Homework 4: Internet Security

This homework is due **April 20, 2022** at **11:59 p.m.** and counts for 10% of your course grade. Late submissions on the next day (24 hours) will be penalized by 30%. Submission within second and third day (24-72 hours) will penalized by 60%. We will not accept any submissions after that. If you have a conflict due to travel, interviews, etc., please plan accordingly and turn in your homework early.

Please submit one pdf file to D2L named **[Your NET_ID]_HW4.pdf**. *Please do not submit an image of your handwritten solutions.*

We encourage you to discuss the problems and your general approach 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 of Arizona Honor Code.

Submission guidelines. There is not length limit on the answers you provide. As long as your write up clearly expresses your answer, you should be fine. If you want to submit the accompanied code (not required), paste the code at the end of the document, mark it with the question number and refer it from the answer. Make sure to comment your code with clear explanations and be sure to *cite any references used*.

---

1. **Authentication protocols.** A large Sonoran 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) **[2 points]** Assuming the protocol is competently implemented and deployed, how might deploying this service improve security on campus?

   (b) **[4 points]** 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 $\text{Sign}(u)$, 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 so.

   (c) **[4 points]** If site A is controlled by an attacker, how can it trivially impersonate the user to other sites that trust the sign-on protocol?

(d) **[4 points]** 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 Alice	$\longrightarrow$	
2		$\longleftarrow$	random nonce n_2, Bob, $\text{HMAC}_k(n_1 \| \| Alice)$
3	*verifies:* $\text{HMAC}_k(n_1 \| \| Alice)$		
4	$\text{HMAC}_k(n_2 \| \| Bob)$	$\longrightarrow$	
5			*verifies:* $\text{HMAC}_k(n_2 \| \| Bob)$

(e) **[6 points]** Mallory, an unauthorized outsider, is able to engage in authentication attempts with any of the group members. She can send them messages, replay messages, and drop messages, but she cannot modify the content of messages in transit. How can she convince Bob she is a member of the group? Note that someone in the group may try to initiate a conversation with Mallory.

(f) **[10 points]** 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) **[6 points]** 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) **[4 points]** 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?

c_i	c_1	c_2	c_3
p_0	hax0r123	A3VpW57s	WEBCuGpn
$h_1 := H(p_0)$	0x0802...	0xe655...	0x3fb8...
$p_1 = R_1(h_1)$	xkjTCScS	Kr24FT6m	qnrnnEn6
$h_2 := H(p_1)$	0xa10c...	0xdc0d...	0x233a...
$p_2 = R_2(h_2)$	uUVdGYvN	RHkDFAXc	fMVvMyyq

Table 1: An example rainbow table

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 (*hash*, *password*) pairs sorted by hash. With this table, he would be able to take a hash and find the corresponding password very quickly.

(c) **[4 points]** 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. Table 1 shows a simple rainbow table with $m = 3$ chains (columns): c_1, c_2, c_3. Each chain stores $k = 3$ passwords (rows): p_0, p_1, p_2 (rows).

Chains are constructed using a family of *reduction functions* $R_1, R_2, \ldots, R_{k-1}$ that deterministically map each hash value to one of the N possible passwords. Each R_i should be a different pseudorandom function. 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`, $R_1(H(\texttt{hax0r123}))$, $R_2(H(R_1(H(\texttt{hax0r123}))))$).

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. In Table 1, we only need to store p_0 and p_2 in each chain.

After building the table, we can use it to quickly find a password p_* that hashes to a particular value h_*. First we apply R_{k-1} to h_* and compare it to the endpoints in the table. If this value is present as an endpoint, we can reconstruct the chain from the corresponding starting point and obtain the original value. If this value is not present as an endpoint, we move backward one step in the chain and check whether $R_{k-1}(H(R_{k-2}(h_*)))$ is an endpoint of any chain. In our example rainbow table in Table 1, we only need to compute $R_2(h_*)$, $R_2(H(R_1(h_*)))$ and

check if these values match with any endpoints of the table. In general, the number of hash operations we need to perform is $k(k-1)/2$.

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) **[4 points]** 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) **4 points]** If $k = 5000$, how many bytes will the attacker's table occupy to represent the same passwords as in (c)?

(f) **[4 points]** Roughly how long would it take to construct the table if the attacker can add 2 million chain elements per second?

(g) **[4 points]** 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) **[4 points]** How does this design partially defend against rainbow table attacks? Note that if the hashes are compromised, likely the secret is too.

(i) **[4 points]** Briefly, how could you adjust the design to provide even stronger protection?

(j) **[2 points]** Computing SHA-256 hash values are now extremely cheap thanks to relatively inexpensive bitcoin mining ASIC hardware, such as ANTMINER S1, which is a Bitcoin mining rig with a speed of 180 GH/s (1 GH/s is 10^9 hashes per second). Update your answer in part (a) accordingly assuming an attacker has access to such ASIC hardware. What is the implication?