CNS Homework 2

B10202012 劉仲楷

Handwriting

1 BGP

a) {10.10.220.0/x, [AS 999]} where $x > 22$. For example, 10.10.220.0/23. Since the one with the longest prefix matched will be selected, the attacker will attack successfully.

b) A BGP blackholing can be used to mitigate the DDOS attack. When an attack is detected on a specific IP address, a null interface can be introduced and advertised by BGP. Then the traffic would be routed to the null interface and be discarded.

c) {10.10.220.0/23, [AS 1, AS 2, AS 999]}. As before, AS 3, AS 4 and AS 5 would prefer the longer prefix and the traffic would be led to AS 999. For AS 1 and AS 2, because of loop prevention, they would drop the advertisement from AS 999 since their own ASN is in AS Path. Thus, this situation in Figure 3 could be the result.

d) Since the path prepending is not signed or verified, anyone on the path can manipulate it. The attackers alter the AS path to control the traffic associated with some heuristic. For example, the attackers can manipulate the length of the AS path to make it more favorable or less attractive according to the purpose. As for loop prevention, since the path can be manipulated, the attackers can create the loop intentionally to make certain servers drop some AS path.

e) In the original purpose, the two strategies are designed to optimize network performance, manage congestion, and etc. The design of the path vector provides convenience for each AS to choose a more desirable path. The loop prevention is used to avoid traffic congestion.

f)
- MPL can prevent malicious attackers from hijacking traffic by advertising more specific routes than those generally used. Since MPL restricts the acceptance of unusually long prefixes, it can protect against attacks where an unauthorized AS announces an IP block that has a longer prefix length, thus ensuring that only legitimate routes are considered.
- By setting an MPL that discards longer prefixes, there may be unintended consequences for networks that legitimately need to advertise more specific prefixes for reasons such as traffic engineering and load balancing. For instance, an organization that needs to route traffic differently for various subnets within its network might find this policy restrictive, leading to less optimal network performance and management capabilities.

2 Internet Insecurity

a)
- In a SYN flood attack, an attacker sends a high volume of SYN packets to a server but does not respond to the SYN-ACK packets, causing the server to wait for the final ACK that never arrives. This can exhaust the server's resources, particularly the memory allocated for such half-open connections, potentially leading to a service outage. SYN cookies allow the server to handle a large volume of incoming SYN requests without over-utilizing its memory and processing capabilities. This technique transfers the storage and the memory overhead into the computational overhead in the final ACK stage.
- Including a timestamp in the SYN cookie helps the server identify and discard old connection requests. It ensures that the SYN-ACK response is relevant only within a specific time window, thus protecting against replay attacks where old packets are resent to disrupt the service.
- Including a Client IP disables attackers to use others' SYN cookies to send to the server. Moreover, it prevents attackers from performing DDOS attacks.

b) Now the server transfers the storage and memory overhead to the computational overhead in the ACK stage. However, MAC should be easy to compute. Therefore, it is hard to perform a DOS attack. The only way is that the attacker has more resources than the server. Then, for example, performing a DDOS attack becomes feasible.

c) Yes, since the SPF checks the IP with DNS, any attack on those would be feasible to spoof an email sender. For example, IP spoofing, DNS hijacking and DNS cache poisoning will all lead the check procedure to fail in detecting the spoofing.

3 Perfect Zero Knowledge

3.1 Graph Isomorphism

a) We can verify whether $G_0 \equiv G_1$ with given π by checking the elements of two adjacent matrix. We can assume the two matrices have the same size since it is the necessary condition for isomorphism.

$$[[(A_{G_0})_{i,j} = (A_{G_1})_{\pi(i),\pi(j)} \quad \forall\{i,j\} \in [n]^2]]$$

If the expression is true for all elements in the adjacent matrix, G_0 is isomorphic to G_1, otherwise not. We can obviously see that it is deterministic polynomial time since there are n^2 matrix elements.

3.2 Zero-Knowledge Interactive Proof Systems

b) Suppose G_0 and G_1 are not the same, i.e. E_0 and E_1 are not the same, or it is trivial. Besides, suppose π is not the same as id_n for the same reason.

