DataPrivacy—hw3

Terence Wang

2023/01/20

Contents

1 Q1

1.1 a

Table 1: the inverse permutation π^{-1}

x	1	2	3	4	5	6	7	8
$\pi^{-1}(x)$	2	4	6	1	8	3	5	7

1.2 b

ciphertext:
TGEEMNEL NNTDROEO AAHDOETC SHAEIRLM
plaintext:
GENTLEME NDONOTRE ADEACHOT HERSMAIL

So the result:**GENTLEMENDONOTREADEACHOTHERSMAIL**.
(GENTLEMEN DONOT READ EACH OTHERS MAIL)

2 Q2

2.1 a

Use Shannon's Theorem:

Since the key is chosen uniformly at random, **the Latin square cipher** satisfies **1.Every key i∈K is chosen with equal probability**$\frac{1}{|K|}$.

Therefore, we just need to prove that **the Latin square cipher** satisfies **2.For every j∈M and every L(i,j)∈C, there exists a single key such that i∈K outputs L(i,j)**.

According to the definition of **n-th order Latin square**: each of the n intergers $1, 2, \cdots, n$ occurs exactly once in each row and each column of L, we can conclude that for a given j∈M and a given L(i,j)∈C, there exists a single corresponding key i∈K.Thus, **the Latin square cipher** satisfies **2.For every j∈M and every L(i,j)∈C, there exists a single key such that i∈K outputs L(i,j)**.

2.2 b

Let us assume that $|M| = |C| = |K| = n$, $M = \{m_1, m_2, \cdots, m_n\}$, $C = \{c_1, c_2, \cdots, c_n\}$, $K = \{k_1, k_2, \cdots, k_n\}$. Since the cipher has perfect secrecy, we can conclude:

1. Every key k∈K is chosen with equal probability $\frac{1}{|K|} = \frac{1}{n}$.
2. For every m∈M and every c∈C, there exists a single key k∈K such that $e_k(m) = c$.(This property means: for every m and k, there exists a single corresponding c; for every c and k, there exists a single corresponding m)

Every k∈K appears exactly once for a given c∈C and every m∈M, such that $e_k(m) = c$. Otherwise, if there exists m_1 and m_2 use the same key k to encrypt and get c, then c can not be decrypted correctly. Since $|M| = |C| = |K| = n$, for a given c, different k is needed to encrypt every m. Therefore, all of the $k \in K$ are used and each is used exactly once.
The number of $c \in C$ is n. For a given m, all of the $k \in K$(**number: n**) is needed to get different $c \in C$(**number:n**). Otherwise, there exist k_1 and k_2 such that $e_{k_1}(m) = e_{k_2}(m) = c$. In this circumstance, we can not get all of the $c \in C$(**i.e. the number of c is less than n**). This contradicts to the definition of **perfect secrecy**.
Therefore, m, k, c correspond with each other exactly. So $\forall c \in C$, $Pr = \sum_{i \in M} P(m = i) \times P(e_k(m) = c) = n \times \frac{1}{n} \times \frac{1}{n} = \frac{1}{n}$. We can conclude that each ciphertext is equiprobable.

3 Q3

3.1 a

$n = p \times q = 11413$, $\phi(n) = (p-1) \times (q-1) = 11200$. Since $gcd(pub, \phi(n)) = gcd(pub, 11200) = 1$, we can conclude that **3839** public keys can be chosen.
The code used to calculate the number of public keys is as follows:

```c
#include <stdio.h>
int gcd(int a, int b)
{
    return b == 0 ? a : gcd(b, a % b);
}
int main()
{
    int cnt = 0;
    for (int i = 2; i < 11200; i++)
    {
        if (gcd(i, 11200) == 1)
        {
            cnt++;
        }
    }
    printf("%d\n", cnt);
    return 0;
}
```

3.2 b

$e = 3533$, $n = 11413$, $M = 9726$ then we can get $c = M^e \mod n = 9726^{3533} \mod 11413 = 5761$
The code used to calculate the ciphertext is as follows:

```c
#include <stdio.h>
```

```
#define MAX 11413
#define E 3533
const long long int N = 9726;
int main()
{
    long long int result = N;
    for (int i = 1; i < E; i++)
    {
        result = result * N % MAX;
    }
    printf("%lld\n", result);
    return 0;
}
```

