END SEMESTER EXAM

Name: Meetakshi Setiya
Roll No: 2019253

1.

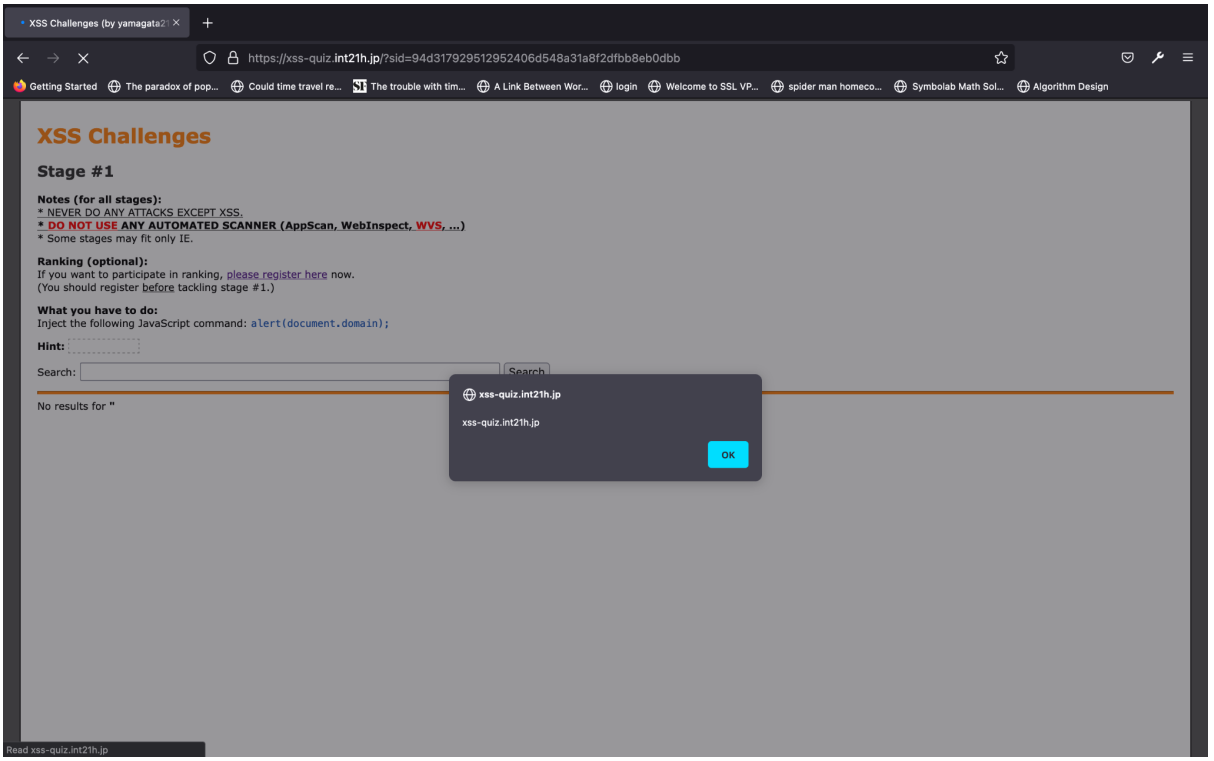
a. Code used:

```
<script>alert(document.domain);</script>
```

HTML component updated/used:

```
<input type="text" name="p1" size="60" value="">
```

Screenshot:



