Fall 2019 CIS 3362 Homework #6 Solutions

The key to solving this problem is reading Knapsack.java and realizing that the first value created in the super-increasing set (private key) is in between 2 and 10, inclusive. The second value in the set could be as low as 3 or as high as 30. These numbers are both relatively small. What occurs with them is that they get multiplied by a secret key w, and then modded by the public key u. Let the private set start with the values a[0] and a[1]. Let the corresponding public set values be b[0] and b[1]. This gives us the following equations:

$w(a[0]) \equiv b[0] \bmod u$
$w(a[1]) \equiv b[1] \bmod u$

While we don't know w and w could be a large range of values, a[0] and a[1] can only be a few things, as previous mentioned. Though it's not 100% certain, it's possible that gcd(a[0], u) = 1. Same goes for gcd(a[1], u) = 1. If either of these is true, then we could either multiply the first equation through by $a[0]^{-1} \bmod u$ or the second equation through by $a[1]^{-1} \bmod u$. Since there are so possible values for a[0] or a[1], what we could do is just "guess" each of them, and see if the guess leads to a potential result.

First of all, we must skip guessing a value for a[0] where $\gcd(a[0], u) \neq 1$. Let g = our guess for a[0]. For cases where gcd(g, u) = 1, calculate $g^{-1} \bmod u$ and multiply the equation through by this to yield:

$g^{-1}(w)(g) \equiv g^{-1}(b[0]) \bmod u$

This simplifies to

$w \equiv g^{-1}(b[0]) \bmod u$

This is a potential guess for w! Collect all of these potential guesses by trying all possible guesses for a[0] and a[1]. (If necessary, you can move to a[2], but for this specific key, that wasn't necessary as one of the two, a[1], doesn't share a common factor with u.)

Once we have collected a set of potential guesses for w, we need a way to check if they are possible or not. If we have the public set, which is created by multiplying values in the private set by w, we can take the public set values and multiply them by w^{-1}. If our guess for w is correct, after this multiplication, we should receive the private set. How do we "know" if our private set is correct? Well, we can't know for sure, but it's pretty easy for us to check if our guessed private set is super-increasing. So, what we do is, the following:

1. For each guess w:
    - 1a. calculate gcd(w, u). If this is not 1, skip this guess.
    - 1b. Otherwise, calculate $w^{-1} \bmod u$.
    - 1c. Then for each b[i], calculate $w^{-1}b[i] \bmod u$.
    - 1d. For this new set, check if it's super-increasing or not. If it is, try this as the private key!

My code that does all of these tasks is included in the file breakknapsack.java.

In my code, I only try values 2 through 9 for a[0] and a[1]. In doing so, I was able to uncover the correct private key. Here is a brief outline of what I am doing on which lines of code:

Lines 23 to 30: Trying different values of a[0], a[1] and using these to create a potential list of w's which I store in wList.

Lines 34 to 40: Trying each potential w and saving the last one that works. (I tweaked the code a bit until the multby was set exactly once.

Lines 94 to 119: This is the method that tests if a particular w value creates a super-increasing list. What I do is first check if my potential w is relatively prime to my mod value u. If not, I automatically return false. If it is, I calculate w^{-1}. (In my code I call it x, and xinv, respectively.) Then, I go through each number in the public list and multiply it by xinv and mod it by mod. This would potentially be the next value in the private list. I check to see if the current running sum of all previous values is less than this. If it's not, I return false. If it is, I just add it to my sum and continue. If we get through the whole set, it's super increasing, so we just return true at the conclusion of the loop.

Once you get the private key, w, what you have to do to decrypt is the following:

1. Take each ciphertext, c, and multiply it by w^{-1} mod u, and then mod this product by u. Call this value t, the target.

2. t is the value we want to sum to, with our private set (the super increasing one) that we previously uncovered.

3. To determine the binary representation of the plaintext, loop from i = 127 downto 0, since the set size is 128, looking to see if a[i] <= t to or not. If it is, the i^{th} bit is 1. If it isn't the i^{th} bit is 0. For each group of eight bits, calculate the binary value and store its corresponding character. (This part is easy - just cast to a char in most languages and it just works.)

In my solution, my loop structure is i=15 downto 0, where i represents the byte I am calculating. (Line 68) Inside of this loop I have another loop for j=7 downto 0 (line 72), which is going through the bits of that particular byte. The actual index into the private set is 8*i+j. Notice, that this formula inside the double loop structure will count down 127, 126, 125, …, all the way to 0. For the way I coded it, when j is 7, the bit value is 1, when j is 6 the bit value is 2, when j is 5, the bit value is 4, etc. In code, given j, the bit value is (1 << (7 - j)). Notice that as j counts down, the bit values go up by powers of 2. << is a left shift, which precisely calculates powers of 2. The exponent to 2 we desire based on the loop structure is 7 - j. I'm accumulating the byte value in this manner on line 78. I add the current character to the beginning of the accumulating string on line 86.

If you run my code with the file breakknapsack.txt, which has the public keys, then an integer indicating the # of blocks of ciphertext, followed by those blocks, it'll spit out the decrypted message.