CS 491/591: Economics of Distributed Systems
Homework 1

Adam Fasulo

February 14, 2025

Problem 1: Collision Resistance vs. Puzzle-Friendliness

Given collision-resistant $H : X \to \{0, \ldots, 2^n - 1\}$, construct $H' : X \to \{0, \ldots, 2^m - 1\}$ $(m > n)$ that is collision-resistant but *not* puzzle-friendly.

Construction of H'

Let $m = n + 1$. Define $H'(x) = b \parallel H(x)$, where b is the last bit of x (0 or 1).

Proof of Collision Resistance of H'

Claim: H collision-resistant $\implies$ H' collision-resistant.

Proof: If $H'(x) = H'(y)$ for $x \neq y$, then x and y must end in the same bit (otherwise, the prepended bit differs). Thus, $H(x) = H(y)$, a collision in H. Since H is collision-resistant, H' is also.

Proof that H' is NOT Puzzle-Friendly

Claim: H' is not puzzle-friendly.

Proof: Let $D = 2^{n-1}$. We seek x such that $H'(x) \leq 2^m/D = 4$. If x ends in 0, $H'(x) = 0 \parallel H(x)$. Choosing $x = 0$ gives $H'(x) = 0 \leq 4$. Thus, H' is not puzzle-friendly.

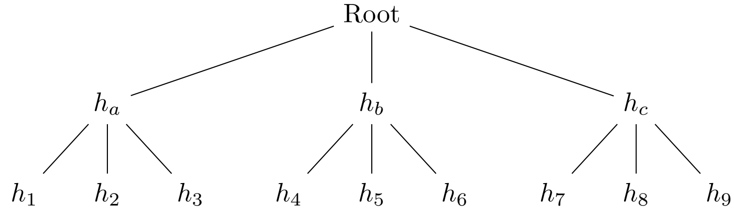
Conclusion

Collision resistance does not imply puzzle-friendliness.

Problem 2: k-ary Merkle Trees

(a) 3-ary Merkle Tree Example (n=9)

Commitment Computation: Leaf nodes are $h_i = H(T_i)$. Internal nodes are hashes of concatenations of their children's hashes. The root is the commitment.



$h_i = H(T_i)$, $h_a = H(h_1 \parallel h_2 \parallel h_3)$, $h_b = H(h_4 \parallel h_5 \parallel h_6)$, $h_c = H(h_7 \parallel h_8 \parallel h_9)$, Root $= H(h_a \parallel h_b \parallel h_c)$.

Proof of Inclusion for T_4: Alice provides T_4, h_5, h_6, h_a, h_c, and the root value. Bob computes $h_4 = H(T_4)$, $h_b = H(h_4 \parallel h_5 \parallel h_6)$, $Root' = H(h_a \parallel h_b \parallel h_c)$, and verifies $Root' =$ root.

(b) Proof Length as a Function of n and k

Proof Length $= (k-1) \cdot \lceil \log_k n \rceil$ (provided sibling hashes at each of the $\lceil \log_k n \rceil$ levels).

(c) Minimizing Proof Size (Binary vs. 3-ary)

Binary (k=2): Proof Length $= \lceil \log_2 n \rceil$. 3-ary (k=3): Proof Length $= 2 \cdot \lceil \log_3 n \rceil \approx 1.26 \lceil \log_2 n \rceil$.

Conclusion: Binary Merkle trees minimize proof size.

Problem 3: One-Way Functions and P vs. NP

Claim: One-way functions exist $\implies$ P $\neq$ NP.

Proof (Contrapositive): Assume P = NP. Let f be poly-time computable with output length $\Theta(n)$ for inputs of length n. We'll show f is invertible in polynomial time.

Define $L = \{(y, w) \mid \exists x$ such that $f(x) = y$ and x has prefix $w\}$.

Step 1: $L \in$ **NP.** Certificate: x. Verification: Check if w is a prefix of x and $f(x) = y$ (poly-time).

Step 2: $L \in$ **P.** Since P = NP and $L \in$ NP, then $L \in$ P. Let A be a poly-time decider for L.

Step 3: Inversion Algorithm ($Invert_f$): Input: y (length m). Initialize: $w = \epsilon$. Iterate for $i = 1$ to n: If $A(y, w \parallel 0) =$ "yes", $w = w \parallel 0$. Else if $A(y, w \parallel 1) =$ "yes", $w = w \parallel 1$. Else, halt (no inverse). Output: $x = w$.