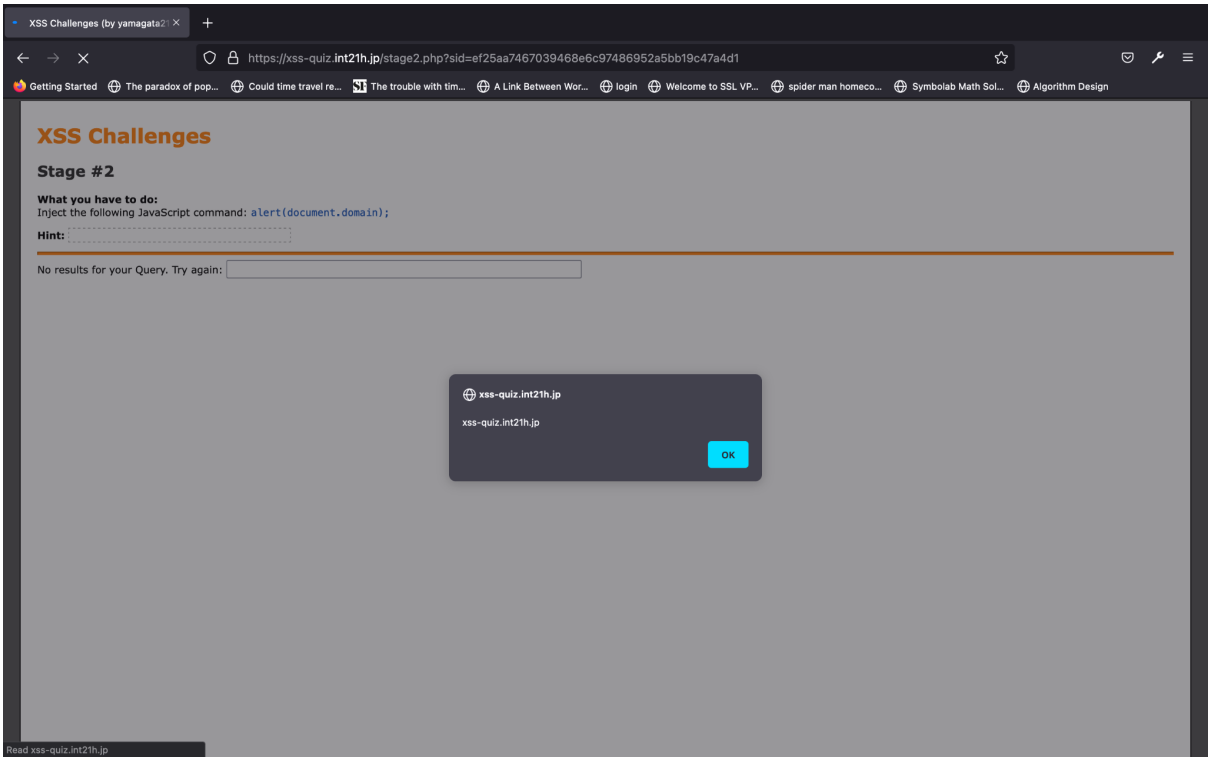
b. Code used:

```
"><script>alert(document.domain);</script>
```

HTML component updated/used:

```
<input type="text" name="p1" size="60" value="">
```

Screenshot:



c. Code used:

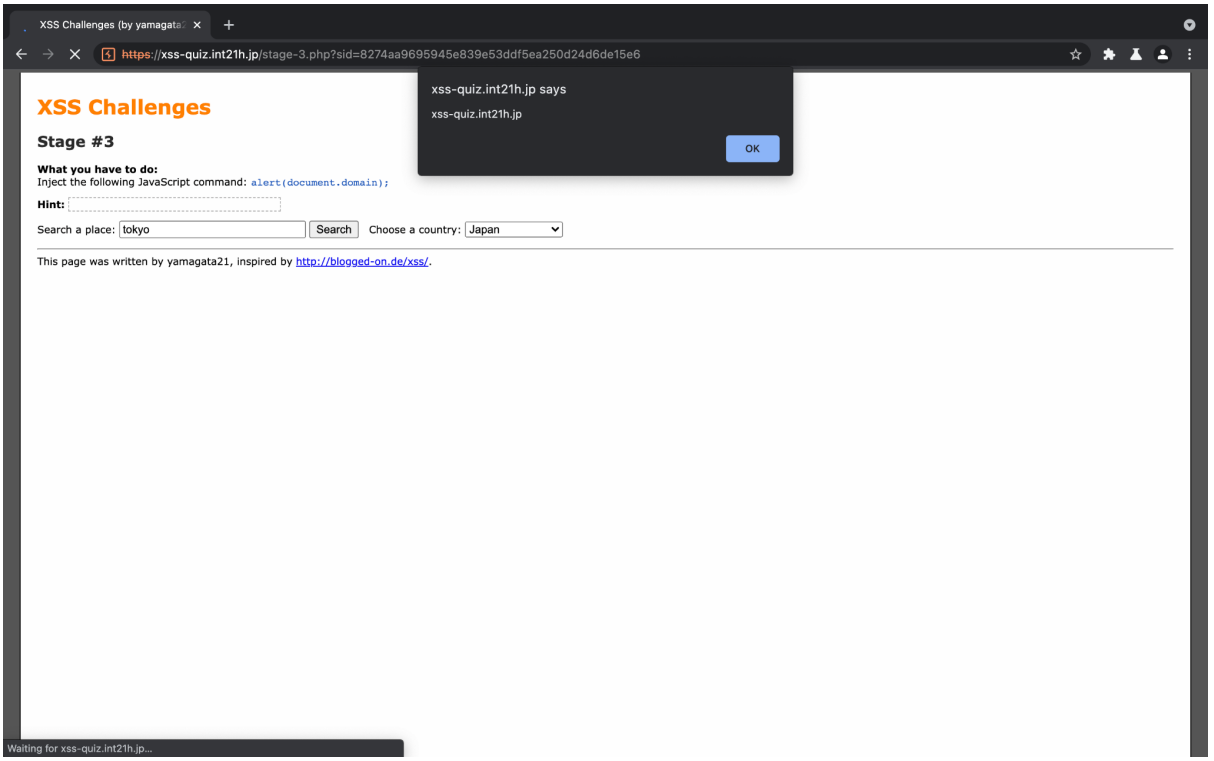
```
<script>alert(document.domain);</script>
```

HTML component updated/used:

```
<select name="p2">
<option>Japan</option>
<option>Germany</option>
<option>USA</option>
<option>United Kingdom</option>
</select>
```

Specifically, the component updated was `<option>Japan</option>`

Screenshot:



I modified the POST request using burp suite.

