Modern Cryptology - Homework 2

Yosef Goren

June 15, 2022

In this homework I will be denoting $\mathbb{P}$ to mean the set of primes. and denoting $a =_N b$ to mean $a = b(mod\ N)$.

1 Public-key Encryption from QR

1.1 QR given factorization

For any $X \in \mathbb{N}$ denote: $QR(X)$ as the set of quadratic residues modulus X.
Lemma I;

$$\forall P, Q \in \mathbb{P} : QR(N) = QR(P) \cap QR(Q)$$

Proof Lemma I:
Let $a \in QR(N)$.

$$\exists x, K : x^2 \underset{N}{=} a \Rightarrow x^2 = a + (K)PQ$$

$$\Rightarrow x^2 = a + (KP)Q \wedge x^2 = a + (KQ)P$$

$$\Rightarrow x^2 \underset{Q}{=} a \wedge x^2 \underset{P}{=} a$$

$$\Rightarrow a \in QR(Q) \wedge a \in QR(P) \Rightarrow a \in QR(Q) \cap QR(P)$$

Let $a \in QR(P) \cap QR(Q)$. Thus:

$$\exists x_1, x_2 : a \underset{P}{=} x_1 \wedge a \underset{Q}{=} x_2$$

Thanks to the Chinese remainder theorem we know there exists a solution x which satisfies:

$$x \underset{P}{=} x_1, x \underset{Q}{=} x_2$$

Thus:

$$\Rightarrow a \underset{P}{=} x^2, a \underset{Q}{=} x^2 \Rightarrow a \underset{PQ}{=} x^2 \Rightarrow a \in QR(N)$$

Now we use the correctness of the Lemma I to define a polynomial algorithm:

```
def qr(a: int, P: int, Q: int)->bool:
  """the algorithm determines if 'x' is
    quadratic residue of under modulus PQ"""
  a_is_P_qr = a**((P-1)/2)%P == 1
  a_is_Q_qr = a**((Q-1)/2)%Q == 1
  return a_is_P_qr and a_is_Q_qr
```

Indeed the names of the variables at lines 4 and 5 are informative (and correct) due to the properties of Euiler's criterion as seen in class, meaining that a is qr modulus P iff $a^{(P-1)/2} =_P 1$, and same with Q.
This together with Lemma I proves the correctness of this algorithm.

We have seen in the last homework how modulus exponantiation can be done efficiently; which makes this algorithm polynomial.

1.2 Generating QR

In the following, all expressions and operations are in the $\mathbb{Z}_N^*$ group unless said otherwise.
Let $x \in QNR(N)$.

Lemma I; $x, z \in QNR(N) \Rightarrow zx^{-1} \in QR(N)$:
Since $QNR(N) = QNR(P) \cap QNR(Q)$ we get:

$$(zx^{-1})^{\frac{p-1}{2}} \underset{P}{=} (z)^{\frac{p-1}{2}} (x^{-1})^{\frac{p-1}{2}} \underset{P}{=} (z)^{\frac{p-1}{2}} (x^{\frac{p-1}{2}})^{-1} \underset{P}{=} (-1)(-1)^{-1} = 1$$

and in the same way under modulus Q we get $(zx^{-1})^{\frac{p-1}{2}} =_Q 1$. From Euile's criterion we get that $(zx^{-1})^{\frac{p-1}{2}} \in QR(N)$.

Lemma II; $\{y^2 x : y \in \mathbb{Z}_N^*\} \supseteq QNR(N)$:
Let $z \in QNR(N)$, from Lemma I we get $x^{-1}z \in QR(N)$ Thus:

$$\exists y : y^2 = zx^{-1} \Rightarrow y^2 x = z \Rightarrow z \in \{y^2 x : y \in \mathbb{Z}_N^*\}$$

Lemma III; $\{y^2 x : y \in \mathbb{Z}_N^*\} \subseteq QNR(N)$:
Let $y \in \mathbb{Z}_N^*$. Assume $\exists z : z^2 = y^2 x$. Thus:

$$z^2 y^{-2} = x \Rightarrow (zy^{-1})^2 = x \Rightarrow x \notin QNR(N)$$

Hence the assumption is incorrect, and $y^2 x \in QR(N)$.

