Spring 2024 Cryptography and Network Security

Homework 2

Release Date: 2024/04/09

Due Date: 2024/05/06, 23:59

TA's Email: `cns@csie.ntu.edu.tw`

Instructions

- This homework assignment is valued at 110 points, but only up to 100 points will be graded, even if you score above 100. Some problems are marked as bonus points just because they may be more challenging than others.
- **Submission Guide:** Please submit all your codes and report to NTU COOL. Please refer to the homework instructions slides for information.
- You may encounter new concepts that haven't been taught in class, and thus you're encouraged to discuss with your classmates, search online, ask TAs, etc. However, **you must write your own answer and code**. Violation of this policy leads to serious consequences.
- You may need to write programs in the Capture The Flag (CTF) problems. Since you can use any programming language you prefer, we will use a pseudo extension code**.ext** (e.g., code.py, code.c) when referring to the file name in the problem descriptions.
- In each of the CTF problems, you need to find out a flag, which is in `CNS{...}` format, to prove that you have succeeded in solving the problem.
- Besides the flag, you also need to submit the code you used and a short write-up in the report to get full points. The code should be named **code{problem_number}.ext**. For example, code3.py.
- In some CTF problems, your solution may involve human-laboring or online tools. These are allowed as long as you get the flag not by cheating or plagiarism. If your code does not directly output the flag (e.g., it requires the user to do manual filtering on some messages), please specify the execution process of your code in **readme.txt** file.
- In some CTF problems, you need to connect to a given service to get the flag. These services only allow connections from 140.112.0.0/16, 140.118.0.0/16, and 140.122.0.0/16.

Handwriting

1. BGP (18%)

In this problem, we would like you to explain and analyze some attacks against the BGP routing protocol. In both attacks, the attacker has control over AS999 and wants to attack AS1000, suppose the attacker wants to target the traffic destined for 10.10.220.XXX. Figure 1 shows the routing paths in the normal state after AS1000 has announced 10.10.220.0/22. Each circle represents an Autonomous System (AS). A solid line indicates a link over which two neighboring ASes can exchange control messages such as BGP update messages. A dashed line indicates an established AS path to 10.10.220.0/22.

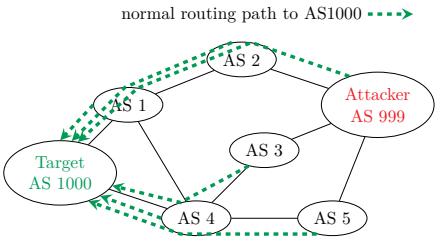

Figure 1: Normal scenario.

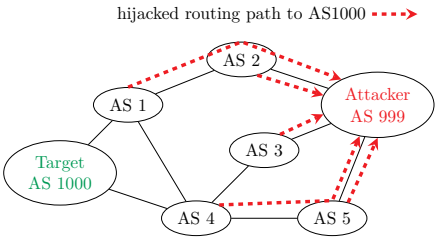

Figure 2: BGP hijacking.

a) (3%) Figure 2 illustrates the routing paths after the attacker launched a BGP hijacking attack. What did the attacker (AS999) announce?

b) (3%) The technique leveraged by BGP hijacking could also be used in a good way, called BGP blackholing. Describe an attack that can be mitigated by BGP blackholing.

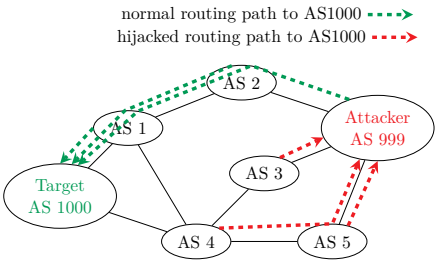

Figure 3: BGP man-in-the-middle.

In the second type of attack illustrated in Figure 3, the attacker can silently redirect the hijacked traffic back to the victim along the path indicated by green lines. This attack exploits **AS Path Prepending**, where an AS inserts AS numbers at the beginning of an AS path to make this path less preferable for traffic engineering purpose, and **Loop Prevention**, where AS x drops any BGP update with itself (i.e., AS x) in the AS-Path attribute to prevent routing loops.

c) (3%) Instead of announcing the ownership of an address block, AS999 announces a fake BGP update. Specify a BGP update message (in the form of {IP prefix, [AS x, AS y, $\cdots$]}) that could cause the result of Figure 3.

d) (3%) Briefly explain how the attacker misuses path prepending and loop prevention for malicious purpose.

e) (3%) Discuss how the technique in (c) might be useful in a good way.

