
Problem Set 3

Submit this problem set in PostScript, PDF, or MS Word format to `6857-submit@csail.mit.edu` before lecture on the due date. We have provided templates for L^AT_EX and Microsoft Word on the course website. Each solution must appear on separate sheets of paper. Mark the top of each sheet with your name(s), the course number (6.857), the problem set number and question, and the date.

You are to work in groups of three or four people and should submit a single set of solutions for all problems parts designated [**Group**]. You should turn in a separate, individual solution to any problems designated [**Individual**].

We may distribute our favorite solution to each problem as the “official” solution. If you do not wish for your homework to be used as an official solution, or if you wish that it only be used anonymously, please note this on your homework.

Problem 3-1. Term Project Ideas [**Individual**]

Problem 3-2. Adding One-Time Pads [**Group**]

Peter Padadder knows from 6.857 that he shouldn't ever re-use a one-time pad. He has eight messages M_1, M_2, \dots, M_8 to encrypt for his friend Mary. Each message contains the title of a sci-fi movie recommendation. The titles are of varying lengths; none are longer than 34 characters. Peter encodes the upper case alphabet and the space character as integers modulo 27 as follows:

$$\begin{aligned}(\text{space}) &= 0 \\ A &= 1 \\ B &= 2 \\ &\vdots \\ Z &= 26\end{aligned}$$

Peter thus wants to create eight pads P_1, P_2, \dots, P_8 so that he can encrypt his messages for Mary. Peter then encrypts each message M_i 's character j as $C_{ij} = M_{ij} + P_{ij} \pmod{27}$. Note that ciphertext C_i is as long as the plaintext; if M_i is shorter than P_i then the extra pad characters are ignored.

Peter and Mary try to be clever. They create two random strings (“proto-pads”) A and B of length 34 characters each, where each character has been chosen randomly modulo 27. They then define their eight pads as:

$$\begin{aligned}P_1 &= A \\ P_2 &= B \\ P_3 &= A + B \\ P_4 &= A - B \\ P_5 &= -A \\ P_6 &= -B \\ P_7 &= -A - B \\ P_8 &= -A + B\end{aligned}$$

Peter figures this is OK since no pad is repeated. Peter encrypts his eight movie recommendations using these pads, and sends them to Mary. His encrypted messages (in arbitrary order) consist of:

‘‘COFBMEGCOFBAVGWVXXOURVIYE’’
 ‘‘CESYIPCUNMAKFZRBKUGNM’’
 ‘‘HMFBDKVPGUMGU HHQC’’
 ‘‘X VM ONSHNRYPFBPNRUGZQQOQUPL’’
 ‘‘ZIWAMFRYCCIDCPQV’’
 ‘‘M WSPN GUJYR FIWSJZQYPADETTACFRA’’
 ‘‘BLEIJGLAISASZZRXBLALXJHASNS’’
 ‘‘XGKXLCUSXQYIUUQCCPPXHAAT GSMFMELLP’’

These eight ciphertexts are available on the 6.857 web site at:

<http://crypto.csail.mit.edu/classes/6.857/ciphers.html>.

Show that Eve, the eavesdropper, can reconstruct these movie recommendations. Explain your work, and explain why Peter’s reasoning was faulty.

Solution: The one time pads are:

$A = \text{“}XWAFPPQPM SXFFJXKHSZZWPROFXPXVQKQ\text{”}$ and

$B = \text{“}FARALMCLUIBLOHHYMHTJVSDWHHYSFVWRF\text{”}$

‘‘FLYING DISC MAN FROM MARS’’

‘‘FIRST MEN IN THE MOON’’

‘‘EIGHT LEGGED FREAKS’’

‘‘OVERDRAWN AT THE MEMORY BANK’’

The resulting movie titles are:

‘‘THE ATOMIC BRAIN’’

‘‘SANTA CLAUS CONQUERS THE MARTIANS’’

‘‘ZOMBIES OF THE STRATOSPHERE’’

‘‘FLASH GORDON CONQUERS THE UNIVERSE’’

Problem 3-3. Ballot Anonymizer [Group]

Suppose an electronic polling station is going to store ballots B_0, \dots, B_{t-1} in an array of size p , where p is prime and $t < p$. Although the ballots do not have any identifying information, if they are stored sequentially, someone with access to the final ballot array could associate the ballot stored at location i with the i th voter.