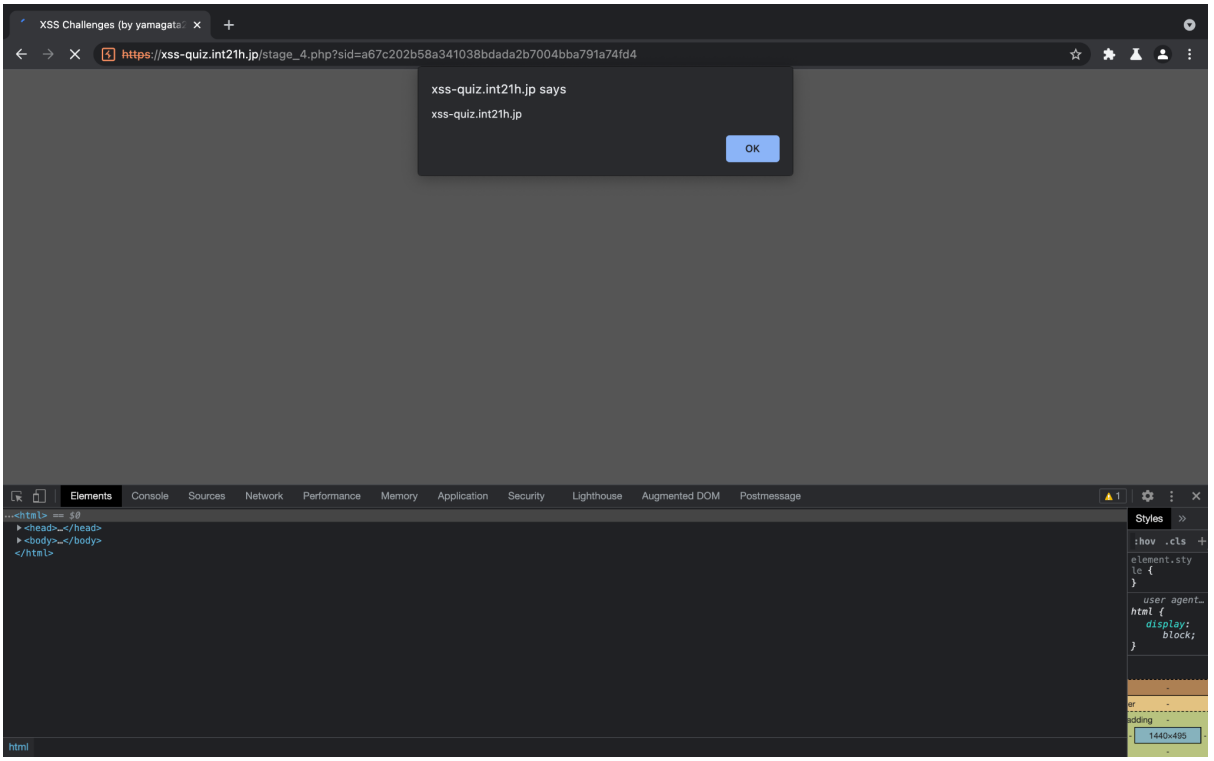
d. Code used:

```
"><script>alert(document.domain);</script>
```

HTML component updated/used:

```
<input type="hidden" name="p3" value="hackme">
```

Screenshot:



I modified the POST request using burp suite.