To mitigate BGP prefix hijacking, some propose the idea of BGP Maximum Prefix Limit (MPL). If a prefix length of an advertisement is longer than the MPL, the advertisement will be discarded. For example, if MPL is set to 24 in a BGP speaker, the announcement of 10.10.220.0/25 will simply be ignored.

f) (3%) Give one advantage and one disadvantage of MPL.

2. Internet Insecurity (15%)

a) (5%) You have learned about the SYN flood attack in class. Explain why SYN cookie can migrate the SYN flood attack and why it needs to contain a timestamp and the client's IP address.

b) (5%) Similiar to SYN flood, ACK flood means that an attacker attempts to overload a server with TCP ACK packets. Think about whether ACK flood is possible to cause denial of service. Please explain why or why not.

c) (5%) You have learned about Sender Policy Framework (SPF) in SMTP and email security. Is there any way to spoof an email sender even if SPF works correctly? Explain why or why not (If the answer is yes, You don't need to introduce a actual attack. Telling where the vulnerability is and a brief explanation will be sufficient).
Hint: Is the assumption of binding a SMTP server with IPs and Domain Names strong enough?

3. Perfect Zero Knowledge (22%)

Convention

- We write $[n] = \{1, 2, \ldots, n\}$ for any positive integer n, and write $f(A) = \{f(a) \in Y : a \in A\}$ and $f^{-1}(B) = \{x \in X : f(x) \in B\}$ for a function $f : X \to Y$ and $A \subseteq X$, $B \subseteq Y$.
- $\mathrm{id}_n : [n] \to [n]$ is the identity function on $[n]$; that is, $\mathrm{id}_n(x) = x$ for every $x \in [n]$.
- $x_1, \ldots, x_n \in_R X$ means for $i \in [n]$, x_i is independently and uniformly sampled from a set X.
- $y \leftarrow A(x)$ sets y to the output of running an algorithm A with input x.
- $x := y$ means assigning y to x.
- S_n is the set of all bijections from $[n]$ to $[n]$, where n is a positive integer.
- $[[\text{some expression}]] = 1$ if the expression is true, and it is 0 if the expression is false.
- $x \bmod p$ is the remainder of x divided by p.

3.1. Graph Isomorphism

Definition 1. A graph $G = (V, E)$ consists of a finite set of vertices V, and a set of edges E. An edge $e \in E$ is a set of two vertices. For any bijection $\pi : V_0 \to V_1$, and a graph $G_0 = (V_0, E_0)$, we define $\pi(G_0)$ to be the graph $G_1 = (V_1, E_1)$ with

$$V_1 = \pi(V_0)$$

and

$$E_1 = \{\{\pi(u), \pi(v)\} \subseteq V_1 : \{u, v\} \in E_0\}.$$

Given graphs $G_0 = (V_0, E_0), G_1 = (V_1, E_1)$, an isomorphism from G_0 to G_1 is a bijection $\pi : V_0 \to V_1$ with $\pi(G_0) = G_1$. If there is an isomorphism from G_0 to G_1, we say that G_0 is isomorphic to G_1, and write $G_0 \equiv G_1$.

We may assume our graphs are represented as adjacent matrices. That is, for graph $G = ([n], E)$, the corresponding n-by-n adjacent matrix A_G is defined as

$$(A_G)_{i,j} = \begin{cases} 1 & \{i,j\} \in E \\ 0 & \{i,j\} \notin E \end{cases}.$$

For example, a triangle $K_3 = ([3], \{\{1,2\},\{2,3\},\{3,1\}\})$ has adjacent matrix

$$K_3 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

Since an adjacent matrix of a graph with n vertices has n^2 binary entries, we take n to be the size of the graph.

