes som sedan hash-algoritmen applicerades på. I min första implementation använde jag en slumpvis vald byte-array för att generera värden men insåg att det var svårt att jämföra hur många försök det tog för att knäcka varje sträng då slumpen flera gånger avgjorde hur många försök det tog att generera ett liknande hashvärde. Jag valde därför att använda denna räknare istället för en enklare jämförelse. För varje hashvärde som tas fram jämförs de första 3 byten med hashvärdet på den ursprungliga strängen. Se metoden för brute-force-attacken nedan:

Sara Rosander
sararosa@kth.se

```
public void bruteForce(byte[] digest) throws UnsupportedEncodingException {
    while(true) {
        counter++;
        String inputText = Long.valueOf(counter).toString();
        tryDigest = makeDigest(inputText);

        if (tryDigest[0] == digest[0] && tryDigest[1] == digest[1] && tryDigest[2] ==
digest[2]){
            System.out.println("It took " + counter + " number of times to find a matching
message digest.");
            System.out.print("The matching digest became: ");
            printDigest(tryDigest);
            return;
        }
```

Nedan följer resultaten för de olika strängarna:

”IV1013 security” – 28 872 840 försök
” Security is fun” – 10 015 372 försök
”Yes, indeed” – 9 777 773 försök
“Secure IV1013” – 23 746 118 försök
”No way” – 1 139 565 försök