Proof:
Define $g(z) = z \cdot x$, $g : QR(N) \longrightarrow QNR(N)$.
From Lemma II and III, we get that g's image is exactly $QNR(N)$.
g is invertible and thus is a bijection.
Let:

$$a^2 \in QR(N),\ y \overset{\$}{\sim} \mathbb{Z}_N^*$$

Since y^2 has four different roots:

$$\Pr_{y \leftarrow \mathbb{Z}_N^*}[y^2 = a^2] = \frac{4}{|\mathbb{Z}_N^*|} = \frac{1}{\frac{|\mathbb{Z}_N^*|}{4}} = \frac{1}{|QR(N)|}$$

$$\Rightarrow y^2 \overset{\$}{\sim} QR(N)$$

And since g is bijection $QR(N) \to QNR(N)$:

$$g(y^2) = y^2 x \overset{\$}{\sim} QNR(N)$$

$$\square$$

2 Statistically Hiding Commitments

2.1 Inner Product with Random String

Let:

$$< a, b >= (\sum_i a_i b_i)\%2$$

$$L = \{0, 1\}^n$$

Proof: Let $b \in L \setminus \{0^n\}$. Let j be the first non-zero index of b. Define:

$$f(a) = a_1 a_2 ... \bar{a_j} .. a_n$$

Lemma I; $< a, b > =_2 < f(a), b > +1$:

$$< a, b > \underset{2}{=} \sum_i a_i b_i \underset{2}{=} \sum_{i \neq j} a_i b_i + a_j b_j \underset{2}{=} \sum_{i \neq j} a_i b_i + a_j$$

$$\underset{2}{=} \sum_{i \neq j} a_i b_i + \bar{a_j} + 1 \underset{2}{=} \sum_i f(a)_i b_i + 1 < f(a), b > +1$$

Lemma II; f is bijection $\{a :< a, b > =_2 1\} \longleftrightarrow \{a :< a, b > =_2 0\}$:

$$x \in \{a :< a, b > \underset{2}{=} 0\} \Leftrightarrow < x, b > \underset{2}{=} 0 \Leftrightarrow < f(x), b > +1 \underset{2}{=} 0$$

$$\Leftrightarrow < f(x), b > \underset{2}{=} 1 \Leftrightarrow f(x) \in \{a :< a, b > \underset{2}{=} 1\}$$

Thus f is bijection.
Proof using the Lemma II: Let:

$$a = |\{a :< a, b > \underset{2}{=} 1\}|, b = |\{a :< a, b > \underset{2}{=} 0\}|$$

From Lemma II: $a = b$.
In addition one of the two cases must always be correct, hence: $a + b = 1$.
Solving these two equations gives $a = b = \frac{1}{2}$.
By definition:

$$a = \Pr_{a \leftarrow L}[< a, b >= \underset{2}{=} 0], b = \Pr_{a \leftarrow L}[< a, b >= \underset{2}{=} 1]$$

meaning they are both $\frac{1}{2}$.

2.2 Inner Product is Universal

Denote: $H = \{h_a : a \in L\}$.
Let $a \in L$. We want to show that:

$$\forall x, y \in L : x \neq y \Pr_{h \leftarrow H}[h(x) = h(y)] \leq \frac{1}{2}$$

Proof:
Let $x, y \in L : x \neq y$.

$$\Pr_{h \leftarrow H}[h(x) = h(y)] = \Pr_{a \leftarrow L}[h_a(x) = h_a(y)]$$

$$= \Pr_{a \leftarrow L}[h_a(x) = h_a(y)] = \Pr_{a \leftarrow L}[< a, x >=< a, y >]$$

$$= \sum_{b \in \{0,1\}} \Pr_{a \leftarrow L}[< a, x >=< a, y >:< a, x >= b] \cdot \Pr[< a, x >= b]$$

$$= \sum_{b \in \{0,1\}} \Pr_{a \leftarrow L}[< a, x >=< a, y >] \cdot \frac{1}{2} = \sum_{b \in \{0,1\}} \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$$

Since $\frac{1}{2}$ is the size of the output space, this indeed shows that H is universal hash function family.