Consider a scheme that places ballot B_i in array index $ai+b \pmod p$, where $a, b \in \mathbb{Z}_p^*$ and $1 \leq a \leq \frac{p-1}{2}$. This scheme is intended to jumble the ballot ordering so that someone cannot associate a ballot’s storage location with a particular voter.

(a) Explain why the ballot positions are distinct.

Solution: Suppose there are two values x and y such that $ax + b = ay + b \pmod p$. Then it must be the case that $ax = ay \pmod p$. Since these values are modulo a prime, a must have some inverse a^{-1} , and $a^{-1}ax = x = y = a^{-1}ay \pmod p$.

- (b) Suppose just three voters have cast their vote and their ballots are stored under this scheme. Explain how you can determine the order that the ballots were cast, based on where they are stored.

Solution: The ballots are stored in b , $a + b$ and $2a + b$, although we don't know which is which. By summing the three values, we obtain $3a + 3b$, and can simply obtain $a + b$, which was the second ballot cast. Call this value $m = a + b$. Since $a < \frac{p-1}{2}$, we know that the other two values must fall in disjoint halves of \mathbb{Z}_p^* . We can quickly find $b \in [m - \frac{p-1}{2}, m)$ and $2a + b \in (m, m + \frac{p-1}{2}]$.

- (c) Suppose the adversary sees that the ballots A, B, C, D, E have been placed in memory positions $0, 1, \dots, 6$ as follows:

0	1	2	3	4	5	6
A	-	B	C	D	E	-

Can an adversary who sees this pattern of ballots in the memory array determine the order of voting? If so, how? What was the order of voting?

Solution: The order of voting was C,E,A,B, and D.

- (d) Show how to determine the location of the $(\frac{t-1}{2})$ th vote cast, given the final location of t ballots. You may assume that t is odd. (Hint: Consider adding up the indices of the filled memory slots.)

Solution: Use the same idea as in part (a):

$$\begin{aligned} \sum_{i=0}^{t-1} ai + b &= tb + \sum_{i=0}^{t-1} ia \\ &= tb + t \left(\frac{t-1}{2} \right) a \\ \frac{\left(\sum_{i=0}^{t-1} ai + b \right)}{t} &= \left(\frac{t-1}{2} \right) a + b \end{aligned}$$

- (e) For some values of t , there are multiple a and b values which could lead to a given final placement of ballots. However, in other cases a final placement of ballots will uniquely specify a and b . Find out which values of t *unambiguously* define a and b values. Show how to find a and b in those cases.

You don't need to perform a brute force search. Given a and $(\frac{t-1}{2}) * a + b$, can numerically solve for b modulo p . (Hint: Consider squares of indices of filled memory slots.)

Solution: We have two unknowns and will use summations, as in part (B), to find to known formulas. One is already known: $\sum_{i=0}^{t-1} ai + b = a \binom{t}{2} + bt$. This is equivalent to:

$$\left(\sum_{i=0}^{t-1} ai + b \right)^2 = a^2 \binom{t}{2}^2 + 2ab \binom{t}{2} t + b^2 t^2$$

We will find the second formula as follows:

$$\begin{aligned}
 \sum_{i=0}^{t-1} (ai + b)^2 &= \sum_{i=0}^{t-1} (a^2 i^2 + 2abi + b^2) \\
 &= a^2 \binom{t}{2} \left(\frac{2t-1}{3} \right) + 2ab \binom{t}{2} + b^2 t \\
 \left(\sum_{i=0}^{t-1} (ai + b)^2 \right) t &= a^2 \binom{t}{2} \left(\frac{2t-1}{3} \right) t + 2ab \binom{t}{2} t + b^2 t^2 \\
 \left(\sum_{i=0}^{t-1} (ai + b)^2 \right) t - \left(\sum_{i=0}^{t-1} ai + b \right)^2 &= a^2 \binom{t}{2} \left(\frac{2t-1}{3} \right) t - a^2 \binom{t}{2}^2
 \end{aligned}$$

