EN.601.441/641: Assignment 1a

Zeyn Schweyk
Blockchains and Cryptocurrencies

September 10, 2024

Question 1. Hash Functions (5 points)

Hash functions are designed to map arbitrarily large inputs to a certain fixed length. SHA2, for example, maps inputs to 256-bits. However, since we are only dealing with phone numbers, which have a fixed length of 10 where each digit can be one of 10 possibilities (from 0 to 9), there are $10^{10} = 10,000,000,000$ distinct phone numbers. Since the input to the hash function is heavily restricted in terms of length and possibilities, it doesn't necessarily matter how 'good' the hash function is because a brute-force attack is entirely feasible. Mallory could continuously hash random phone numbers until she gets matches for all of the hashes in Alice's contacts.

Question 2. Signatures (5 points)

Since signatures are computationally expensive to compute especially over large messages, one could simply hash the message first, assuming we are given a collision-resistant hash function, and then proceed to sign it. Similarly when $verify$ing, one could simply hash the message before proceeding to verify the signature. This hashing is useful because it compresses the message into a fixed-length value to input into $sign$ and $verify$
(sk, pk) := generateKeys(keysize)
sig := sign(sk, H(message))
isValid := verify(pk, H(message), sig)

Question 3. Merkle Damgard (10 points)

1. Describe an easy way to find two messages that are broken up into the same number of pieces, which have the same hash value under h.

   Let a be an n-bit number, let b be an t-bit number, let c_1 be a $t-2$-bit number, and let c_2 be an $t-1$-bit number, where n and t are integers defined in the problem.

   Since H expects block sizes of n-bits for the first piece and t-bits for each piece that follows, if the last piece doesn't have the correct number of bits, t, H will pad that last piece with zeros.

   Let $m_1 = a||b||c_1$

Let $m_2 = a||b||c_2$

We see that H will pad c_1 and c_2 with zeros such that they are both t-bit numbers. Basically, if $c_1 \neq c_2$ and c_2 is a power of 2 greater than c_1, we'll see that H will pad both c_1 and c_2 by appending 0s. In doing so, those new pieces will have the same value, and therefore two distinct messages with the same number of pieces will evaluate to the same hash digest, causing a collision.

A concrete example of this is if we assume that $n = t = 4$ and if we let $m_1 = 1111||1111||100$ and $m_2 = 1111||1111||10$.

m_1 will be modified to $m_1 = 1111||1111||1000$ and m_2 will be modified to $m_2 = 1111||1111||1000$. Since these messages are now the same, although they were different before the padding, they will hash to the same value under H.

2. Let $m_1 = a||b||c$ be a message composed of a, b, and c concatenated to each other, where $|a| = n$, $|b| = |c| = t$, such that $|m_1| = n + 2t$. Since m_1 can be split into 3 parts of the correct length, with $y_0 = a$, $x_1 = b$, and $x_2 = c$, we know from how H is defined that $H(m_1) = y_2$:

   $y_0 = a$
   $y_1 = h(y_0||x_1) = h(a||b)$
   $y_2 = h(y_1||x_2) = h(h(a||b)||c)$

   Since there is no initial IV block according to how the hash function H is defined, it is clear how H iteratively constructs a hash based on the message's component pairs. However, since there is no final padding block which encodes the length of the message, one could take advantage of this and find a collision by constructing a shorter message m_2 of length $|m_2| = n + t$. If we allow $temp = H(a||b) = h(a||b)$, we can see how $temp$ is equivalent to y_1 of m_1. Now, if we choose $m_2 = temp||c$, letting $y_0 = temp$ and $x'_1 = c$, we see that $H(m_2) = y'_1$, as defined below:

   $y'_0 = temp$
   $y'_1 = h(y'_0||x'_1) = h(temp||c) = h(H(a||b)||c) = h(h(a||b)||c)$

   From this, we see that $y_2 = y'_1$. In other words, if we choose messages m_1 and m_2, as defined above, where $m_1 \neq m_2$ and $|m_2| < |m_1|$, we see that they have the same hash value under H since $H(m_1) = H(m_2)$. We have therefore found a collision.

Question 4. PoW difficulty (10 Points)

If we construct a hash H' with length $n' > n$ and define H' to start with $n' - n$ zeros, such that $H'(p||s) = \{0\}^{n'-n}||H(p||s)$, we can argue that H' is indeed collision resistant, since we have only just prepended 0s to H. Therefore, it is clear how a collision on H' will happen if there is a collision on H.

Since we have defined H' to output $n' > n$ bits, then the number of possible solutions $s \in S$ to the puzzle $p \in P$ has increased since the threshold $\frac{2^{n'}}{d} > \frac{2^n}{d}$ has increased for a fixed difficulty d. What's important here is that there are still 2^n distinct possible outputs

of H', but since the threshold ratio is now applied to a larger output space, the *number* of possible solutions that exist less than the threshold is larger.

For different values of n', we can analyze how the threshold changes as a result. When $n' = n + 1$, the threshold will be 2 times larger than the threshold of H. When $n' = n + 2$, the threshold will be 4 times larger than the threshold of H. Extending this logic, we see that when $n' = n + a$, for some integer a, the threshold will be 2^a times larger than that of H. So we can see that for every extra zero we prepend, we raise the threshold by a factor of 2.

This shows how for a fixed d, defining a new hash function H' as above would *not* make H' Proof-of-Work secure since it would become 2 times easier to find a solution $s \in S$ to the puzzle p for each extra prepended zero. If d were to scale appropriately such that it counteracts each prepended zero, H' would be PoW secure. However, since d is fixed, we have proved that H' cannot be PoW secure.

Question 5. Bitcoin (5 points)

Due to how the Bitcoin PoW consensus mechanism works, whenever a node solves the hash puzzle and proposes a new block, all other nodes must either confirm or deny that block by either building off that proposed block or not, respectively. For this reason, there can be many temporary forks of the blockchain at any point, so it is clear how the most recently broadcast block has to wait for other nodes to verify and build off of it, so that that block (and fork) becomes more reliable. Because nodes typically choose to build off of the longest current temporary fork, the verification of the current block will only strengthen over time. In Bitcoin, for a block to be deemed reliable, one typically waits for 6 other nodes to build off of that block and continue that chain. Then, the probability that that fork is the agreed upon state of the blockchain is very close to 100%.