2.3 Purifying Randomness

3 Is Factoring NP complete?

3.1 Equivalence to Factoring

We show a there exists a polynomial turing machine for deciding L iff there exists a polynomial factoring TM by showing:

1. Polynomial TM for factoring which uses a polynomial TM for deciding L.
2. Polynomial TM for deciding L which uses a polynomial TM for factoring.

Armed with the church turing thesis; we describe these two turing machines as code of a contemporary programming language.

1. Factoring given decision:

```python
def factor(N: int, decide_oracle)->tuple:
  """given a number 'N',
    and given a 'decide_oracle' function which decides 'L',
    return the list of prime factors of 'N'.
    The implementations is a loop of binary searches."""
  prime_factors = []
  while N > 1:
    #find the largest prime factor of current 'N':
    left, right = 1, N
    while left < right-1:
```

```
      M = (left+right)//2
      if decide_oracle(N, M):
        left = M
      else:
        right = M
    largest_prime_factor = right
    #add it to list and update 'N':
    prime_factors.append(largest_prime_factor)
    N = N//largest_prime_factor
  return prime_factors
```

Indeed this implementation does a polynomial (w.r to #bits) number of calls to the oracle, and executes only a polynomial number of operations itself:
note how each iteration of the binary search algorithm is polynomial with the number of bits of N since it is logarithmic with the size of the search space. In addition, there can only be a polynomial (w.r to #bits) number of factors since each one is at-least 2; this means there are only a polynomial number of iterations in the main loop.

2. Decision given factoring:

```
def decide(N: int, M: int, factoring_oracle)->bool:
  """returns 'True' if (N,M) is in L, and False otherwise."""
  prime_factors = factoring_oracle(N)
  return max(prime_factors) > M
```

3.2 coNP

Here we show $L \in NP \cap coNP$ in two parts:

- $L = L_a \in NP$.
- $\bar{L} = L_b \in NP$.

To show each of these we will define a relation R which will satisfy all conditions:

1. $\forall(x, y) \in R, |y| = Poly(|x|)$
2. $(x, y) \in R \Leftrightarrow x \in L$
3. $\exists M_R,$ NTM which decides R and is polynomial.

- Define R_a for L_a:

$$R_a = \{((N, M), p) : p \in \mathbb{P} \wedge p > M \wedge N\%p = 0\}$$

It is easy to see that if $(N, M), p \in R_a$ then since p is a prime factor of N which is larger than N - it means that $(N, M) \in \mathbb{P}$. In addition, $\#bits(p) \leq \#bits(N)$ thus the length is linear (and polynomial).
As for the last condition, we define a NTM M_a which will decide R_a:
Given some input M_a will simply check for the three conditions for the

pair of inputs to be contained within R_a. Checking for the first conditon in polynomial time is far from trivial, nevertheless it is widely known that $PRIME \in \mathbb{P}$, and thus it is possible to check if $p \in \mathbb{P}$ in polynomial time; ofcourse it is possible for the other two conditions also.

- Define R_b for L_b:

  $$\{((N, M), S) : \prod_{p \in S} p = N \wedge S \subseteq \mathbb{P} \wedge max(S) \leq M\}$$

  Note how S is the set of prime factors of N.

  We define a NTM to decide R_b:
  Given an input $((N, M), S)$ - the machine will check each condition for being contained in R_b and accept iff all three are statisfied; taking the product of a set of numbers can comparing it can be done in polynomial time, same as taking the maximal value in a list. As before, we know that checking if a number is prime can be done in polynomial time, and there can only be a polynomial number of elements in the input to begin with (w.r to the size of the input...).

  Additionally, it is worth noting how in this case too the length of S will be polynomially bound to the length of N (hence to the length of (N, M) too): This is because the number of prime factors of N is $O(log(N))$, and each of them has length of $O(log(N))$.

To sum up, we have shown that $L \in NP$, and that $L_b = \bar{L} \in NP$ hence $L \in coNP$, meaining $L \in NP \cap coNP$.