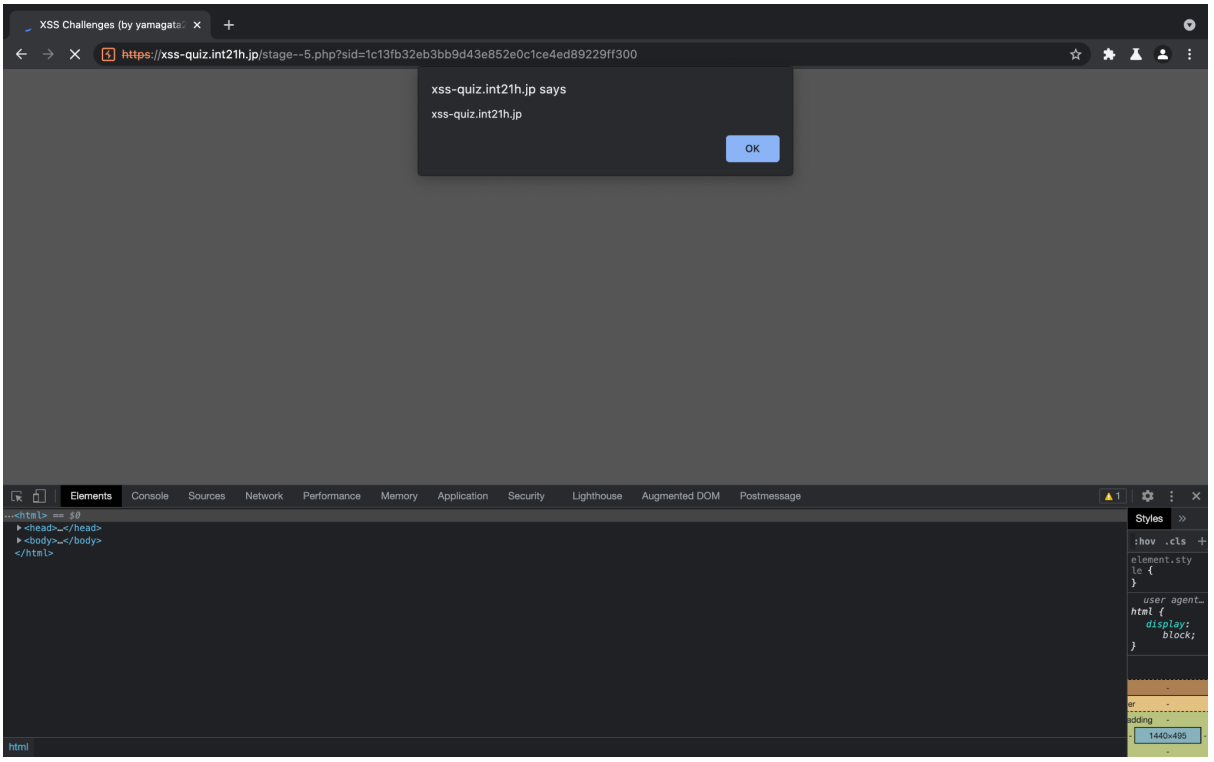
e. Code used:

```
"><script>alert(document.domain);</script>
```

HTML component updated/used:

```
<input type="text" name="p1" maxlength="15" size="30"
value="">
```

Screenshot:



I modified the POST request using burp suite.

2.

Note: I have assumed that 'disabling' an account permanently (until the admin team restores it after a formal application).

a.

Suppose a malicious user attempts to log into a legitimate user's account and purposefully enters the wrong password thrice. In this case, the application records three consecutive failed login attempts and disables the account. If the legitimate user wants to access their account now, they would not be able to do that. This is how the mentioned technique can prevent legitimate users from accessing the system.

The principle of least common mechanism states that mechanisms used to access resources must not be shared. The mechanism to access the user's account also makes use of the given 'account disabling' feature. The access mechanism to an account is actually shared between all users- both legitimate and unauthorized. Since this mechanism for accessing a user's account does not use multi-factor

authentication, IP-address based rate limiting, or any other technique to establish a difference between legitimate and unauthorized users, the action is a violation of the principle of least common mechanism.

b.

The principle of fail safe defaults states that unless a subject is explicitly given access to a resource they should be denied access to the resource.
According to me, the given example adheres to the principle of fail safe defaults. The principle requires that the default access to an object, unless explicitly stated, must be denied. Going by this logic, the known safe state is when default access to a user's account is denied to everyone- including the legitimate user.
Now, when a user successfully logs into their account, they are whitelisted and explicitly allowed to access their account (access based on permission or authorization).
When an attacker tries to log into a legitimate user's account and enters 3 wrong passwords, the system disables the user's account which basically means denying everyone access to that particular account, including the legitimate user. Thus, the system detects a fault in the login process here and automatically reverts back to its known, safe state (which is account access denied to everyone). This affects the system availability but in no case affects its safety, thus ensuring that the system fails securely.

3.

a. **Commandment 7:** Thou shalt not use other people's computer resources without authorization or proper compensation.
   Reason: It is unethical to use password breaking techniques to gain access to someone else's computer resources without permission.

b. **Commandment 6:** Thou shalt not use or copy software for which you have not paid.
   Reason: Regardless of motivation, pirating copyrighted proprietary software is illegal and unethical.

c. **Commandment 9:** Thou shalt think about the social consequences of the program you are writing or the system you are designing

Reason: Technical skills and computers must not be used to play petty pranks to disrupt classes and waste everyone's time. Besides, the code to hack into the institute's speackers is technically malicious.

d. **Commandment 7:** Thou shalt not use other people's computer resources without authorization or proper compensation.
Reason: The user is given the permission to use HPC only specifically for research work related activities. Instead, the user is using someone else's computational resources for personal use without authorization or permission. This is unethical.

e. **Commandment 3:** Thou shalt not snoop around in other people's computer files.
Reason: Snooping around other people's private information is disrespectful and an invasion of privacy.