Since $e \times d \equiv 1 \mod \phi(n)$, we have $3533 \times d \equiv 1 \mod 11200$. Thus we can get $d = 6597$

The code used to calculate d is as follows:

```
#include <stdio.h>
#define MAX 11200
#define E 3533
int main()
{
    long long int d = 1;
    while (d * E % MAX != 1)
        d++;
    printf("%lld\n", d);
    return 0;
}
```

$M = c^d \mod n = 5761^{6597} \mod 11413 = 9726$

The code used to calculate M is as follows:

```
#include <stdio.h>
#define MAX 11413
#define D 6597
const long long int c = 5761;
int main()
{
    long long int result = c;
```

```
    for (int i = 1; i < D; i++)
    {
        result = (result * c) % MAX;
    }
    printf("%lld\n", result);
    return 0;
}
```

3.3 c

Since p and q are prime numbers, we can get $\phi(n) = (p-1) \times (q-1) = p \times q - p - q + 1 = n - p - q + 1$. $\rightarrow \begin{cases} p \times q = n \\ n - p - q + 1 = \phi(n) \end{cases} \rightarrow \begin{cases} p \times q = n \\ p + q = n + 1 - \phi(n) \end{cases}$

Without loss of generality, let us assume that $p > q$, then we can get $p = \frac{n+1-\phi(n)+\sqrt{(n+1-\phi(n))^2-4n}}{2}$, $q = \frac{n+1-\phi(n)-\sqrt{(n+1-\phi(n))^2-4n}}{2}$.

Therefore, we can compute p and q in polynomial time.

4 Q4

4.1 a

$n = p \times q = 187$, $\lambda = lcm(p-1, q-1) = 80$, $g = n + 1 = 188$. $r = 83$, $m = 175$, $c = g^m \times r^n \mod n^2 = (n+1)^m \times r^n \mod n^2$. Therefore, $c = 23911$.

The code used to calculate c is as follows:

```
#include <stdio.h>
#define N 187
#define M 175
#define R 83
const long long int mod = N * N;
int main()
{
```

```
    long long int n = N;
    long long int r = R;
    long long int result = n + 1;
    for (int i = 1; i < M; i++)
    {
        result = (result * (n + 1)) % mod;
    }
    for (int i = 1; i <= N; i++)
    {
        result = (result * r) % mod;
    }
    printf("%lld\n", result);
    return 0;
}
```

Proof of the Homomorphic addition property:

$$\begin{aligned} Decrypt((c_1 \cdot c_2) \mod n^2) &= Decrypt((g^{m_1} \cdot r_1^n \cdot g^{m_2} \cdot r_2^n) \mod n^2) \\ &= Decrypt((g^{m_1+m_2} \cdot (r_1 \cdot r_2)^n) \mod n^2) \\ &= m_1 + m_2 \end{aligned}$$

4.2 b

Let $z_i = x_i \oplus y_i$, $c_i = a_i \oplus b_i$. Since $x_1 \oplus x_2 \oplus x_3 = 0$ and $y_1 \oplus y_2 \oplus y_3 = 0$, we can get $z_1 \oplus z_2 \oplus z_3 = 0$. Observe that for every $i \in \{1, 2, 3\}$ it holds that $c_i = z_{i-1} \oplus (v_1 \oplus v_2)$, where $i - 1$ denotes 3 when $i = 1$. e.g. we have $c_1 = a_1 \oplus b_1 = x_3 \oplus v_1 \oplus y_3 \oplus v_2 = (x_3 \oplus y_3) \oplus (v_1 \oplus v_2) = z_3 \oplus (v_1 \oplus v_2)$. Thus each P_i locally computes (z_i, c_i) and no communication is needed in order to compute a secret sharing of $v_1 \oplus v_2$.