Problem 2 (Graph Isomorphism Problem)**.** The Graph Isomorphism Problem (GI) is the following: given graphs G_0 and G_1, decide whether $G_0 \equiv G_1$. That is,

$$\text{GI} = \{(G_0, G_1) : G_0 \equiv G_1\}.$$

For example, let $G_1 = ([5], E_1)$, $G_2 = ([5], E_2)$, and $G_3 = ([5], E_3)$, where

$$E_1 = \{\{1,2\},\{2,3\},\{3,1\}\}, \quad E_2 = \{\{1,3\},\{5,1\},\{5,3\}\}, \quad E_3 = \{\{1,2\},\{2,3\},\{3,5\}\}.$$

Then $G_1 \equiv G_2$ because there is an isomorpism

$$\begin{aligned} \pi(1) &= 1 \\ \pi(2) &= 3 \\ \pi(3) &= 5 \\ \pi(4) &= 2 \\ \pi(5) &= 4 \end{aligned}$$

from G_1 to G_2. $\{1,2\} \in E_1$ corresponds to $\{1,3\} \in E_2$, $\{2,3\} \in E_1$ corresponds to $\{3,5\} \in E_2$, and $\{3,1\} \in E_1$ corresponds to $\{5,1\} \in E_2$. However, $G_1 \not\equiv G_3$ because there is not any isomorphism between G_1 and G_3.

Before introducing more advanced proof systems, we rephrase the complexity class NP to introduce the terminology.

Definition 3. $L \in \mathsf{NP}$ if there is a polynomial time deterministic verifier V and a polynomial p such that

- (Completeness): if $x \in L$, then there is a proof $y \in \{0,1\}^{p(|x|)}$ with $V(x,y) = 1$.
- (Soundness): if $x \notin L$, then for every proof $y \in \{0,1\}^{p(|x|)}$, $V(x,y) = 0$.

a) (3%) Show that GI $\in \mathsf{NP}$.

Hint: You have to design a deterministic polynomial time verifier V such that for every pair of graphs (G_0, G_1), $G_0 \equiv G_1$ if and only if there is a proof π of size polynomial in $|G_0| + |G_1|$.

3.2. Zero-Knowledge Interactive Proof Systems

In addition to non-interactive proof systems like those in NP, we are also interested in interactive proof systems, where the verifier may challenge the prover, and decide whether they want to accept the statement depending on how well the prover responded to the challenge.

An interactive proof system is a protocol between a (possibly computationally unbounded) prover P and a polynomial time randomized verifier V. In an interaction, only one of P and V is active at any time. For example, when P is executing, V waits until P stops and sends a message to V. At the beginning of the interaction, P and V both get a common problem instance x, and P may get an additional input y that helps it prove $x \in L$ to V. P and V execute and send messages in turn, until V halts and outputs 1 for accepting, or outputs 0 for rejecting. We assume that the messages sent by the prover and the verifier are in the correct format, so each can smoothly parse the received message. We also assume that both do not abort the interaction. (Although we do not use abort in this homework, it is necessary in some more complicated cases.)

We denote by $\langle P, V\rangle(x)$ the output of V after an interaction. Note that $\langle P, V\rangle(x)$ is a random variable because both P and V are allowed to use randomness in their computation.

We hope that if both the prover and the verifier obey the protocol, the verifier accepts the statement $x \in L$ after their interaction. However, when $x \notin L$, we hope that no matter what a cheating prover P^* does, the verifier rejects the statement $x \in L$. We formalize the idea in the following definition.

Definition 4. We say a protocol (P, V) is an interactive proof system for a language L if

- (Completeness): if $x \in L$, then
$$\Pr[\langle P, V\rangle(x) = 1] \geq \frac{2}{3}.$$
- (Soundness): for every prover P^*, if $x \notin L$, then
$$\Pr[\langle P^*, V\rangle(x) = 1] \leq \frac{1}{3}.$$

We call
$$\max_{x \in L} 1 - \Pr[\langle P, V\rangle(x) = 1]$$
the completeness error of the protocol, and call
$$\max_{x \notin L, P^*} \Pr[\langle P^*, V\rangle(x) = 1]$$
the soundness error of the protocol.

For many cryptographic applications, we might require that the verifier learn nothing besides that the statement is correct. In an interaction between P and V, the view of V is defined to be its input, the randomness it uses, and the message it receives from the prover P. For convenience, we sometimes assume a verifier V outputs its view. We can do this because we can take the code of a verifier V' that only outputs 0 or 1, and build a new verifier V that outputs the view of V' (and V itself); the view has a polynomial length and the original output can be reconstructed from the view of V' in polynomial time, so V is still randomized polynomial time. From now on, we will assume V outputs 0/1 or its view depending on which one is suitable in the situation.