Solving this for a^2 , we get:

$$a^2 = \frac{12}{t^2(t^2 - 1)} \left[\left(\sum_{i=0}^{t-1} (ai + b)^2 \right) t - \left(\sum_{i=0}^{t-1} ai + b \right)^2 \right]$$

Since $a \leq \frac{p-1}{2}$, there is a unique solution for a for any $t > 1$. This fails if $t = 0$ or $t = \pm 1$. Intuitively, this makes sense. If $t = 0$, nothing has been hashed so there is no way to derive a . The same applies if $t = 1$, since only b has been filled. If $t = -1 = (p-1)$, then every slot has been filled except one. This is the dual of the case where only one item has been placed.

Once we have solved for a , we can find b by iterating backward from any filled ballot until we hit an empty space.

Problem 3-4. TSA Database [Group]

The Tasmanian Security Administration wants to keep a database of all people on their “no-fly” list. They wish to keep the list itself secret from casual inquiries, but want airline security personnel to be able to query the list to find out if a particular individual is on it.

However, they are concerned about an attacker spoofing the database to the security personnel. Thus, they want to publish some information in a publicly available manner (e.g. on their website, on distributed fliers, etc.) so that they can then prove (based on the public information) whether or not a name is indeed in the database. They decide to use hashing to accomplish this.

Solutions to this problem are thanks to Tawanda Sibanda, Akshay Patil, Tony Lim, and Kathryn Shih.

- (a) Suppose they decide that, for each person on the list, they will publish $h(ID)$, where h is some hash function and ID is some unique identifier for that person (for example, their name and date of birth). What properties must h have in order for the system to prevent casual browsing and not implicate the wrong people?

Solution: To prevent casual browsing, h must be one-way. To prevent the list from implicating the wrong people, it should have weak collision resistance for all of the names on the list.

- (b) Why does this system not provide adequate secrecy of the list?

Solution: The list is not adequately secret because its easy for anyone with access to the public hash values and individual IDs to see if an individual is on the list.

- (c) In addition to being insecure, the TSA decides that publishing a large number of hash values is inefficient from a practical perspective, and would prefer to publish just one hash value. Their next attempt is to simply publish a hash of the entire list of names.

Explain what is wrong with this system.

Solution: If the TSA wants to prove to airport personnel that a name is or isn't on this list, the only way for them to do it is to release the entire list (in plaintext) so that the hash can be computed and verified. Transmission impracticalities aside, presenting the list plaintext on every query won't protect the privacy of the list for very long.

- (d) In order to come up with a viable solution, the TSA hires Alyssa P. Hacker, an MIT graduate, as a consultant. Alyssa quickly realizes that any solution should, on each query, have the database reveal *some* information to the airport security personnel (otherwise anyone could determine the contents of the database for themselves), but should not reveal any *secret* information beyond what was specifically queried (to protect privacy and national security). Alyssa sketches a solution in which each ID is concatenated with a long random value, then hashed using a suitably strong hash function. The TSA then publishes a hash of the concatenation of those hash values. Upon each (properly authenticated) query, the airport security employee receives back the random value associated with the queried ID, as well as the list of hashes. The computer at the airport then computes two hashes: one to verify that the ID and randomness indeed hash to a value in the list, and one to verify that the entire list hashes to the public value.

Argue that this scheme protects the secrecy of the list and does not implicate anyone not on the list.

Solution: In order to implicate someone not on the list, an adversary would have to find a random value for the innocent person that hashed to the same value as the name and randomness of a list member. This is equivalent to finding a collision in the hash, and should be prohibitively difficult if the hash is suitable strong.

In order to protect the secrecy of the list, persons with access to the public hash should be unable to retrieve meaningful information about the list. Intuitively, this holds; the entire list is being shortened to one hash value, and too much information will be lost for anything meaningful to be extracted. Additionally, a suitable strong hash should be one-way, which means it will be impossible to retrieve the list of hash values from it. Also, the list should be safe against airport personnel retrieving all of the names on it, but this scheme protects

against that; they never receive the plaintext of the list, and onewayness prevents them from retrieving it from the list of hash values.

- (e) What functional requirement is *not* met by the scheme in part (d)?