4.

OWASP top 10 security risks and how we handled them:

1) **Broken Access Control**
We handled broken access control in the following ways:
   - ➔ Except public resources like product images, all other resources are denied by default (both frontend and backend).
   - ➔ The Cross-Origin Resource Sharing (CORS) policy has been properly configured by setting it to 'same origin' to restrict cross origin interactions between scripts, documents etc.
   - ➔ All the API routes are protected using middlewares
   - ➔ Web server directory listing is not possible and sourcemaps have been removed.
   - ➔ Rate limiting of 10 requests per second per IP address has been set on the Nginx web server to prevent automated brute force and other attacks.
   - ➔ Session tokens are randomly generated with high entropy and are invalidated as per idle timeout, absolute timeout and when the user logs out.

➔ URLs of seller proposal pdfs are random 128-bit unique strings with high entropy and not guessable.

2) **Cryptographic Failures**

We handled cryptographic failures in the following ways:

➔ Unnecessary sensitive data like credit card information etc used during payment (by Razorpay) is not stored anywhere.

➔ Bcrypt library is being used to hash and salt all plaintext passwords before storing them in the database. Passwords are not stored as plaintext.

➔ All data is encrypted by SSL-TLS since we are using HTTPS to serve all resources from our website. HTTP Strict Transport Security (HSTS) header has also been set to enforce encryption and prevent attacks like SSL stripping.

➔ Strong standard algorithm was used to generate private-public key pairs used in the self signed certificate.

3) **Injection**

We handled injection in the following ways:

➔ Prisma ORM was used instead of raw MySQL queries in the backend to eliminate any chances of SQL Injection.

➔ Next.js was used for developing the frontend. It sanitises inputs by default.

➔ Server side uses Joi to do input validation before doing any database queries.

4) **Insecure Design**

We handled insecure design in the following ways:

➔ OWASP cheat sheet for various implementations like authentication, access control etc was used and followed to ensure that the recommended secure design practices were adhered to.

➔ All sensitive data communication from server to client was done through POST requests only.

➔ All NPM libraries used were legitimate and latest.

➔ Multi-factor authentication is being used while registering new users.

➔ Virtual keyboard is being used while inputting token based OTPs to prevent keylogger attacks.

➔ The resource identifiers use uuid() and are unguessable and unique.

5) **Security Misconfiguration**

We handled security misconfiguration in the following ways:

- ➔ Production build was used for deployment on the VM.
- ➔ Rate limiting of 10 requests per second per IP address has been set on the Nginx web server to prevent automated attacks.
- ➔ No unused frameworks, libraries or dependencies have been installed.
- ➔ All ports except the ones being used by the application are closed.
- ➔ Appropriate security headers like HSTS, csrf-token, content security policy etc have been properly used.
- ➔ Environment variables were used to store API keys, access credentials to the database and other sensitive information for improved security. These were not managed by version control (git or github).
- ➔ No cloud service is being used.

6) **Vulnerable and Outdated Components**

We handled vulnerable and outdated components in the following ways:

- ➔ There are no unused dependencies, features or files.
- ➔ Npm package manager was used to manage all libraries and dependencies and to ensure that they come from official sources only.
- ➔ Latest version of Node, MySQL and Nginx is being used and these were downloaded from the official websites over secure links.

7) **Identification and Authentication Failures**

We handled identification and authentication failures in the following ways:

- ➔ Multi-factor authentication was used during Sign up and OTPs were used while doing sensitive actions like changing password or deleting a user's account.
- ➔ No default credentials were used in the production build for any resource or dependency.
- ➔ Strong password policy is being enforced on users while signing up and changing their passwords. The passwords have to be 12-22 characters long, must have at least one uppercase letter, one lowercase letter, one number and one special character.

- ➔ Proper error handling is done with standard error messages that are brief and do not go into too much detail or pinpoint the exact problem.
- ➔ Rate limiting of 10 requests per second per IP address has been set on the Nginx web server to prevent automated attacks.
- ➔ Session tokens are randomly generated with high entropy and are invalidated as per idle timeout, absolute timeout and when the user logs out.
- ➔ All of my group members log into the VM using public-private SSH keys only.

8) **Software and Data Integrity Failures**

We handled software and data integrity failures in the following ways:

- ➔ We have used a self-signed digital certificate to ensure the client that the data they are receiving is from the expected source and has not been altered.
- ➔ Npm uses trusted repositories only.
- ➔ The CI/CD pipeline is using github to access the code and has proper access control for code deployment.
- ➔ No data being sent to or received from the client is unencrypted.
- ➔ PM2 production process manager was used to ensure that the application stays alive, reloads without downtime and auto restarts after crashes.