Step 4: Correctness and Running Time: $Invert_f$ builds x bit-by-bit, using A to ensure each prefix is valid. The loop runs n times, with each iteration taking polynomial time (due to A). Since n is polynomial in m, the algorithm is poly-time in m.

Conclusion: P = NP implies poly-time inversion of any poly-time computable function, contradicting one-way function existence. Thus, one-way functions exist $\implies$ P $\neq$ NP.

Problem 4: Finite Order in Finite Groups

Claim: In a finite group G, every element has finite order.

Proof:

Let G be a finite group with order —G— (the number of elements in G). Let * be the group operation (we'll use multiplicative notation). Let g be an arbitrary element of G, i.e., $g \in G$.

Consider the sequence of elements generated by successive powers of g:

$$g^1, g^2, g^3, g^4, \ldots$$

Since G is finite, this infinite sequence *must* contain repetitions. That is, there must exist positive integers i and j with $i < j$ such that:

$$g^i = g^j$$

Now, because every element in a group has an inverse, we can multiply both sides of the equation by the inverse of g^i, denoted as $(g^i)^{-1}$:

$$(g^i)^{-1} \cdot g^i = (g^i)^{-1} \cdot g^j$$

By the properties of inverses, $(g^i)^{-1} \cdot g^i = e$, where e is the identity element of the group. Also, $(g^i)^{-1} = (g^{-1})^i = g^{-i}$. Thus:

$$e = g^{-i} \cdot g^j = g^{j-i}$$

Since $i < j$, the exponent $j - i$ is a positive integer. Let $k = j - i$. Then we have:

$$g^k = e$$

where k is a positive integer. This means that there exists a positive integer power of g that results in the identity element.

The *order* of an element g, denoted as $ord(g)$, is the *smallest* positive integer n such that $g^n = e$. While we've shown that *some* positive integer k exists such that $g^k = e$, we haven't explicitly shown that the *smallest* such integer is finite. However, we know that the set of positive integers $\{n \mid g^n = e\}$ is non-empty (since it contains k) and is a subset of the natural numbers. By the well-ordering principle, this set has a smallest element. This smallest element is, by definition, the order of g, and it must be less than or equal to k, and hence is finite.

Therefore, every element g in the finite group G has a finite order.

Problem 5: The Modular Arithmetic Magic Trick

(a) Computing the Outcome by Hand

Let $p = 13$ and $x = 5$ be our chosen integer. We need to compute $(13 \cdot 5 + 1)^{13} \pmod{13}$.

1. **Simplify inside the parentheses:**

$$13 \cdot 5 + 1 = 65 + 1 = 66$$

2. **Apply modular arithmetic:** We want to find $66^{13} \pmod{13}$. We can reduce 66 modulo 13 *before* raising it to the power:

$$66 \equiv 66 - (5 \cdot 13) \equiv 66 - 65 \equiv 1 \pmod{13}$$

3. **Compute the power:** Now we have $1^{13} \pmod{13}$. Since 1 raised to any power is 1:

$$1^{13} \equiv 1 \pmod{13}$$

Final Answer: 1 (Eerie, indeed!)

(b) Explaining the Magic Trick

The magic trick works because of Fermat's Little Theorem and properties of modular arithmetic.

- **Fermat's Little Theorem:** If p is a prime number and a is an integer not divisible by p, then:

$$a^{p-1} \equiv 1 \pmod{p}$$

- **The Setup:** The expression is $(px + 1)^p \pmod{p}$. Since $p = 13$ is prime, we can apply Fermat's Little Theorem.

- **Simplification:** Notice that $px \equiv 0 \pmod{p}$ for any integer x, because px is a multiple of p. Therefore:

$$(px + 1) \equiv (0 + 1) \equiv 1 \pmod{p}$$

- **Final Calculation:** Substituting this into the original expression:

$$(px + 1)^p \equiv 1^p \equiv 1 \pmod{p}$$

The trick works because $(px + 1)$ is always congruent to 1 modulo p. Raising 1 to any power still results in 1.

(c) A New Magic Trick

The Trick:

1. Choose a prime number p. Let's use $p = 7$.
2. Choose any integer a that is *not* a multiple of p. Let's use $a = 3$.
3. Compute $a^{p-1} \pmod{p}$.
4. The result will always be 1.