Solution: It is impossible to prove that a name isn't on the list. For that matter, it's virtually impossible to prove that a name doesn't correspond to a single hash value; if the random values are large compared to the size of the hash output, it is likely that one of them will make the name hash to the value (but since there's exponentially many possible random values, it's computationally prohibitive to check them). Alternately, if the random values are small enough that it is practical to check them all, they no longer protect the ID's from a dictionary attack. The only way to prove that this wasn't the pair the TSA intended is to release the ID and random value that they originally used to generate the hash value, which fails to protect secrecy.

- (f) In addition to the problem in part (e), the TSA has N people on the no-fly list (N being a rather large number), and does not have the bandwidth to send N hash values each time a query is made of the database. Undaunted, Alyssa realizes that while bandwidth is expensive, computation is cheap. She comes up with a new design for the database in which each ID is hashed (without additional randomness), and then the N hashed values form the leaves of a binary tree. (You may assume that N is an exact power of 2.)

Explain how Alyssa's idea works. Include the number of values the database needs to send per query (and which ones), the number of hashes the querying computer needs to perform in order to verify that an ID is in the database, and what (single) value is published initially.

Solution: As already stated, the data structure is a binary tree with the hash values of the IDs on its leaves. For every non-leaf node, define its value to be the hash of the concatenation of its children's values: that is, $V_p = h(V_{lc}||V_{rc})$, where V_p is the parent value and V_{lc}, V_{rc} are respectively the left and right child values.

The public value should simply be the top node for the entire tree. To verify a (leaf) value, the TSA must transmit the hash values that were used to generate the values for every node along the branch from the top of the tree to the leaf in question. That is, going up the branch from the leaf to the top, the TSA must send the numbers that created the value of the next parent node. Since the verifying client will always be able to compute one child value from the information it has already received, the TSA will have to send one extra hash value per level of the tree. In addition to the hash values, the TSA will have to transmit a single bit on each level to indicate which of the two values – the one being transmitted, or the one the client was already able to compute – is the left child and which is the right child. This will be a total of $\log_2 N$ hash values, and the client will have to compute the same number of hashes. (Presumably the client will hash the ID to ensure that it is what the TSA claims.)

This scheme is secure because if an adversary can somehow manipulate an ID to follow a path to the top node that was not in the original tree, the path will have to converge with one that was in the tree - it's a series of hashes of different values that eventually equal the same thing. This is equivalent to finding a collision in the hash function.

- (g) The TSA receives a request from Ken Teddedy that if an ID is *not* on the no-fly list, the database should unambiguously prove this fact to the airline security employee so that there are no misunderstandings. Alyssa points out that, if the database she proposed is modified slightly, one can show that an ID is *not* in the list. How is this done, and how many hash values does the database need to send in this case?

Solution: The data structure needs to be changed so that the leafs are sorted by hash value from left to right. Formally, this means that for all nodes, all of the values of the leafs on the left child's branch will be less than all of the leaf values on the right child's branch.

Once the leafs are sorted like this, you know that if a hash value exists in the tree, it will exist between the largest hash less than it and the smallest hash greater than it. To prove that a leaf is not in the tree, you simply need to prove that there is an adjacent set of hash values – one greater than it, and one smaller.

So to verify that a name isn't on the list, TSA then provides the two adjacent leaves along with their verification paths; the receiver then checks that the name in question hashes to a value between the two and that the verification paths are correct. It can also check that the two leaves are indeed adjacent by checking the bits on the verification paths – when walking down the tree, after the two paths diverge the lesser leaf's path should always branch right, while the greater leaf's path should always branch left. (Assuming that we sort the leaves left to right in increasing order)

In the worst case scenario, the two paths will diverge immediately from the top, public node and the TSA will have to send information for two entirely separate branches. This means it will require $2 \cdot \log_2 N$ hash values to be transmitted, and it will require the same number of hashes to be computed for verification. Note that this is an upper bound; in some cases the adjacent hash values will share the same parent.

If you wish, to ensure that the TSA honestly sorts the leafs, you can make a series of queries for random IDs that are and aren't in the database in the database and ensure that the results are consistent with an honest sorting. While this won't catch all potential TSA dishonesty, it will make it much harder.