Definition 5. We say a proof system (P, V) for a language L is perfect zero-knowledge if for every (possibly cheating) randomized polynomial time verifier V^*, there is a randomized simulator S^* that runs in expected polynomial time 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^*.

As an example, we consider a protocol for proving Diffie-Hellman tuples. You may skip to the next protocol if you are familiar with the proof of completeness, soundness, and perfect zero-knowledge. Suppose G is a cyclic group of order q, and g is a generator. A tuple (g, h, y_1, y_2) is a Diffie-Hellman tuple if there are $a, b \in [q]$ with

$$(g, h, y_1, y_2) = (g, g^a, g^b, g^{ab}),$$

or equivalently, if there is $x \in [q]$ with

$$(y_1, y_2) = (g^x, h^x).$$

We use Protocol 1 to prove this relation.

P	g, h, y_1, y_2	V
$y_1 = g^x$ and $y_2 = h^x$		
$r \in_R [q]; A := g^r; B := h^r$	$\xrightarrow{A,B}$	
	$\xleftarrow{c}$	$c \in_R \{0, 1\}$
$s := cx + r \bmod q$	$\xrightarrow{s}$	V outputs $[[A = g^s/y_1^c$ and $B = h^s/y_2^c\ ]]$

Table 1: Protocol for proving Diffie-Hellman tuples.

Protocol 1 is complete because when $y_1 = g^x$ and $y_2 = h^x$,

$$g^s = g^{cx+r} = (g^x)^c\, g^r = y_1^c A,$$

and

$$h^s = h^{cx+r} = (h^x)^c\, h^r = y_2^c B,$$

which implies

$$\Pr[\langle P, V\rangle(g, h, y_1, y_2) = 1] = 1.$$

Now we compute the soundness error of the protocol. Suppose (g, h, y_1, y_2) is not a Diffie-Hellman tuple. A cheating prover P^* may try to fool the verifier V by violating the protocol. (Actually, P^* cannot obey the protocol since the protocol does not specify what a prover should do when the statement is false.) We do a case analysis on A, B.

In the first case, there is some $r \in [q]$ with $A = g^r$ and $B = h^r$. Then if V chooses $c = 1$, which happens with probability $1/2$, V will output 0. Why? If $c = 1$, then $s = x + r \bmod q$, and V outputs 1 if and only if

$$Ay_1 = g^{x+r} = g^x A$$

and

$$By_2 = h^{x+r} = h^x B,$$

which happens if and only if

$$y_1 = g^x$$

and

$$y_2 = h^x.$$

But this cannot happen since (g, h, y_1, y_2) is not a Diffie-Hellman tuple. As a result, V will output 0.

In the second case, for every $r \in [q]$, either $A \neq g^r$ or $B \neq h^r$. Then if V chooses $c = 0$, which happens with probability $1/2$, V will output 0. Why? If $c = 0$, then V outputs 1 if and only if

$$A = g^{cx+r} = g^r$$

and

$$B = h^{cx+r} = h^r,$$

but these cannot happen at the same time by the assumption of this case. Therefore, V outputs 0.

In both cases, V outputs 0 at least half of the time, and V outputs 1 at most half of the time. Hence, Protocol 1 has soundness error $1/2 > 1/3$, so the protocol itself is not sound according to our definition. However, since the protocol has zero completeness error, we can run the protocol twice independently and accept only when V outputs 1 twice. This way, the probability that a cheating prover fools the verifier twice is $1/2 \times 1/2 = 1/4 < 1/3$, and the protocol is sound. The completeness error is obviously still zero.

Finally, we show that the protocol is perfect zero-knowledge. We just show that one run of the protocol is perfect zero-knowledge, since we may execute the simulator twice independently to simulate the two runs. Given a possibly cheating randomized polynomial time verifier V^*, we have to design a randomized polynomial time simulator S^* that simulates the view of V^* for every Diffie-Hellman tuple. Here, we assume V^* outputs its view (its inputs, randomness, and messages received from the prover). S^* does the following:

- S^* chooses $c' \in_R \{0, 1\}$ and $r \in_R [q]$.
- S^* computes $A = g^r / y_1^{c'}$ and $B = h^r / y_2^{c'}$.
- S^* sends A, B to V^* and get a challenge bit $c \in \{0, 1\}$.
- If $c = c'$, S^* sends r to V^* and outputs whatever V^* does.
- If $c \neq c'$, S^* goes back to step one and use new independent c' ad r.