9) **Security Logging and Monitoring Failures**

We handled security logging and monitoring failures in the following ways:

- ➔ Proper application side error handling and logging
- ➔ Nginx also maintains proper access and failure logs.
- ➔ PM2 also maintains error and out logs.
- ➔ Razorpay also logs all transactions and their status.
- ➔ Proper input validation prevents the possibility of injection attacks on the logs.

10) **Server-side Request Forgery**

We handled server side request forgery in the following ways:

- ➔ Proper firewall rules to block out all but essential intranet traffic.

- ➔ All client side data is being automatically sanitized by Next.js. Validation of data is also being done on the backend by Joi.
- ➔ HSTS is being used to prevent HTTP redirections and to enforce HTTPS.
- ➔ CORS policy has been used to whitelist only the website's origin to restrict cross origin interactions between scripts, documents etc.
- ➔ DNS rebinding attack is not possible since the IP address of the VM is being used to access the website directly.
- ➔ Attackers cannot access sensitive local files because they are not exposed in any way and not in the public folder.

5.

Situation where hiding information does not provide security:

Suppose an application uses a crackable hashing technique like MD5 to secure sensitive data like passwords. The software then appends a few predictable fix-length strings to the start and end of the generated hash to disguise the actual base hashing algorithm (MD5) and create a custom hashing algorithm. If the data about how these hashes are generated is kept secret, the system security works decently. However, once someone figures out the hash generation method i.e. knows how to generate the strings appended to the end of the hash and the base hashing method (MD5), the system will offer no protection at all. Thus, in this situation, the system is secure only as long as the attacker seems to not know about how the hashing algorithm works.

Situation where hiding information provides security:

An example of this situation would be a user hiding their private SSH key (or any cryptographic key) on their computer such that it is not shared with anyone else. The key, if generated with non-trivial, strong algorithms, is non-guessable and is private information that is meant to be kept secret by hiding it on the user's computer. Revealing the private key would let an attacker take over the user's account. In this scenario, hiding information provides security.

6.

Note: I have assumed that 'raw data from network' means unencrypted network packets (or encrypted packets that the research group has the permission to unencrypt) that are carried to the SSH department.

a.

If the research group wants to obtain all raw data from the network traffic to the SSH Department, the mentioned email policy would not allow that directly. The policy actually mentions that even though the emails are not private, the only way they can be read accidentally is while doing system maintenance ("or other ways stated in the full policy").

It is also mentioned that supervisors can examine their employees' mails only when they relate to the job but since IIITD is an educational institution and the research team would comprise professors and students only, there is no supervisor-employee or job system.

Thus keeping in mind the electronic mail policy, the following entails:

- Data collection would not be allowed at all to the research group because it has emails too.
- Data collection would be allowed but the SMTP traffic will be filtered.

b.

Updates that can be made to the privacy policy to allow the research group to collect data without abandoning the principle that emails should be protected are:

- For research purposes, electronic mail traffic would be shared only to IIIT Delhi's research groups and in no way can anyone outside IIIT Delhi be allowed or considered to access it.
- The investigators who wish to collect email traffic are expected to meet any one of the following criteria:
    - The investigator must be a faculty at IIIT Delhi working independently or in a team.
    - The investigator provides the name of the faculty at IIIT Delhi who is overseeing their research and is approved for access to the emails.
- The emails would either be properly and completely anonymized (see answer 9 part d example) or their contents would be hashed/encrypted before sending them to the research group.
- No research personnel is allowed to redistribute or share the email traffic with any third person unless the person concerned has applied and has been approved to access the information.

- The email collection should be kept secure at all times by password protecting the file(s).
- The research investigators would be held accountable individually and as a group to ensure proper use of the allowed email traffic access.

7.

The main characteristics of a good classical cryptosystem are as follows:

- Keys and the enciphering algorithm must not be complex. The key has to be transmitted and memorized so it must be short. If the enciphering algorithm is too complex, people won't use it.
- Implementation of the process must be as simple as possible.
- Errors in ciphering should not propagate and cause corruption of further information in the message
- The length of the cipher text must not be greater than the length of the plaintext input.

Caesar cipher is a simple cryptosystem that just uses an integer k from 1-25 as the encipherment key and 26-k as the decipherment key. The keys are easy to remember and the length of the ciphertext is equal to the length of the plaintext. Thus, caesar cipher is a classical cryptosystem (it possesses all the above mentioned properties).

Now, the main characteristics of a public key cryptosystem are as follows:

- It is computationally infeasible to derive the public key from the private key.
- It is computationally infeasible to determine the private key using a plaintext attack.
- It is computationally feasible and easy to encipher and decipher a message using the correct key.