- Completeness: For the case $b = 1$, if $G_0 \equiv G_1$, then
$$\pi_b(G) = \pi(G_0) = G_1 = G_b.$$
Hence, the V will output 1. On the other hand, for $b = 0$, the V output 1, too, because
$$\pi_b(G) = \mathrm{id}_n(G_0) = G_0 = G_b.$$
As a result,
$$\Pr[\langle P, V\rangle(G_0, G_1) = 1] = 1$$
For Protocol A, the completeness error is still $0 < 1/3$. There is no violation.
- Soundness: Suppose $G_0 \not\equiv G_1$ and a cheating prover P^* trying to fool V'. The only case V will output 1 is that $b = 0$.
$$[[\pi_b(G) = \mathrm{id}_n(G_0) = G_0 = G_b]] = 1,$$
which happens with probability $1/2$. The prover fools the verifier twice with probability $1/2 \times 1/2 < 1/3$. There is no violation.
- Perfect zero knowledge: A cheating verifier V^* sends $b = 1$. If $G_0 \equiv G_1$, the verifier gets $\pi = \pi_b$ within one round. Thus, the perfect zero knowledge property is violated.

c) i) **Construction**

Suppose E_0 and E_1 are not the same. We assume the verifier V^* outputs its view (its inputs (G_0, G_1), randomness b and messages received from the prover π_b. The S^* does the following.

- S^* choose $b' \in_R \{0, 1\}$ and $\tau \in_R S_n$.
- S^* computes $G = \tau(G_{b'})$.
- S^* sends G to V^* and gets a challenge bit $b \in \{0, 1\}$.
- If $b = b'$, S^* sends τ to V^* and outputs whatever V^* does.
- If $b \neq b'$, S^* goes back to step one and uses new independent b' and τ.

ii) **Identically Distributed**

First, we need to show the distribution of $G = \tau(G_{b'})$ sent by S^* identical to the one sent by P. In the case $b' = 0$, it is exactly the same. As for the case $b' = 1$, since b' and τ sampled independently,
$$\Pr[G = \tau(G_0)] = \Pr[G = \tau(G_{b'})]$$
for any fixed τ, G_0 random over b'. Since G_0 is isomorphic to G_1, the distribution of $\pi(G_0)$ and $\pi(G_1)$ is the same for random π. Thus, the distribution of $G = \pi(G_{b'})$ is independent of b'. Conditioned on event $b = b'$, since G has the correct distribution, the output of S^* and V^* are identically distributed. S^* retries independently when $b \neq b'$ until $b = b'$, but this makes sure for every $G_i \in \{\pi(G_0) : \forall \pi \in S_n\}$
$$\Pr[G = G_i] = \Pr[G = G_i \,|\, b = b'] = \Pr[G = G_i \text{ when } V^* \text{ is interacting with } P]$$
The distribution of the view of V^*, when engaging with S^*, is essentially indistinguishable from the distribution of the view of $V*$ under the same circumstances of interacting with S^* and where b is equal to b'. Additionally,

this distribution is equivalent to the outcome produced by V^* when it interacts with P. Hence the output of the simulator S^* is identically distributed as the view of V^*.

iii) **Expected Polynomial Time**
It remains to show that S^* runs in expected polynomial time. Note that the distribution of G is identical in the case $b' = 0$, 1. Moreover, b' is sampled uniformly and independent of b. Thus, the case $b' = b$ happens with probability 1/2. Hence, the expected tries of S^* is $1/p = 2$, which is polynomial time on average.

3.3 Non-interactive Zero-Knowledge in the plain model

d) i) **Algorithm Construction**

- Use the given proof system (P, V) to construct a randomized simulator S^* which runs in expected polynomial time $E[T(x)]$ such that for every $x \in L$, the distribution of the output of $S^*(x)$ is identical to the distribution of the view of V^* (by definition 5).
- Run the simulator S^* with input x. If the output of S^* would convince V, output 1; otherwise output 0. This could be achieved because the proof system is actually non-interactive.
- Suppose the $A(x)$ run S^* for at most $k \cdot E[T(x)]$. If S^* runs exceed this time interval, $A(x)$ outputs 0.

ii) **Polynomial Time**
The algorithm suffices because it runs at most $k \cdot E[T(x)]$. The $E[T(x)]$ is the expected polynomial time.