Demonstration:
We have $p = 7$ and $a = 3$. We compute $3^{7-1} \pmod{7} = 3^6 \pmod{7}$.

$$\begin{aligned}
3^1 &\equiv 3 \pmod{7} \\
3^2 &\equiv 9 \equiv 2 \pmod{7} \\
3^3 &\equiv 3 \cdot 2 \equiv 6 \pmod{7} \\
3^4 &\equiv 3 \cdot 6 \equiv 18 \equiv 4 \pmod{7} \\
3^5 &\equiv 3 \cdot 4 \equiv 12 \equiv 5 \pmod{7} \\
3^6 &\equiv 3 \cdot 5 \equiv 15 \equiv 1 \pmod{7}
\end{aligned}$$

The result is 1.

Explanation:
This trick is a direct application of Fermat's Little Theorem. As stated before: If p is prime and a is not divisible by p, then $a^{p-1} \equiv 1 \pmod{p}$. The trick simply demonstrates this theorem.

Problem: Fiat-Shamir Signature - Toy Example

We are given:

- Group: $Z_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
- Generator: $g = 2$
- Alice's private key: $x = 2$
- Alice's public key: $y = g^x = 2^2 = 4$
- Target (commitment): $t = g^7 = 2^7 \equiv 7 \pmod{11}$ (since $2^7 = 128 = 11 \cdot 11 + 7$)
- Challenge: $c = 2$

(a) Alice's Response (r)

Alice needs to find r such that $g^r y^c \equiv t \pmod{11}$. Substituting the given values:

$$2^r \cdot 4^2 \equiv 7 \pmod{11}$$
$$2^r \cdot 16 \equiv 7 \pmod{11}$$
$$2^r \cdot 5 \equiv 7 \pmod{11}$$

We need to find the multiplicative inverse of 5 modulo 11. We can do this by trying values or using the extended Euclidean algorithm. We find that $5 \cdot 9 = 45 \equiv 1 \pmod{11}$, so $5^{-1} \equiv 9 \pmod{11}$. Multiplying both sides of our equation by 9:

$$9 \cdot 2^r \cdot 5 \equiv 9 \cdot 7 \pmod{11}$$
$$2^r \equiv 63 \pmod{11}$$
$$2^r \equiv 8 \pmod{11}$$

Since $2^3 = 8$, we have $r \equiv 3 \pmod{11}$.
Therefore, Alice's correct response is $\boxed{r = 3}$.
We check this answer:

$$2^3 \cdot 4^2 \equiv 8 \cdot 16 \equiv 8 \cdot 5 \equiv 40 \equiv 7 \pmod{11}$$

(b) Hardness of Finding r

When the group size is large (i.e., the modulus is a very large prime), finding the correct r without knowing x becomes computationally hard. This difficulty stems from the Discrete Logarithm Problem (DLP).

The equation we're essentially trying to solve (after substituting $y = g^x$) is:

$$g^r \cdot (g^x)^c \equiv t \pmod{p}$$
$$g^{r+xc} \equiv t \pmod{p}$$

To find r, an attacker would need to solve for the exponent r in this congruence. Taking the discrete logarithm base g of both sides, this becomes:

$$r + xc \equiv \log_g t \pmod{\phi(p)}$$

In the case of Z*p with prime p.

$$r \equiv \log_g t - xc \pmod{p-1}$$

If the attacker *doesn't* know x (Alice's private key), they need to compute $\log_g t$ and essentially solve for x, given y. Computing discrete logarithms in large groups (especially those chosen for cryptographic purposes) is believed to be computationally infeasible.

(c) Non-Interactive Zero-Knowledge Proof (Choosing c)

If Alice wants to prove she knows x *without* relying on Bob to choose the challenge c (making it non-interactive), she can use a cryptographic hash function, H.

Instead of Bob providing c, Alice computes c herself as:

$$c = H(g, y, t, M)$$
$$c = H(g, y, g^k, M)$$

Where M represents the message being signed, or some data associated with this specific proof. The key is that c is now a hash of values that include t (or, equivalently, g^k).

Here's why this works:

1. **Binding** Alice cannot choose t (or k) *after* knowing c, because c is derived from a hash that *includes* t. This prevents her from choosing a t that makes it easy to find a suitable r. 2. **Soundness:** If Alice doesn't know x, it's computationally infeasible for her to find a t and an r that will satisfy the verification equation, given the hash c. Any attempt to forge a proof will involve finding a collision in the hash function, which is assumed to be hard. 3. **Zero-Knowledge:** If Alice follows the protocol, she doesn't reveal any information about *x* directly to Bob. The only information bob can calculate is either public information, or requires solving the discrete logarithm.

This use of a hash function transforms the interactive Fiat-Shamir protocol into a non-interactive zero-knowledge proof, and it forms the basis of many digital signature schemes.