If we try to compare the information above with the caesar cipher cryptosystem, we would find that it indeed is not a public key cryptosystem. The public key cryptosystem says that deriving private key from public key must not be possible, but if k is the public key for caesar cipher cryptosystem, anyone who knows k would immediately know that 26-k is the private key. Thus, the keys can be derived from each other. Public key cryptosystems must be immune to plaintext key-guessing

attacks. In Caesar cipher, plaintext and ciphertext are of the same size and any changes made to one character in the plaintext is predictable in the ciphertext. Thus, if Caesar cipher is used, it is very easy to catch a pattern in the plaintext and the ciphertext available to the attacker and to guess the encipherment and decipherment keys.

Hence, caesar cipher is a classical cryptosystem and not a public key cryptosystem.

8.

a.

A cryptographic checksum is a mathematical value generated by performing a series of complicated mathematical operations (called a cryptographic hash function) on a piece of data, like a single file. The cryptographic checksum converts the file or data to a string with a fixed number of digits called a checksum. Without knowing the cryptographic algorithm used to create the checksum, it is highly infeasible to change the contents of the original file or data without modifying the corresponding checksum as well. Thus, checksums are used to ensure integrity of the files or data after it has been transmitted or shared. This can simply be done by comparing checksum computed from the received file with the one that was transmitted by the source of the file or the data.

The identity function is not a good checksum function because of the following main reasons:

- It does not produce a fixed-size string as the checksum.
- It is extremely reversible. Thus, the original data/file can be modified along with the checksum by any attacker.
- Most importantly, a good cryptographic checksum must ensure that small changes to the data create large, seemingly unpredictable changes to the checksum. The identity checksum just reflects any change in the input file/data as is.

b.

XOR is not a good checksum function either because it cannot fully ensure the integrity of the input data due to the following reasons:

- XOR does not care about the order of the inputs. Thus, swapping of two or more words in the original data/file would not be detected. Example:

$$10101 \oplus 10010 = 00111$$
$$10010 \oplus 10101 = 00111$$

- It does not detect bit errors/flips if they are even in number and lie at the same bit position in the checksum computational chunk. In fact, if the length of the code words goes to infinity, and *n* is the checksum size in bits, then the probability that a 2 bit error is undetected is 1/n which is a poor performance. An example of undetected 2 bit flips is given below:

$$10101 \oplus 10010 = 00111$$
$$10100 \oplus 10011 = 00111$$

- Any transmission error that causes even a single bit of the message to flip would be regarded as an incorrect checksum. It has no tolerance for genuine network errors. An example is given below:

$$10101 \oplus 10010 = 00111$$
$$10101 \oplus 10011 = 00110$$

9.

To find the level of anonymity, I used this source:
https://link.springer.com/chapter/10.1007/978-3-642-80350-5_4

Level of anonymity in each level of sanitization:

a. Simple Sanitization: In this level of sanitisation, all information except the names of the commands that have been issued are deleted. Information like the user, input arguments of the command, or the results of the commands is never shown. This method of sanitization, while bolstering anonymity, removes too much useful information. For example, an intruder may have viewed the '/etc/passwd' file using 'cat' and all that simple sanitation logs of this activity is the 'cat' command with no information about the file name or other arguments whatsoever. Thus, it is impossible to determine if the action was as harmless as fixing a spelling error or as dangerous as editing the passwords file. The level of anonymity is 0 (no identification of user).
   An example:
   ls
   cat

nano

b. Information-Tracking Sanitization: This method tracks sensitive information through log files. Since the sanitizer keeps a table of sensitive information that must be removed or replaced, this method can actually be used to retain and note all references to a particular file or user without revealing their exact names. This sanitization method can also record the outputs of system commands like 'who' and 'finger' that have a consistent output format. Thus, the output/log here is more useful and also preserves anonymity by suppressing sensitive information. The level of anonymity is 2 (Pen-name Identification).
   An example:
   SYSTEM1% ls ~/USER1
   SYSTEM1% cat FILE1
   SYSTEM1% nano FILE1

c. Format Sanitization: This process of sanitization suppresses as little information as possible and even handles sanitization of different data formats like uuencoded files, and not just text files by locating segments of information that have been reformatted. The level of anonymity is 2 (Pen-name Identification).

d. Comprehensive Sanitization: This sanitization method properly examines every segment of the log file and even the source code. Text files like emails also have their headers and their body sanitized which includes replacing domain names and usernames with generic ones. Additionally, this process also allows the sanitization of proprietary information. All the previous sanitization methods would completely remove proprietary data, but this sanitization method can actually minimize this information loss by creating a custom dictionary of commonly used words to assist in identifying and sanitizing sensitive data. The level of anonymity can be 2 (Pen-name Identification).
   An example:
   SYSTEM1% ls ~/USER1
   ~Example Mail, not in canon format, just for demonstration purposes~
   From: USER3@SYSTEM2.DOMAIN5 (PERSON1)
   Message-Id: <12356478.DD2345@SYSTEM2.DOMAIN5>