5 Q5

5.1 a

Interchangeable means that the two libraries have the same effect on **all calling programs**, while **Indistinguishable** means that the two libraries are distinguishable if **all polynomial-time**

calling programs have negligible advantage in distinguishing them.
Interchangeable: Let $\mathcal{L}_1$, $\mathcal{L}_2$ be two libraries with a common interface. $\mathcal{L}_1 \equiv \mathcal{L}_2$ if for all programs $\mathcal{A}$ that output a single bit, $\Pr[\mathcal{A} \diamond \mathcal{L}_1 \Rightarrow 1] = \Pr[\mathcal{A} \diamond \mathcal{L}_2 \Rightarrow 1]$
Indistinguishable: Let $\mathcal{L}_{left}$, $\mathcal{L}_{right}$ be two libraries with a common interface. $\mathcal{L}_{left} \approx \mathcal{L}_{right}$ if for all polynomial-time programs $\mathcal{A}$ that output a single bit, $\Pr[\mathcal{A} \diamond \mathcal{L}_{left} \Rightarrow 1] \approx \Pr[\mathcal{A} \diamond \mathcal{L}_{right} \Rightarrow 1]$

5.2 b

For every polynomial function $p(\lambda)$:
(1)$\lim\limits_{\lambda\to\infty} \frac{p(\lambda)}{2^{\frac{\lambda}{2}}} = 0$
(2)$\lim\limits_{\lambda\to\infty} \frac{\lambda^4}{2^{\log(\lambda^2)}} = \lim\limits_{x\to\infty} \frac{x^2}{2^{\log x}} = \lim\limits_{x\to\infty} x = \infty$
(3)$\lim\limits_{\lambda\to\infty} \frac{p(\lambda)}{\lambda^{\log(\lambda)}} = \lim\limits_{\lambda\to\infty} \lambda^{C-\log(\lambda)} = 0$
(4)$\lim\limits_{\lambda\to\infty} \frac{\lambda^3}{\lambda^2} = \lim\limits_{\lambda\to\infty} \lambda = \infty$
(5)$\lim\limits_{\lambda\to\infty} \frac{p(\lambda)}{2^{(\log\lambda)^2}} = 0$
(6)$\lim\limits_{\lambda\to\infty} \frac{\lambda^2}{(\log\lambda)^2} = \lim\limits_{t\to\infty} \frac{(2^t)^2}{t^2} = \infty$
(7)$\lim\limits_{\lambda\to\infty} \frac{\lambda}{\lambda^{\frac{1}{\lambda}}} = \lim\limits_{\lambda\to\infty} \lambda^{1-\frac{1}{\lambda}} = \infty$
(8)$\lim\limits_{\lambda\to\infty} \frac{\lambda}{\sqrt{\lambda}} = \lim\limits_{\lambda\to\infty} \sqrt{\lambda} = \infty$
(9)$\lim\limits_{\lambda\to\infty} \frac{p(\lambda)}{2^{\sqrt{\lambda}}} = 0$
Therefore, (1) (3) (5) (9) are negligible functions in λ.

5.3 c

5.3.1

Proof:
Since f and g are negligible, we can get $\lim\limits_{\lambda\to\infty} f(\lambda)\cdot p(\lambda) = 0$, $\lim\limits_{\lambda\to\infty} g(\lambda)\cdot p(\lambda) = 0$. Thus $\lim\limits_{\lambda\to\infty}(f+g)(\lambda)\cdot p(\lambda) = \lim\limits_{\lambda\to\infty} f(\lambda)\cdot p(\lambda) + \lim\limits_{\lambda\to\infty} g(\lambda)\cdot p(\lambda) = 0$.
So we can conclude that $f+g$ is negligible.

5.3.2

Proof:
Since f and g are negligible, we can get $\lim_{\lambda\to\infty} f(\lambda)\cdot p(\lambda) = 0$, $\lim_{\lambda\to\infty} g(\lambda)\cdot p(\lambda) = 0$. Thus $\lim_{\lambda\to\infty}(f\cdot g)(\lambda)\cdot p(\lambda) = \lim_{\lambda\to\infty} f(\lambda)\cdot p(\lambda)\cdot \lim_{\lambda\to\infty} g(\lambda)\cdot p(\lambda) = 0$.
So we can conclude that $f\cdot g$ is negligible.

5.3.3

Counterexample:
Let $f(\lambda) = \frac{1}{2^{\frac{\lambda}{2}}}$, $g(\lambda) = \frac{1}{2^{\lambda}}$. Apparently, f and g are negligible. However, $\frac{f(\lambda)}{g(\lambda)} = 2^{\frac{\lambda}{2}}$ and $\lim_{\lambda\to\infty} 2^{\frac{\lambda}{2}}\cdot p(\lambda) = \infty$.
So we can conclude that $\frac{f(\lambda)}{g(\lambda)}$ is not negligible.