Assignment 7

Exercise 1:

a:

1. A $\rightarrow$ S : A
2. S $\rightarrow$ A: n
3. A $\rightarrow$ S: X = h^{n-1}(pw)

Een aanval op dit protocol kan er zo uit zien:

1. A $\rightarrow$ B: A
2. B $\rightarrow$ S: A
3. S $\rightarrow$ B: n
4. B $\rightarrow$ A: n
   (B weet nu wat n is)
5. B $\rightarrow$ A: n-1
6. A $\rightarrow$ B: X = h^{n-2}(pw)
   (Nu heeft B de hash die A de volgende keer naar S gaat sturen, als B $\rightarrow$ A: n-2 of n-3 stuurt kan B achter alle volgende hashes komen)

b:

Nee, als Eve alle hashes die worden verstuurd heeft vastgelegd en er wordt opnieuw bij 10000 begonnen kan Eve uit die hashes die ze heeft verzameld aflezen welke hash de volgende keer nodig is om te kunnen authenticeren bij S.

c:

Wanneer er tussendoor een hash bekend wordt, kan de aanvaller alle volgende hashes berekenen.

d:

Dit is eigenlijk hetzelfde als bij c alleen nu kan de aanvaller terugrekenen in plaats van verder rekenen.

Exercise 2:

a:

We hebben K en c_1

1. Hash $c_1 || K$
2. XOR c_2 met $H(c_1 || K)$
   C_2 XOR $H(c_1 || K) = m_2$
3. Hash $K || m_2$
4. XOR c_1 met $H(K || m_2)$
   C_1 XOR $H(K || m_2) = m_1$
5. $m = m_1 || m_2$

b:

Collisian resistance, het moet lastig zijn om het bericht te kunnen aanpassen.

c: (verder in engels)

They can now recover c1 as well, since k{m} = c1||c2. Using c1, c2, t1, and t2, the attacker can compute the key if he knows how to dehash.

d:

Yes, it is still a problem. The attacker knows both c1 and c2. It doensn't matter if you use them in different tags or in one. The attacker can still know the key after dehashing.

3.

a) (Second) Preimage resistance. It must be infeasible for a hacker to find a file (that is different from the original file) that has the same hash as the original file.

b) The file Alice wants to download is 2 GB. That is 2.000.000.000 bytes. The file is split into pieces. One piece is 16 KB big. That is 16.000 bytes. This means the file is split into 2.000.000.000 / 16.000 = 125.000 pieces. Each piece needs one hash. One SHA-1 hash is 20 bytes big. This means the torrent file is at least 125.000 * 20 = 2.500.000 bytes big. That is 2,5 MB.

c) When the download is completed. Only then you have full possession of all the files, which is needed to calculate the root. As soon as you have every piece, you calculate the root of all the pieces/hashes together, and compare it to the root in your torrent file.

d) N0 2 , N5 0 , N3 1

e) In the binary hash tree in (d), there were 8 leaf nodes. The tree had 3 levels below the root (23=8). Given the hash from 1 leaf node, it needed 1 more hash from every level to calculate the root. In that case that were 3 hashes, since there were 3 levels below the root. If we have 1024 pieces (lead nodes), which is equal to 210, there are 10 levels below the root, and so we need 1 hash from every level (=10) together with a piece to calculate the root.