First, we show that A, B sent by S^* is distributed identically as A, B sent by P. S^* sends

$$(A, B) = (g^r / y_1^{c'}, h^r / y_2^{c'}) = (g^{r-c'x}, h^{r-c'x}).$$

In the case $c' = 0$, this is exactly the distribution of A, B sent by P. But even in the case $c' = 1$, since c' and r are sampled independently, $r - c'x$ is uniformly distributed in $[q]$, and hence

$$\Pr[(A, B) = (g^r, h^r)] = \Pr[(A, B) = (g^{r-c'x}, h^{r-c'x})]$$

for any fixed r, x, and over random c'

Conditioned on the event $c = c'$, since A, B have the correct distribution, the output of S^* and V^* are identically distributed. S^* retries independently when $c \neq c'$, until $c = c'$, but this makes sure for every $a, b \in G$

$$\begin{aligned}\Pr[(A, B) = (a, b)] &= \Pr[(A, B) = (a, b) \mid c = c'] \\ &= \Pr[(A, B) = (a, b) \text{ when } V^* \text{ is interacting with } P].\end{aligned}$$

That is, the distribution of view of V^* when interacting with S^* is identical the distribution the view of V^* when interacting with S^* and $c = c'$, and is in turn identical to the output of

V^* interacting with P. Hence, the output of the simulator S^* is identically distributed as the view of V^*.

It remains to show S^* runs in expected polynomial time. Note that (A, B) is distributed identically in the cases $c' = 0$ and $c' = 1$, so the probability that $c = c'$ is $1/2$, and the expected number of tries of S^* is

$$\sum_{i=1}^{\infty} \frac{i}{2^i} = \sum_{i=1}^{\infty} \sum_{j=i}^{\infty} 2^{-j} = \sum_{i=1}^{\infty} 2^{-i+1} = 2.$$

This completes the proof that Protocol 1 is complete, sound and perfect zero-knowledge interactive proof system.

We now return to the Graph Isomorphism Problem. In the following protocol, the prover P tries to prove that a graph $G_0 = ([n], E_0)$ is isomorphic to $G_1 = ([n], E_1)$.

P	$G_0 = ([n], E_0), G_1 = ([n], E_1)$	V
$\pi(G_0) = G_1$		
	$\xrightarrow{G := G_0}$	
	$\xleftarrow{b \in_R \{0,1\}}$	
$\pi_b := \begin{cases} \mathrm{id}_n & b = 0 \\ \pi & b = 1 \end{cases}$	$\xrightarrow{\pi_b}$	
		V outputs $[[\pi_b(G) = G_b]]$

Table 2: A zero-knowledge interactive proof system?

Table 2 only shows one round of the protocol, while in the actual protocol, which we call Protocol A, we run Table 2 2 times and the verifier outputs 1 only if V outputs 1 in both rounds. In the following, we use (P', V') to denote Protocol A, and (P, V) to denote the inner protocol (Protocol 2).

However, Protocol A is not a perfect zero-knowledge interactive proof system for the Graph Isomorphism Problem. To see why, answer the following questions.

b) (5%) Find out which of the three properties (completeness, soundness, and perfect zero-knowledge) is not satisfied by Protocol A, the two-round protocol that runs Protocol 2 two times. Describe an attack. More specifically,

- If Protocol A is not complete, find two graphs $G_0 \equiv G_1$ but the $\langle P', V' \rangle(G_0, G_1) = 1$ with probability less than $2/3$.
- If Protocol A is not sound, find two graphs $G_0 \not\equiv G_1$ and a (possibly computationally unbounded) cheating prover P^* with

$$\Pr[\langle P^*, V' \rangle(G_0, G_1)] > \frac{1}{3}.$$

- If Protocol A is not perfect zero-knowledge, find a polynomial time cheating verifier V^* that given $G_0 \equiv G_1$, can find out the isomorphism π with $\pi(G_0) = G_1$ from its interaction with the honest prover P'.

