Homework 3: Authentication

This homework is due **Monday, November 6** at **6 p.m.** and counts for 6% of your course grade (4% if you are a graduate student taking CMSC 33250). You will have a budget of four extensions (24-hour periods) over the course of the quarter that you can use to turn assignments in late without penalty and with no questions asked. You cannot consume partial days. Once your extensions are used up, further extensions will only be granted in extraordinary circumstances.

We encourage you to discuss the problems in general terms with other students in the class. However, the answers you turn in must be your own original work, and you are bound by the University's policy on Academic Honesty and Plagiarism. Also, please document any material discussions you had with others about this assignment (e.g., "Note: I discussed this exercise with Jane Smith").

Solutions should be submitted electronically via chisubmit in plain text format using the template found at `hw3/hw3.txt` in the upstream repository.

Concisely answer the following questions. (Limit yourself to at most 80 words per subquestion.)

1. **Authentication protocols.** A medium-sized Midwestern research university wants to implement a central sign-on facility where users authenticate themselves to an official site then receive a token that confirms their identity to all other campus sites.

   (a) Assuming the protocol is competently implemented and deployed, how might deploying this service improve security on campus?

   (b) Under the same assumptions, how might it hurt security?

   Suppose the sign-on protocol proceeds as follows: When the user visits site A, which requires authentication, site A redirects the user to the central sign-on site. Following authentication, the central sign-on site redirects the user's browser back to a standardized HTTPS URL at site A with the following parameters: u, the user's username, and $\mathrm{Sign}(u)$, a digital signature produced with the sign-on site's private key. (Assume that the corresponding public key is widely known.) The site checks that the signature is valid for u, and considers the user authorized if it is valid.

   (c) Assume a site A is controlled by an attacker. Assume a user signs on to the central sign-on site and then uses his or her authentication parameters to access site A. How can the attacker impersonate this user in order to deceive other sites that trust the sign-on protocol?

(d) Propose a simple change to the protocol that would fix the problem identified in (c).

Professor Vuln would like to provide a simple mechanism to allow members of his massive research group to authenticate to each other before they exchange confidential data. He decides to distribute a shared secret key k that will allow the group members to mutually authenticate each other. His protocol is given below:

	Alice		Bob
1	"Hello", random nonce n_1	$\longrightarrow$	
2		$\longleftarrow$	random nonce n_2, $\text{HMAC}_k(n_1)$
3	*verifies:* $\text{HMAC}_k(n_1)$		
4	$\text{HMAC}_k(n_2)$	$\longrightarrow$	
5			*verifies:* $\text{HMAC}_k(n_2)$

(e) Mallory, an unauthorized outsider, is able to engage in authentication attempts with any of the group members. She can send them messages, but she cannot intercept or modify the messages they send to each other. How can she convince Bob she is a member of the group?

(f) Modify the protocol so that it achieves mutual authentication securely while still using a single shared secret, and *briefly* argue that your answer is correct.

2. **Password cracking.** Suppose you are in charge of security for a major web site, and you are considering what would happen if an attacker stole your database of usernames and passwords. You have already implemented a basic defense: instead of storing the plaintext passwords, you store their SHA-256 hashes.

Your threat model assumes that the attacker can carry out 4 million SHA-256 hashes per second. His goal is to recover as many plaintext passwords as possible from the information in the stolen database.

Valid passwords for your site may contain only characters a–z, A–Z, and 0–9, and are exactly 8 characters long. For the purposes of this homework, assume that each user selects a random password.

(a) Given the hash of a single password, how many hours would it take for the attacker to crack a single password by brute force, on average?

(b) How large a botnet would he need to crack individual hashes at an average rate of one per hour, assuming each bot can compute 4 million hashes per second?

Based on your answer to part (a), the attacker would probably want to adopt more sophisticated techniques. You consider whether he could compute the SHA-256 hash of every valid

password and create a table of $(\textit{hash}, \textit{password})$ pairs sorted by hash. With this table, he would be able to take a hash and find the corresponding password very quickly.

(c) How many bytes would the table occupy?

It appears that the attacker probably won't have enough disk space to store the exhaustive table from part (b). You consider another possibility: he could use a *rainbow table*, a space-efficient data structure for storing precomputed hash values.

A rainbow table is computed with respect to a specific set of N passwords and a hash function H (in this case, SHA-256). We construct a table by computing *m chains*, each of fixed length k and representing k passwords and their hashes. The table is constructed in such a way that only the first and last passwords in each chain need to be stored: the last password (or *endpoint*) is sufficient to recognize whether a hash value is likely to be part of the chain, and the first password is sufficient to reconstruct the rest of the chain. When long chains are used, this arrangement saves an enormous amount of space at the cost of some additional computation.

Chains are constructed using a family of *reduction functions* $R_1, R_2, \ldots, R_k$ that deterministically but pseudorandomly map every possible hash value to one of the N passwords. Each chain begins with a different password p_0. To extend the chain by one step, we compute $h_i := H(p_{i-1})$ then apply the ith reduction function to arrive at the next password, $p_i = R_i(h_i)$. Thus, a chain of length 3 starting with the password `hax0r123` would consist of (`hax0r123`, $R_1(H(\texttt{hax0r123}))$, $R_2(H(R_1(H(\texttt{hax0r123}))))$).

After building the table, we can use it to quickly find a password p_* that hashes to a particular value h_*. The first step is to locate a chain containing h_* in the table; this requires, at most, about $k^2/2$ hash operations. Since h_* could fall in any of $k-1$ positions in a chain, we compute the password that would *end up* in the final chain position for each case. If we start by assuming h_* is right before the end of the chain and work backwards, the possible endpoints will be $R_k(h_*)$, $R_k(H(R_{k-1}(h_*)))$, ... We then check if any of these values is the endpoint of a chain in the table.

If we find a matching endpoint, we proceed to the second step, reconstructing this chain based on its initial value. This chain is very likely to contain a password that hashes to h, though collisions in the reduction functions cause occasional false positives.

(d) For simplicity, make the optimistic assumption that the attacker's rainbow table contains no collisions and each valid password is represented exactly once. Assuming each password occupies 8 bytes, give an equation for the number of bytes in the table in terms of the chain length k and the size of the password set N.

(e) If $k = 5000$, how many bytes will the attacker's table occupy to represent the same passwords as in (c)?

(f) Roughly how long would it take to construct the table if the attacker can add 2 million chain elements per second (rather than 4 million because each password requires computing both H and R_i)?

(g) Compare these size and time estimates to your results from (a), (b), and (c).

You consider making the following change to the site: instead of storing SHA-256(*password*) it will store SHA-256(*server_secret*||*password*), where *server_secret* is a randomly generated 32-bit secret stored on the server. (The same secret is used for all passwords.)

(h) How does this design partially defend against rainbow table attacks?

(i) Briefly, how could you adjust the design to provide even stronger protection? □