To: USER2@SYSTEM4.DOMAIN2
Subject: Important
Status: R

PERSON4,
Hello, please contact me at my new email address USER13@DOMAIN3 regarding the status of our meeting.
Thanks,
PERSON1

Each level of sanitization can in fact, be automated with varying degrees of ease of automation and the requirement of human oversight.
Simple Sanitization has the highest ease of automation because of its very straightforward implementation. Little to no human oversight is required in Simple Sanitization.
Information-Tracking Sanitization would require a little human intervention especially for creating the symbol table used by the sanitizer. After this, the entire sanitization process can be pretty much automated. The automation would be a bit complex since the data has to be replaced by appropriate identifiers during the sanitization process.
Format Sanitization is more complex to automate since it is difficult to create the necessary patterns indicating the beginning and ending of the data. Also, it is possible that a segment is compressed multiple times. Thus, it must be explicitly specified and decided by a human how many times the sanitizer will process a segment before it is deleted.
Comprehensive Sanitization is difficult to fully automate because of its complex nature and requires the most human assistance. This includes building a custom dictionary of sensitive words as well as providing some method of determining what is considered sensitive information and what is not. For example, the word ‘Winter’ occurring in a username like ‘Bethany Winter’, would have to be sanitized, but if it occurs in a sentence like ‘I like the winter season’ does not have to be sanitized. Using a natural language parser can help assist this task but at the beginning, the process would be heavily user assisted since the user would be asked to respond to prompts whenever the sanitizer comes across unfamiliar data. As the sanitizer learns along the way by encountering new information, only occasional user assistance would be required. Nevertheless, the sanitizer would always require some degree of assistance and direction.

10.

a.

Let K be a random variable that represents the number of emails the attacker has to send to the Cypherpunk service before all mails in the original pool have been sent.
Total mails in the pool at a time = $n-1$

After $i-1$ mails from the original pool have already been sent, let k_i be the number of mails to be sent by the attacker so that the i^{th} mail from the original pool can be sent.
Remaining mails from the original pool in the modified pool = $n-i$

The probability mass function (PMF) of k_i would be:
$$= \left(\frac{i}{n}\right)^{k_i-1}\left(\frac{n-i}{n}\right)$$
Which can be rewritten as:

$$\left(1-\frac{n-i}{n}\right)^{k_i-1}\left(\frac{n-i}{n}\right)$$
Examining the above expression clearly implies that k_i has a geometric distribution.

Therefore, the probability $p_{k_i} = \frac{n-i}{n}$

And, the expected value of k_i can be written as:
$$E(k_i) = \frac{1}{p_{k_i}} = \frac{1}{\left(\frac{n-i}{n}\right)}$$
$$E(k_i) = \frac{n}{n-i}$$

Since we want to remove all n-1 emails from the original pool, i will take the values $\{1, 2, .., n-1\}$

Thus, the random variable K can be written as:

$$K = k_1 + k_2 + \ldots + k_{n-1}$$

$$K = \sum_{i=1}^{n-1} k_i$$

The expected number of messages $E(K)$ that the attacker would have to send to achieve their goal would be computed as:

$$E(K) = E\left(\sum_{i=1}^{n-1} k_i\right) = \sum_{i=1}^{n-1} E(k_i)$$

$$E(K) = E(k_1) + E(k_2) + \ldots + E(k_{n-1})$$

$$E(K) = \frac{n}{n-1} + \frac{n}{n-2} + \ldots + \frac{n}{1}$$

$$E(K) = n\left(\frac{1}{n-1} + \frac{1}{n-2} + \ldots + \frac{1}{1}\right)$$

Or,

$$E(K) = n\sum_{i=1}^{n-1} \frac{1}{i}$$

This is the required answer.

b.

Assuming that the attacker monitors all incoming and outgoing traffic from Cypherpunk remailer, they would just have to wait for the n-1 original mails to be sent which can be seen from the network traffic. The attacker can send some unique identifier with their mails or set a unique header that is non-duplicatable and can help them differentiate their emails with the rest of the email traffic. Now, the attacker just has to simultaneously look out for outgoing emails (that are not theirs) while flooding the server. Once the attacker counts n-1 such emails, they would know that the original n-1 messages that were in the pool have been sent.

Additionally, if some x other emails arrive at the Cypherpunk while the attacker is flooding the server, the attacker, who is monitoring the incoming traffic would just have to look out for those additional x emails in addition to the original n-1 emails to go out in the outgoing traffic.