We claim Protocal 3, although very similar to Protocol 2, is an interactive proof system for graph isomorphism with completeness error 0, soundness error $1/2$, and perfect zero-knowledge.

P	$G_0 = ([n], E_0), G_1 = ([n], E_1)$	V
$\pi(G_0) = G_1$		
$\tau \in_R S_n$	$\xrightarrow{G := \tau(G_0)}$	
	$\xleftarrow{b \in_R \{0,1\}}$	
$\pi_b := \begin{cases} \tau^{-1} & b = 0 \\ \pi \circ \tau^{-1} & b = 1 \end{cases}$	$\xrightarrow{\pi_b}$	
		V outputs $[[\pi_b(G) = G_b]]$

Table 3: A zero-knowledge interactive proof system for GI.

c) (7%) Show that Protocol 3 is perfect zero-knowledge. *Given any (possibly cheating) randomized polynomial time verifier V^*, you have to design a randomized expected polynomial time simulator S^*. Observe that the permutation τ^{-1} somehow hides the isomorphism π in the case $b = 1$.*

3.3. Non-interactive Zero-Knowledge in the plain model

Recall our definition of interactive proof systems, the prover and the verifier are allowed to send multiple messages in turn. In real-world applications, to run an interactive proof, both parties have to be online during the execution. It will be much more convenient if we can achieve zero-knowledge non-interactively, like the proofs in NP.

However, interaction is necessary if we want zero-knowledge proof systems for non-trivial languages.

d) (7%) Suppose L is a language with a perfect zero-knowledge interactive proof system (P, V) with completeness error c and soundness error s, and the only message between P and V is sent from P to V (so it is actually non-interactive). Show that for every $\epsilon > 0$, there is a randomized polynomial time algorithm A such that

- if $x \in L$,
$$\Pr[A(x) = 1] \geq 1 - c - \epsilon,$$
- and if $x \notin L$,
$$\Pr[A(x) = 1] \leq s.$$

Hint: V can decide L with the required completeness and soundness when it interacts with P. Can a simulator provide the same power? A caveat to the proof is that the simulator may run for a very long time for some internal randomness, but this cannot happen often according to Markov's Inequality.

From what you just showed, if we want a zero-knowledge proof system for some language that cannot be decided by randomized algorithms in polynomial time; for example, some NP-complete language (assuming $\mathsf{NP} \not\subseteq \mathsf{BPP}$), then we need interaction — or do we?

Furthermore, we can show that interactive proof systems without randomness are no more powerful than NP, and cannot be zero-knowledge (if the language is not in P). Can you see why?

The Graph Isomorphism Problem is unlikely NP-complete, and there is also evidence that a perfect zero-knowledge proof system does not exist for NP-complete problems. However, if we

only require the output of the simulator to be computationally indistinguishable (as in PRG security games) from the view of the verifier, we get a weaker concept of zero-knowledge called computational zero-knowledge. We have computational zero-knowledge proof systems for all NP.

Capture The Flag

4. TLS (15%)

You found a pcap file (`tls/tls.pcapng`) of data transferred between a CNS TA and a server. Eve, who is your classmate, thinks that maybe there are leaks about the CNS exam in the pcap file. Please help Eve figure out what is actually in the pcap file, connect to the server `cns.csie.org:4000`, and steal the secret from the server.

Note: try to connect to the server by yourself and get the flag. (Ncat doesn't support TLS)

Hint: What if the two prime factors p and q of an RSA modulus n are too close to each other?

5. Ditto Knowledge Proof (15% + 5% Bonus)

Given a group $\mathbb{G}$ and a generator g, the CDH assumption states that given (g, g^a, g^b), it is hard to compute g^{ab} without knowing a or b. Diffie-Hellman Key Exchange and ElGamal are based on this assumption. Here, we proposed a zero-knowledge proof scheme based on this assumption.
Alice has an ElGamal private-public key pair $(x, y = g^x)$ in group Z_p^*. Now she want to prove to Bob that she owns the private key x. Here is how she does:

1. Alice: choose random r and send $a = g^r \pmod p$ to Bob.
2. Bob: choose a random challenge $c \in Z_p$ and send c to Alice.
3. Alice: compute $w = cx + r$ and send w to Bob.
4. Bob: verify that $g^w = y^c a \pmod p$. If it is correct, Bob believes that Alice knows x.