iii) **Completeness**
For $x \in L$, $\Pr[A'(x) = 1] \geq 1-c$[1]. Moreover, according to Markov's Inequality,

$$\Pr[T(x) \geq k \cdot E[T(x)]] \leq \frac{1}{k}.$$

We assume that the events $A'(x) = 1$ and $T(x) \geq k \cdot E[T(x)]$ are independent.

$$\begin{aligned}
\Pr[(A(x) = 1)] &= \Pr[(A'(x) = 1) \wedge (T(x) < k \cdot E[T(x)])] \\
&= \Pr[A'(x) = 1] \cdot \Pr[T(x) < k \cdot E[T(x)]] \\
&\geq (1-c)\left(1 - \frac{1}{k}\right) \\
&= 1 - c - \frac{1}{k} + \frac{c}{k} \\
&\geq 1 - c - \epsilon
\end{aligned}$$

Thus, the last inequality can be achieved by choosing $k \leq (1-c)/\epsilon \quad \forall \epsilon > 0$.

iv) **Soundness**
For $x \notin L$, $\Pr[A'(x) = 1] \leq s$

$$\begin{aligned}
\Pr[(A(x) = 1)] &= \Pr[(A'(x) = 1) \wedge (T(x) < k \cdot E[T(x)])] \\
&\leq \Pr[A'(x) = 1] \\
&\leq s
\end{aligned}$$

[1]For clarity, here I denote A' as the result of simulator without imposing the "early stop" constraint.

Capture The Flag

The write-ups are mainly about what the vulnerability is, how to implement and the logic of capturing the flag. More implementation details are in `readme.txt`.

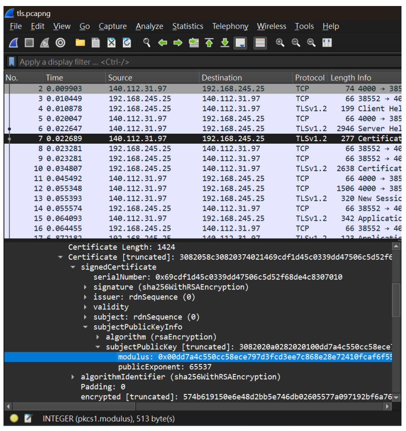
4 TLS

`CNS{TLS_4nD_w1re5h4rK_@rE_FuNNNNN!}`

We first find the public key in the `tls.pcapng` file using Wireshark. Since the two prime factors p and q of an RSA modulus n are too close to each other, we can exploit the Fermat attack with RsaCtfTool. The code is in the first part of `code4.ipynb`.

Referring to the website, after getting the private key file, set it into the Wireshark by `Edit > Preferences > Protocols > TLS > RSA keys list > Edit... > + > Key File`. Apply a display filter "data". Then we can get the username `just_a_user`, the password `ILOVEw1Re5hark`, the command `giveMeTheFl4g`, the instruction for the next step `NOT_FL4G{connect_to_the_server_yourself_to_get_the_fl4g}` and the root CA key.

Referring to the website, using the root CA key, we can sign the CA certificate and use it to sign the client certificate. The certificate info can be found in the decrypted package too. The main difference is `OU = VIP` and `CN = Alice413` in the client certificate. After connecting and verified by the server, we just enter the username, the password and the command found just now and we can get the flag. The code is in the second part of `code4.ipynb`.

