CIS4212 Trustworthy Infrastructures
Undergrad Group 17
Henrique Gamonal De Castro & Carlos Pineda
2/15/2026

Homework 2 Extra Credit

Cryptographic Puzzle Construction

Why is the k leading zeros construction simpler to implement and verify than a partial pre-image construction?

The "leading zeros" method is generally considered simpler and more efficient than the "partial pre-image" method because of how the server verifies the anwser. In a leading zeros puzzle, the server just has to hash the client's answer and check if the resulting hash value is smaller than a certain target number (which visually looks like the hash starting with a bunch of zeros). This is a very fast, stateless numerical comparison. In contrast, a partial pre-image puzzle often requires the server to check if the client's answer matches a specific, hidden value that the server might have to store or look up. The leading zeros approach allows the server to verify the work without needing to remember exactly which puzzle it sent to which client, saving memory.

What advantage does partial pre-image have over k leading zeros?

The big plus of the partial pre-image method (using HMAC) is speed. For the leading zeros way, the server usually has to digitally sign the puzzle so you know its legit. Digital signatures use asymmetric crypto which is super slow and eats up the servers CPU. The HMAC method uses symmetric crypto which is lightning fast. So it takes way less computer power for the server to make the puzzle using HMAC compared to the digital signature way.

Client Server Puzzles

A) Desirable Properties of the (X,Y)-Puzzle Design

For this puzzle system to work properly, it relies on four main properties that keep the server safe. First, the puzzles have to be stateless to stop a Denial of Service attack where an attacker tries to fill up the servers memory. The server fixes this by creating the puzzle using a secret code S that only it knows, combined with the timestamp and request info, like in the formula Y = HMAC(S, timestamp, Request). When you send back the answer, the server just runs that formula again to check if its right, so it never has to save a database record of what it sent you.

Second, the puzzles need to be easy to verify because we dont want the defense mechanism to crash the server itself. The client has to do a massive brute-force search taking millions of operations to find the missing part of the puzzle, but the server only has to run one single hash calculation to check if the answer matches the target Y. This creates a huge difference in work where the server does almost nothing while the client does all the heavy lifting.

Third, the hardness of the puzzles has to be controllable so the server can react if an attack gets worse. The server can change the difficulty by adjusting the variable k, which is how many bits of the answer are missing. If the server increases k, the number of guesses the client has to make doubles for every extra bit, allowing the server to fine-tune the delay from milliseconds to minutes.

Finally, the puzzles use standard crypto primitives instead of weird custom math that might have bugs. This design uses standard hash functions like SHA or MD5 because they are proven to be one-way,

meaning you cant just reverse them to find the answer. Since these standards are solid, we know the only way to solve the puzzle is by doing the actual work and not by finding a shortcut.

B) Advantage of Sub-Puzzles

The main reason we use sub-puzzles instead of one big puzzle is to make it impossible for an attacker to just guess the answer without doing the work. If you have one big puzzle, the chance of guessing it is small, but if you break it into m smaller puzzles, the attacker has to guess every single one of them correctly in a row to get in. The probability of guessing one big puzzle might be 2 to the power of negative (k plus log m), but guessing m small puzzles is 2 to the power of negative (k times m). 2 to the power of negative km is an exponentially smaller number, meaning the attackers chance of lucking out drops to basically zero while the honest users work stays the same.

C) Design Rationales for Broadcast Puzzles

The first rationale is about efficiency through broadcasting. In a standard system, the server has to compute a unique puzzle for every single request it receives, which wastes CPU power. In this design, the server generates one single puzzle, digitally signs it, and broadcasts this same message to all users at once. This allows the server to do the work just once to serve milllions of users and lets it pre-compute these parameters offline so it doesn't waste processing power during an active attack.

The second rationale is about binding the solution to your specific identity to prevent cheating. Since everyone receives the exact same broadcast puzzle, a hacker could theoretically solve it once and share the answer with thousands of bots to bypass the protection. To fix this, the puzzle's hash formula requires you to include your specific "Client Identity" inside the calculation. This ensures that if a hacker solves the puzzle, the answer is mathematically tied to their ID and will not validate for any other Client ID.

The third rationale is about allowing legitimate users to create multiple valid tickets without contacting the server again. The design allows you to include your own random number, called a "Client Nonce", in the solution hash. This is helpful if you need to access the server multiple times, like downloading ten different images for a webpage. Instead of asking the server for ten new puzzles, you can take that one broadcast puzzle and solve it ten times using ten different random numbers you generated yourself, proving you did the work without making the server issue new challenges.