You can access Alice by `nc cns.csie.org 23461` and access Bob by `nc cns.csie.org 23462`. The challenge source is provided in `ditto-knowledge-proof/`.

a) (5%) flag1: Please draw a sequence diagram to explain how they perform the zero knowledge proof (3%) and get the flag (2%).
Hint: for sequence diagram, you can use any drawing tools, or some markdown language, reference: https://hackmd.io/@codimd/extra-supported-syntax#Sequence-Diagram.

b) (5%) flag2: You may notice that Carrol talks to Bob, and Bob will send the FLAG2 to Carrol once the verification is done. However, Carrol is going to travel and come back after the deadline of this homework. Obviously you cannot wait for Carrol as you need the FLAG2 ASAP. Can you pretend to be Carrol and trick Bob into thinking you have the private key?
Hint: Randomness generation is important in cryptography protocols.
Hint: You may want to find out what is Linear Congruential Generator (LCG). You can visit the following link to learning more details about LCG: https://en.wikipedia.org/wiki/Linear_congruential_generator

c) (5%) flag3: There is actually an Admin who also talks to Bob. However, Admin is very busy and doesn't always listen on a port. Admin also wants to make the protocol non-interactive so that he/she doesn't have to connect all the time just to wait for Bob to response the challenge, because he/she's really busy. However, the protocol seems to be insecure, can you briefly explain why it is insecure and how to fix it? (3%) Once you know why it is insecure and what attacks it is vulnerable to, try to get FLAG3 (2%).
Hint: You may want to have a look at the Admin source code `Admin.py`.

d) (Bonus 5%) flag4: You have successfully tricked Bob into believing that you are the Admin and that you have the private key, so he has decided to encrypt the secret FLAG4 using the ElGamal encryption with the corresponding public key, believing that you can decrypt it with the private key while preventing eavesdropping. Can you exploit Admin's private key to decrypt the ciphertext provided by Bob to obtain FLAG4?
Hint: find the order of the group. Can you factorize the order?

6. So Anonymous, So Hidden (15% + 5% Bonus)

a) (5%) You learned what mix network is in class. Suppose you are running a mix. Alice and other users will give you some packets, and it is your job to decrypt the packets and forward the packets to the next mixes. Access the service at `nc cns.csie.org 3001`, where you will get more instructions on how to interact with the service. You are also a great security engineer, and you know that timing analysis on the mix network is bad, so you want to make sure that there is less than a 1% chance that the adversary correctly guesses which incoming message corresponds to a specific outgoing message.

Note: The packets come aperiodically, and the plaintext message starting with `CNS` *would be sent when the challenge is over, either with a flag or a fail message.*

b) (5%) Suppose you are Alice. You want to ask Bob to give you the flag. Finish the TODOs in `mix-tor/lib.py` and access the service at `nc cns.csie.org 3002`. The server source code is available as `mix-tor/main2.py`.

Note: Despite its name, what the function `add_next_hop` *actually does is "prepend" a hop to path. The first hop the packet is sent to is the last hop added to the packet. The TA is a terrible security engineer, so no delay is applied in this subproblem.*

For subproblem (b), besides the script used for generating packets, also upload your implementation of `lib.py` and rename it to `code6b-lib.py`. You may not change any other files.

c) (Bonus 5%) Eve, a sneaky hacker, has somehow intercepted a message between Alice and the first mix. But she fell asleep in class so she knows nothing about cryptography. Can you decrypt the message for her? Eve is at `nc cns.csie.org 3003` waiting for you to contact her. You will receive partial credit for pointing out possible weaknesses in the writeup without being able to crack the ciphertext.

Note: There is no expected solution to this problem, although there are some intentional bad practices or designs in the implementation. You can perform any attack other than denial of service attacks.

d) (5%) Alice and Bob eventually learn that it is a bad idea to implement the security protocol themselves, so they decide to use a robust anonymous network with a sophisticated implementation. They decide to use Tor. Bob sets up a hidden service. The domain name looks so random that Bob thinks no one will ever find his cool service. Moreover, there are 65,535 ports, no one has time to try them all. He also published his public key online (at

`mix-tor/tor.pub`); after all, it is a *public* key. Find out about Bob's cool hidden service and steal his flag!