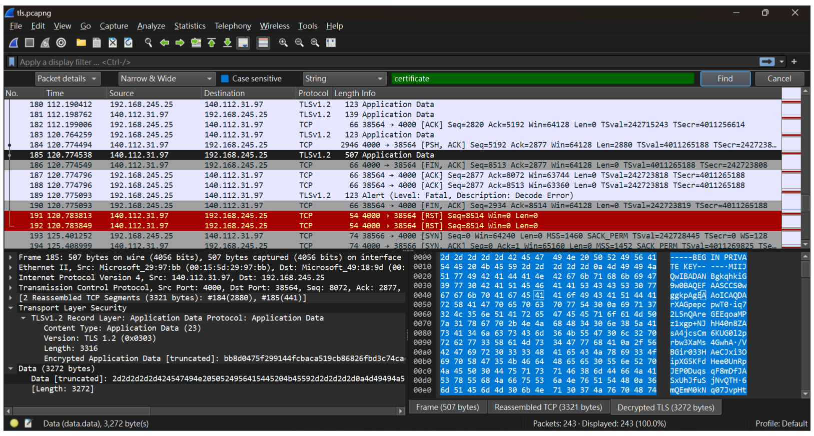

tls.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time      Source          Destination     Protocol Length Info
   2 0.009903  140.112.31.97   192.168.245.25  TCP        74 4000 → 385
   3 0.010449  192.168.245.25  140.112.31.97   TCP        66 38552 → 40
   4 0.010878  192.168.245.25  140.112.31.97   TLSv1.2   199 Client Hel
   5 0.020047  140.112.31.97   192.168.245.25  TCP        66 4000 → 385
   6 0.022647  140.112.31.97   192.168.245.25  TLSv1.2  2946 Server Hel
   7 0.022689  140.112.31.97   192.168.245.25  TLSv1.2   277 Certificat
   8 0.023281  192.168.245.25  140.112.31.97   TCP        66 38552 → 40
   9 0.023281  192.168.245.25  140.112.31.97   TCP        66 38552 → 40
  10 0.034807  192.168.245.25  140.112.31.97   TLSv1.2  2638 Certificat
  11 0.045492  140.112.31.97   192.168.245.25  TCP        66 4000 → 385
  12 0.055348  140.112.31.97   192.168.245.25  TCP      1506 4000 → 385
  13 0.055393  140.112.31.97   192.168.245.25  TLSv1.2   320 New Sessic
  14 0.055574  192.168.245.25  140.112.31.97   TCP        66 38552 → 40
  15 0.064093  140.112.31.97   192.168.245.25  TLSv1.2   342 Applicatic
  16 0.064455  192.168.245.25  140.112.31.97   TCP        66 38552 → 40
```

```
      Certificate Length: 1424
    ▾ Certificate [truncated]: 3082058c30820374021469cdf1d45c0339dd47506c5d52f6
      ▾ signedCertificate
          serialNumber: 0x69cdf1d45c0339dd47506c5d52f68de4c8307010
        ▸ signature (sha256WithRSAEncryption)
        ▸ issuer: rdnSequence (0)
        ▸ validity
        ▸ subject: rdnSequence (0)
        ▾ subjectPublicKeyInfo
          ▸ algorithm (rsaEncryption)
          ▾ subjectPublicKey [truncated]: 3082020a0282020100dd7a4c550cc58ece
              modulus: 0x00dd7a4c550cc58ece797d3fcd3ee7c868e28e72410fcaf6f5
              publicExponent: 65537
      ▸ algorithmIdentifier (sha256WithRSAEncryption)
        Padding: 0
        encrypted [truncated]: 574b619150e6e48d2bb5e746db02605577a097192bf6a76
```

```
INTEGER (pkcs1.modulus), 513 byte(s)
```

Figure 1: Finding Public Key in `tls.pcapng` using Wireshark


tls.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Packet details
Narrow & Wide
Case sensitive
String
certificate
Find
Cancel
Frame (507 bytes)
Reassembled TCP (3321 bytes)
Decrypted TLS (3272 bytes)

