PART_I: Crypto

1. Why is the one-time pad difficult to use in practice?
2. How does the man-in-the-middle attack work?

PART_II: Web Security

1. **Web attacks.** Consider a fictitious social networking site called OurPlace. OurPlace has millions of users, not all of whom are particularly security-conscious. To protect them, all pages on the site use HTTPS. (20 points)

(a) OurPlacehomepage has a “Delete account” link which leads to the following page:

```
<p>Are you sure you want to delete your account?</p>
<form action="/deleteuser" method="post">
  <input type="hidden" name="user"
  value="{{username}}"></input>
  <input type="submit" value="Yes, please delete my
account"></input> </form>
```

(The web server replaces {{username}} with the username of the logged-in user.)

The implementation of /deleteuser is given by the following pseudocode:

```
if account_exists(request.parameters['user']):
    delete_account(request.parameters['user']) return '<p>Thanks for
    trying OurPlace!</p>'
else:
    return '<p>Sorry,' + request.parameters['user'] + an error
    occurred.</p>'
```

Assume that the attacker knows the username of an intended victim. What's the simplest way the attacker can exploit this design to delete the victim's account?

(b) Suppose that /deleteuser is modified as follows:

```
if user_is_logged_in(request.parameters['user'],
    request.cookies['login_cookie']):
    delete_account(request.parameters['user']) return '<p>Thanks for
    trying OurPlace!</p>'
else:
    return '<p>Sorry,' + request.parameters['user'] + an error
    occurred.</p>'
```

(Assume login_cookie is tied to the user's account and difficult to guess.)

Despite these changes, how can the attacker use CSRF to delete the victim's account?

(c) Suppose that the HTML form in (a) is modified to include the current user's login_cookie as a hidden parameter, and /deleteuser is modified like this:

```
if request.parameters['login_cookie'] == request.cookies['login_cookie']
and user_is_logged_in(request.parameters['user'],
request.cookies['login_cookie']):
delete_account(request.parameters['user']) return '<p>Thanks for trying
OurPlace!</p>'
else:
    return '<p>Sorry,' + request.parameters['user'] + an error
    occurred.</p>'
```

The attacker can still use XSS to delete the victim's account. Explain how.

2.You have located a SQL injection vulnerability in a string parameter. You believe the database is either MS-SQL or Oracle, but you can't retrieve any data or an error message to confirm which database is running. How can you find this out? (4 points)

3. You have submitted a single quotation mark at numerous locations throughout the application. From the resulting error messages you have diagnosed several potential SQL injection flaws. Which one of the following would be the safest location to test whether more crafted input has an effect on the application's processing? (4 points)

(a) Registering a new user

(b) Updating your personal details

(c) Unsubscribing from the service

4. You have found a SQL injection vulnerability in a login function, and you try to use the input ' or 1=1-- to bypass the login. Your attack fails, and the resulting error message indicates that the -- characters are being stripped by the application's input filters. How could you circumvent this problem? (4 points)

5. You have found a SQL injection vulnerability but have been unable to carry out any useful attacks, because the application rejects any input containing whitespace. How can you work around this restriction? (4 points)

6. The application is doubling up all single quotation marks within user input before these are incorporated into SQL queries. You have found a SQL injection vulnerability in a numeric field, but you need to use a string value in one of your attack payloads. How can you place a string in your query without using any quotation marks? (4 points)

PART_III

1. Define a denial-of-service (DoS) attack.
2. List four characteristics used by firewalls to control access and enforce a security policy.