Anish Banerjee, Shankh Gupta **Problem Set - 4 b**
COL759: Cryptography November 2023

Problem 1: RSA with a low-entropy prime generator

Solution: Here, the main idea is that if the prime generator has a low-entropy, it is very likely that two of the sampled N will have a common factor. Since the gcd of two numbers can be calculated efficiently, we can easily factorize the number and thus break RSA.

```
def attack():
    L=[] # A list for storing N
    for i in range(200):
        (N, e), ct = restart_system()
        for i in L:
            p=gcd(i,N)
            if p!=1 and p!=N and p!=N:
                q=N//p
                phiN=(p-1)*(q-1)
                d = inverse(e, phiN)
                return dec(ct, N, d)
        L.append(N)

```

Listing 1: RSA with a low-entropy prime generator

Problem 2: Another Attack on RSA Signatures

Solution: The attack follows in three steps :

1. Choose a string s so that it's cube doesn't exceed `0x00 0x01 0x00 ... 0x00` by taking cube root.
2. Set the last $|M| + 1$ bytes (here $|M|$ denotes size of message in bytes) so that the cube of s (say t) has the last $1 + |M|$ bytes as `0x00`$[M]$. To do this, we compare t (cube of s) and our target suffix (`0x00`$[M]$) until we find a mismatch at some position (say i), upon which we flip the bit at position i in s. This will not alter the previously matching bits and the current bit in t will match the one in our target suffix. We do this until whole of the target suffix is matched in t.
3. To ensure that the cube of s has no zero bytes in the middle part, we sample random bits (in a range so that t doesn't exceed `0x00 0x02 0x00 ... 0x00`) until the point when there are no zero bytes left.

Problem 3: Attack on RSA PKCS Padding: Bleichenbacher's attack

Solution: Here we implemented Bleichenbacher's attack as mentioned in his paper. We also referred to this video for understanding the attack better.

```
def attack(cipher_text, N, e):
    ct_bin = ''.join(format(byte, '08b') for byte in cipher_text)
    k = len(cipher_text)     # number of bytes in the cipher text
    B = pow(2, 8*(k-2))      # bound
    M = [(2*B, 3*B-1)]       # set of intervals
    i = 1                    # number of successful s values found
    # Implement ceil and floor functions as math library functions don't work for large integers
    s = ceil(N, (3 * B))
    while True:
        min_n = M[0][0]
        max_n = M[0][1]
```

```
        for r in M: # min_n, max_n based on previous step
            if r[0] < min_n:
                min_n = r[0]
            if r[1] > max_n:
                max_n = r[1]
        if (i > 1 and len(M) == 1):      # Step 2.c from the paper
            a, b = M[0]
            r = floor(2*(b*s - 2*B), N)
            counter = 1
            while True:
                s = ceil((2*B + r * N), b)
                s_max = ceil((3*B + r * N), a)
                found = False
                while s <= s_max:
                    ct_mod = (pow(pow(s, e) * (bin_to_int(ct_bin)), 1, N))
                    if (rsa.check_padding(rsa.num_to_bytes(ct_mod))) :
                        found = True
                        break
                    s += 1
                if found:
                    break
                else:
                    if counter%1000 == 0: print(counter)
                    counter += 1
                r += 1
        else:        # Step 2.a,b from the paper
            while True:
                s += 1
                ct_mod = (pow(pow(s, e) * (bin_to_int(ct_bin)), 1, N))
                if (rsa.check_padding(rsa.num_to_bytes(ct_mod))) :
                    break

        # update the set M (Step 3 from the paper)
        M_new = []

        for m in M:
            a, b = m
            # Compute large values one time
            r_min = (a*s - 3*B + 1) // N
            r_max = (b*s - 2*B) // N

            r = r_min
            while r <= r_max:
                a_new = max(a, ceil((2*B + r*N), s))
                b_new = min(b, floor((3*B - 1 + r*N), s))
                if a_new > b or b_new < a:
                    r += 1
                    continue

                if a_new == b_new: # Answer found
                    crackList = list(rsa.num_to_bytes(a_new))
                    final_msg = []
                    zeroFound = False
                    for c in crackList[2:]:
                        if(zeroFound):
                            final_msg.append(c)
                        if(c == 0 and not zeroFound): zeroFound = True
                    return final_msg
                M_new.append((a_new, b_new))
                r += 1
        M = M_new
        i += 1
```

Listing 2: Bleichenbacher's attack