```
No.  Time        Source          Destination     Protocol Length Info
180 112.190412  192.168.245.25  140.112.31.97   TLSv1.2   123 Application Data
181 112.198762  140.112.31.97   192.168.245.25  TLSv1.2   139 Application Data
182 112.199006  192.168.245.25  140.112.31.97   TCP        66 38564 → 4000 [ACK] Seq=2820 Ack=5192 Win=64128 Len=0 TSval=242715243 TSecr=4011256614
183 120.764259  192.168.245.25  140.112.31.97   TLSv1.2   123 Application Data
184 120.774494  140.112.31.97   192.168.245.25  TCP      2946 4000 → 38564 [PSH, ACK] Seq=5192 Ack=2877 Win=64128 Len=2880 TSval=4011265188 TSecr=2427238…
185 120.774538  140.112.31.97   192.168.245.25  TLSv1.2   507 Application Data
186 120.774549  140.112.31.97   192.168.245.25  TCP        66 4000 → 38564 [FIN, ACK] Seq=8513 Ack=2877 Win=64128 Len=0 TSval=4011265188 TSecr=242723808
187 120.774796  192.168.245.25  140.112.31.97   TCP        66 38564 → 4000 [ACK] Seq=2877 Ack=8072 Win=63744 Len=0 TSval=242723818 TSecr=4011265188
188 120.774796  192.168.245.25  140.112.31.97   TCP        66 38564 → 4000 [ACK] Seq=2877 Ack=8513 Win=63360 Len=0 TSval=242723818 TSecr=4011265188
189 120.775093  192.168.245.25  140.112.31.97   TLSv1.2   123 Alert (Level: Fatal, Description: Decode Error)
190 120.775093  192.168.245.25  140.112.31.97   TCP        66 38564 → 4000 [FIN, ACK] Seq=2934 Ack=8514 Win=64128 Len=0 TSval=242723819 TSecr=4011265188
191 120.783813  140.112.31.97   192.168.245.25  TCP        54 4000 → 38564 [RST] Seq=8514 Win=0 Len=0
192 120.783849  140.112.31.97   192.168.245.25  TCP        54 4000 → 38564 [RST] Seq=8514 Win=0 Len=0
193 125.401252  192.168.245.25  140.112.31.97   TCP        74 38566 → 4000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=242728445 TSecr=0 WS=128
194 125.408999  140.112.31.97   192.168.245.25  TCP        74 4000 → 38566 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1452 SACK_PERM TSval=4011269825 TSe…
```

```
▸ Frame 185: 507 bytes on wire (4056 bits), 507 bytes captured (4056 bits) on interface
▸ Ethernet II, Src: Microsoft_29:97:bb (00:15:5d:29:97:bb), Dst: Microsoft_49:18:9d (00:
▸ Internet Protocol Version 4, Src: 140.112.31.97, Dst: 192.168.245.25
▸ Transmission Control Protocol, Src Port: 4000, Dst Port: 38564, Seq: 8072, Ack: 2877,
▸ [2 Reassembled TCP Segments (3321 bytes): #184(2880), #185(441)]
▾ Transport Layer Security
  ▾ TLSv1.2 Record Layer: Application Data Protocol: Application Data
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 3316
      Encrypted Application Data [truncated]: bb8d0475f299144fcbaca519cb86826fbd3c74ca
▾ Data (3272 bytes)
    Data [truncated]: 2d2d2d2d2d424547494e205052495641544520204b45592d2d2d2d2d0a4d49494a5
    [Length: 3272]
```

```
0000  2d 2d 2d 2d 2d 42 45 47  49 4e 20 50 52 49 56 41   -----BEG IN PRIVA
0010  54 45 20 4b 45 59 2d 2d  2d 2d 2d 0a 4d 49 49 4a   TE KEY-- ---·MIIJ
0020  51 77 49 42 41 44 41 4e  42 67 6b 71 68 6b 69 47   QwIBADAN BgkqhkiG
0030  39 77 30 42 41 51 45 46  41 41 53 43 43 53 30 77   9w0BAQEF AASCCS0w
0040  67 67 6b 70 41 67 45 41  41 6f 49 43 41 51 44 41   ggkpAgEA AoICAQDA
0050  72 58 41 47 70 65 70 63  70 77 54 30 0a 69 71 37   rXAGpepc pwT0·iq7
0060  32 4c 35 6e 51 41 72 65  47 45 45 71 6f 61 4d 50   2L5nQAre GEEqoaMP
0070  7a 31 78 67 70 2b 4e 4a  68 48 34 30 6e 38 5a 41   z1xgp+NJ hH40n8ZA
0080  73 41 34 6a 63 73 43 6d  36 4b 55 47 30 6c 32 70   sA4jcsCm 6KUG0l2p
0090  72 62 77 33 58 61 4d 73  34 47 77 68 41 0a 2f 56   rbw3XaMs 4GwhA·/V
00a0  42 47 69 72 30 33 33 48  41 65 43 4a 78 69 33 4f   BGir033H AeCJxi3O
00b0  69 70 58 47 35 4b 46 64  48 65 65 30 55 6e 52 70   ipXG5KFd Hee0UnRp
00c0  4a 45 50 30 44 75 71 73  71 46 38 6d 44 66 4a 41   JEP0Duqs qF8mDfJA
00d0  53 78 55 68 4a 66 75 53  6a 4e 76 51 54 48 0a 36   SxUhJfuS jNvQTH·6
00e0  6d 51 45 6d 4d 30 6b 4e  71 30 37 4a 76 70 48 74   mQEmM0kN q07JvpHt
```

