600.454: Practical Cryptographic Systems December 5, 2017

Take Home Final

Instructor: Matthew Green *Due: 12:00pm, December 22*

Name: _____

The final should be completed individually. You are permitted to use the Internet and any printed references. Note: this final exam is *optional*. Feel free to skip it if you want to. The exam must be turned in by the conclusion of the final exam time (12pm) on December 22nd.

Please submit the completed answers via Blackboard, preferably as a text or PDF file.

Problem 1: Arguing with Merkle (35 points)

A cryptographic hash function $H : \{0,1\}^* \rightarrow \{0,1\}^d$ takes an arbitrary string as input and outputs a d-bit digest. Informally, we say that H is *collision-resistant* if it is hard for an attacker to find a pair of messages (M, M') such that $M \neq M'$ and $H(M) = H(M')$.[1]

Hashing collections of messages: Sometimes a user needs to hash a *list* of strings $(M_1, M_2, \ldots, M_N)$ to obtain a single digest D. One simple approach to this problem is to concatenate the strings together and feed the result to the hash function like so:

$$D = H(M_1 \| M_2 \| \ldots \| M_N)$$

Sometimes a user needs to *prove* to another user (a "verifier") that a single string M_j was included in the digest D. If the digest was computed as above, the user must hand over *the full list of strings* $M_1, \ldots, M_N$. (The verifier can then recompute the digest D over all of the provided strings.) The disadvantage of this approach is that conducting this proof requires the prover to send the full list of N strings to the verifier.

Ralph Merkle developed a different construction that allows a user to hash a list of messages $(M_1, \ldots, M_N)$. Merkle's approach has the following special properties:

1. It's possible to hash a list of strings into a single digest D, using any hash function H.
2. A user can construct a "proof" π that any given string M_j is in the digest D.
3. The size of this proof is generally smaller than the full list of strings.

In Merkle's construction, which we'll call $\mathsf{Merkle}(M_1, \ldots, M_N)$, each of the input strings is placed at a leaf of a binary tree. Then each internal node is computed by hashing the concatenation of that node's two children. The final digest is a single value that is located

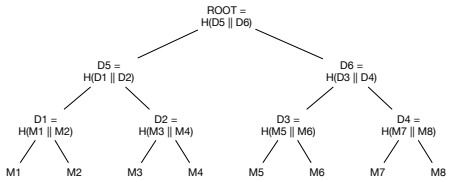

Figure 1: A Merkle tree example. The root of the tree is at the top.

at the root of the tree. An illustration of a Merkle tree with eight strings is presented in Figure 1.

Answer the following three questions:

Question 1 (5 points): You have constructed a Merkle tree using the strings $(M_1, M_2, \ldots, M_8)$ as illustrated in Figure 1. This produced a root R, which you sent to me. What values do you need to send to me (the verifier) as a proof (π) order to convince me that M_2 is contained within R? **Hint:** Ideally, your proof π should only need to contain 3 values (not counting R and M_2).

Question 2 (5 points): More generally, if I place a list of N strings $(M_1, M_2, \ldots, M_N)$ into the leaves of a Merkle tree and compute a root R, how many and *which* values need to be included in the proof π to demonstrate M_j was included in this root? Give me a brief description of how to assemble π and express the number of distinct strings you need to send as a function in terms of N (you can be approximate).

Question 3: (25 points): Let's assume the underlying hash function H is collision-resistant. Give a proof (or convincing argument) that the resulting Merkle tree proof is secure. More formally, consider the following proposition:

> If H is collision-resistant, then an attacker will not be able to output $R, (M_j, \pi), (\mathrm{M}'_j, \pi')$ such that (1) $M_j \neq M'_j$, (2) both M_j and M'_j appear at the same leaf (j) in the tree, and (3) π, π' are both valid proofs of membership for M_j and M'_j (respectively) with respect to R.

Hint: approach this as a proof by contradiction. Your proof could take the following steps:

1. Assume that H is collision resistant.
2. Now imagine that there is an adversary who outputs $R, (M_j, \pi), (M'_j, \pi')$ such that both proofs are valid with respect to R; $M_j \neq M'_j$; and M_j, M'_j are in the same leaf of the tree.
3. Show that you can now output a valid collision in the hash function H. This violates your first assumption.

[1] Formalizing the definition is a bit more complex. See `https://eprint.iacr.org/2006/281.pdf` for a good discussion. We won't sweat much about this for now.

Problem 2: TLS with a bad server random number generator (30 points)

In class we looked at the TLS protocol, with both the Diffie-Hellman and RSA handshakes (as implemented in TLS 1.2, for example). You can find descriptions of both protocols online.

Let's assume we have a TLS server with a bad random number generator like the Debian generator. Here "bad" means, the attacker can predict the exact value of all of the random bits that come out of the generator and will be used by the server during the protocol run (to generate any random value, including nonces and ephemeral keys). You can assume the client is honest and has a working random number generator.

(To make this a little less terrible, let's assume that any long-term secret keys like RSA keypairs and signing keypairs on the server were generated using a different, *working* random number generator. The generator only became broken after those keys were made.)

Answer the following questions:

1. What happens to the security of the RSA handshake in this case? What attacks can a man-in-the-middle attacker execute against the server or the client?

2. What happens to the security of the Diffie-Hellman handshake in this case? What attacks are there?

Problem 3: Bitcoin (35 points)

In class we discussed how Bitcoin mining inolves a proof of work, which is a hash function computed over a candidate block B, such that $H(B) < T$ for some *difficulty level* T. In class we left out many of the details, including that the hash function H is actually computed as *two* consecutive invocations of SHA2, and the exact process of hashing the block. You can find the full details online.

Using any written or online resources you want (but please cite them), answer the following questions:

1. Describe the exact process of assembling a candidate Bitcoin block and solving the proof of work. What goes into the block, how is it hashed, etc.?

2. There is an optimization to Bitcoin mining called ASICBOOST. This uses partial collisions in the SHA hash function in order to speed up the process of finding valid blocks. Explain how this optimization works.