**EMCS2020: Advanced Topics in Computer Security**

Assignment: Encryption Revisited

Brian Russel Davis, [brian\_davis@brown.edu](mailto:brian_davis@brown.edu)

***Suppose an attacker subsequently finds OTP x8 and sends it to the server trying to be authenticated as the legitimate client. Explain the computation performed by the server upon receiving x8 and how the server determines that x8 is an invalid OTP, thus defeating the replay attack.***

Once the server has received and run the hashing function on X7 earlier items in the chain become invalid. The server does not save a copy of the passwords, instead, it uses the values that are sent by the client in a hashing function, i.e. :

**Check t = h(x7)**

**Check t = h(h(x7))**

**Set t = x7**

After t has been set to X7, if an attacker tried to send X8 that the second check above would fail. Basically, the server invalidates all of the items that are X+1 or greater, since hashing of the hash of an item is a one-way function.

***Referring to slide 16, show why for the corrupted file, the standard Merkle tree proof-verification computation will fail to produce the root hash, g. Namely, with almost certainty, we have that the root hash, g, is not equal to the value resulting from the verification computation.***

The security and safety of Merkle trees come from the way they handle items in their system. For a set of N children ( where N is an even number ), they create an N/2 parents, and those parents create (N/2)/2 grandparents until there are only 2 nodes left which creates the root. A collision hashing function is employed to create the parents from the children, so any change in the children would create parents with different values, and those parents would create grandparents with different values and so on. The proof for X8 would be :

**(x7, R),(f, L),(c, R)**

And the verification would be:

**g = h(h(e h(x7 x8)) c)**

Any variation in the child elements is quickly noticed because the verification mentioned above will fail.

***Show that it is unfeasible for Bob to compute such a forged proof.***

For the same reasons above Bob’s forgery would be quickly noticed. Furthermore, it would be highly improbable for Bob to find new numbers that compute to the same root as the hashing function is built to be collision-resistant. If there were two sets of numbers that resulted in the same hash this would indicate a “collision”, which is extremely rare. In this case, Bob would have to find a value when hashed has the same value as the original. Simplified it would mean:

**x8** and **x9**, such that **hash(x8)= hash(x9)**

This is why the version of SHA and Message Direct Algorithm ( MD5 ) matters so much. In lower versions of SHA, there is a higher probability of collisions. “The collision attacks against MD5 have improved so much that, as of 2007, it takes just a few seconds on a regular computer.”[[1]](#footnote-0)

***In cloud storage, it is useful for a server to be able to prove to a client not only that certain files were uploaded to the cloud, but also that other files were not uploaded. In the same cloud storage example as in the previous question, at some point, Alice requests from Bob the file with pathname Documents\D.txt. Show how Bob can prove to Alice that such a file is not in the set of uploaded files.***

Again, since the referenced rouge file exists outside of the Merkle tree, it is very easy to invalidate the existence of the file in the set even though it seems like there is a “D.txt” and “E.txt” missing from the file list. Once the set of children has been hashed and the Merkle Tree has been created the root value is dependent on the exact values derived from the number of children and their hashed values. Furthermore, a change in the number of children and the existence of an odd number of children produces two additional results that would be noticed if “D.txt” were present. An odd number of child items would mean that one child would be duplicated in the tree to maintain balance. Also, the additional items adds to the size of proof in logarithmic proportion to the number of items in the tree.

***Explain why with almost certainty, the wallet will not accept transaction T2 as valid even if the student operates a rogue full Bitcoin node that is queried by the wallet in the SPV protocol to obtain information about transactions.***

Any changes in the header of the block after the transaction has been initiated would result in a change in the local hash associated with that block. Presenting an invalid hash to the network would subsequently invalidate the version of the transaction the rogue student is trying to assert in the system. The system works on the idea of peer consensus, so it follows that one rouge block with competing ( and in this case bad ) information cannot compromise the system. The rogue student would have to operate and control at least 51% of the nodes in BTC to propagate this type of bad data, which would be infeasible and very expensive.

1. M.M.J. Stevens (June 2007). ["On Collisions for MD5"](http://www.win.tue.nl/hashclash/On%20Collisions%20for%20MD5%20-%20M.M.J.%20Stevens.pdf) (PDF). [...] we are able to find collisions for MD5 in about 224.1 compressions for recommended IHVs which takes approx. 6 seconds on a 2.6GHz Pentium 4. [↑](#footnote-ref-0)