```
Data (data.data), 3,272 byte(s)          Packets: 243 · Displayed: 243 (100.0%)          Profile: Default
```

Figure 2: Root CA.key Found in the Decrypted Package

5 Ditto Knowledge Proof

a) `CNS{Y0U_1N_7H3_M1DDL3_4774CK_10c9d390f57b6b6f690519206ff3579b}`
The sequence diagram is shown in table 1. Since we can access both Alice and Bob,

we can perform the Man in the Middle attack. Simply open two channels for Alice and Bob respectively and send everything received from one to another.

Alice		Bob
choose random r		
calculate $a = g^r \mod p$	$\xrightarrow{a}$	
	$\xleftarrow{c}$	choose random $c \in Z_p$
calculate $w = cx + r$	$\xrightarrow{w}$	
		Verify $g^w = y^c a \mod p$
		correct $\Rightarrow$ Alice knows x

Table 1: Zero-Knowledge Proof Sequence Diagram, where private-public key pair is $(x, y = g^x)$ in group Z_p^*

b) `CNS{P53UD0_R4ND0M_4R3_N07_R4ND0M_6ee07d69ce5587b46f638c90b341f43f}`
We can look up the source code and get the method of generating random c. Then, once we get a c, we can predict the next c' and construct $a = y^{-c'}$. Therefore, the w is trivially 0.

c) `CNS{R3PL4Y_4774CK_15_50000_51MPL3_a80e34ed1a90b3ae51755fc290dc81a8}`
The vulnerability is the admin mis-saved the private file to the public file. Moreover, the admin uses the c which depends on a (and p, g, y but they are the same). Therefore, once we get a pair of a and w, we can impersonate the admin. To fix it, we can change the c by adding a timestamp. However, here we assume that the two parties are performing the protocol instantly without a time lag.

d) `CNS{CDH_15_1N53CUR3_WH3N_50M37H1N6_F41L3D_f680114b0db7bd36f17b4c129c78f2de}`
Pohlig–Hellman algorithm can solve a given discrete logarithm problem, whose order is a smooth integer. An n-smooth number is an integer whose prime factors are all less than or equal to n. Thus, we can perform this algorithm to solve the private key x. I used the online tool[2] to compute for the x and recover the flag in the `code5_cd.py`.

6 So Anonymous, So Hidden

a) `CNS{Y0u_Ar3_A_g00D_m1x3R}`
After hex-decoding a received packet, we can utilize the function `decrypt_server` in the `lib.py` file to get the next hop and the decrypted message. Buffer the packet until there are 10 packets in total. Send the ten packets in random order.

b) `CNS{b0X_1n_B0x_=_b0xC3pTi0n}`
Simply reverse the order of decryption. The core concept is taught in class:

$$E_{pk_{mix}}(R_1 || E_{pk_b}(R_0 || M) || \text{“Bob's IP”})$$

[2]The notation on the online tool is slightly different. The h on the website is the public key y in this context.

However, the encryption method is slightly different in this code. The public key is used to encrypt the symmetric key given in the message. While the main message uses the symmetric key to encrypt. Thus, the ciphertext is in the format of

$$E_{pk_{mix_1}}(sym_1)||E_{sym_1}(\text{“Bob's IP”}||E_{pk_b}(sym_b)||E_{sym_b}(M))$$

c) The public keys of the servers are all not very long. Some of them may have some vulnerability and can be factored by some RSA attack methods. For example, we can look up the website factordb. We do not need to factor out all of the public keys. Since Alice only chooses four of them and Bob, we only need to factor out the five, which increases the success probability. However, we don't know which four are chosen, therefore, we can try our best to factor out eleven of them (Bob's must be factored out) and try to decrypt the packet. If we have at least factored out four public keys and Bob's one, we can get the flag in probability above $1/C^10_4 \approx 0.5\%$. If these are not the four servers Alice chose, we can retry again.

d) I used the code in this website to get the domain name from public key which is `cns24otgzs5zmn2oztxnmiff7bscpg7kp7p6wnkayjnvr5fzxbqzeuqd.onion`. I tried to connect the server with all ports but failed to get the flag. I guess I get the wrong domain name.