Handwriting

1. **SYN Cookies**

   1) 在沒有用cookies的情況下，server在送出ACK之後便會把connection加入到自己的記憶體，並一直等待client回傳ACK，因此會有資源佔用的問題。但是cookies只會根據SYN request的state來計算，並不會在連線建立以前儲存requeset的state，所以不會有佔用的問題，也不會受到SYN flood的影響。
   2) 包含client的IP address是為了之後能夠辨識不同的client，以提供適當的服務，或者是在某位client有不法行為時，能有所反應。而timestamp則是用來限定client必須在規定時間內回應（ACK），並且也能夠讓server更有效的管理cookies，像是設定expire的時限。
   3) Hash function實際上不能提供隱密性，畢竟只要有需要的東西，任何人都能夠計算出cookie；而MAC則有加密的步驟，因此即使能夠知道server跟client的IP, port number，也很難輕易的取得對應的cookies。

2. **BGP**

   1) During the forwarding, the IP with the longest prefix matched will be selected. Thus, if AS999 announces 10.10.220.0/24, prefix hijacking could happen.
   2) a) One possible solution is {10.10.220.0/24, [AS999, AS2, AS1, AS1000]}.
      b) In Figure 3, the routes of AS3, AS4, and AS5 are different from those in Figure 1. Since the IP announced by AS999 has more prefix matched, it is a more preferable choice for AS3, AS4, and AS5. On the other hand, the route announced by it contains AS1 and AS2. According to the policy of loop prevention, AS1 and AS2 won't choose it. Therefore, Figure 3 is generated.
   3) - **Advantage:**
      This mechanism does help to prevent prefix hijacking by pruning malicious advertisement. Besides, since the routers just check less bits of IP, the overheads can somehow decrease.
      - **Disadvantage:**
      If the value of MPL is too small, it could somehow limit the number of valid advertisement because the number of possible and valid announcement of IP range will decreases.

3. **Knowing What I Am, Not Knowing Who I Am**

   1) Since the user only sends the w' it gets to the server apart from the initial request, it is the only way the adversarial server knows who the user is. Thus, it should make different user return different w' or it can only randomly guess who the user is, which leads to $Pr[Expt^{anon}_{\prod,\mathcal{A},l}(n) = 1] = \frac{1}{l}$. However, if it encrypts different w for different user, the user will view it as $\mathbf{cheat}_S$ after receiving $c_1, ..., c_l$. Therefore, it is impossible to make $Pr[Expt^{anon}_{\prod,\mathcal{A},l}(n) = 1] > \frac{1}{l}$ so we can say that this protocol achieves verifiable anonymity.
   2) If we split users into several groups and each group contains at most k users. Each time a user wants to connect to the server, the server only has to send at most k ciphertext to the users which are encrypted with the public keys from the same group. In this way, the server encrypts at most k messages each time a new connection comes. However, this could cause that malicious server has chance to target a specific group or a user.

3) We can assume that the delivery of keys from different clients isn't simultaneous, and some of them could be much earlier than others. With this assumption, a malicious server can start the authentication process once it receives some but not all keys. If it wants, it could even initiate the process each time it gets a new key. In this way, the server won't be viewed as cheating and can split users, which helps it distinguish them. Besides, if necessary, the server can forge numerous pairs of keys to deceive the users during the authentication.

Capture The Flag

4. **TLS**

   Flag: CNS{ph4Ul7y_K3y_93n3R471oN_15_D4N93rOU2!}

   First, with wireshark, I found N and e of RSA, and compute the value of p, q, and d with online tools. (The reference is shown in code.) Then, I add it to the wireshark to decrypt message.
   After that, I found the information we need to generate our certificate, including the public key of server and the needed information for certificate. I ran following command to get the certificate.

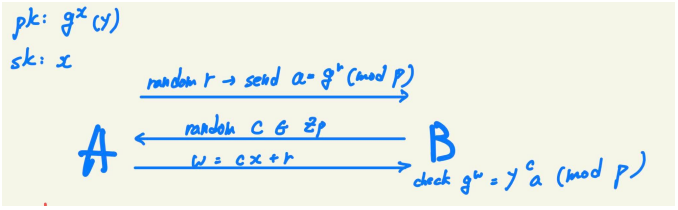
```
# generate csr
#(TW / Taiwan / Taipei / NTU CNS / VIP / VIP / cns@csie.ntu.edu.tw)
openssl req -new -key rootKey.key -out request.csr
# generate root certificate
# (TW / Taiwan / Taipei / NTU CNS / ROOT / ROOT / cns@csie.ntu.edu.tw)
openssl req -x509 -new -key rootKey.key -out root.crt
# get eve.crt:
openssl x509 -req -in request.csr -CA root.crt \
-CAkey rootKey.key -CAcreateserial -out eve.crt -days 365
```

   Then, the following command can connect to server.

```
# connect to server:
openssl s_client -connect cns.csie.org:12345 -cert eve.crt -key rootKey.key
# enter Alice413, dogsarecute, Flag...plzzzzz...
```

5. **Little Knowledge Proof**

   a) CNS{Man_1n_4he_M1dd1e_a44ack}



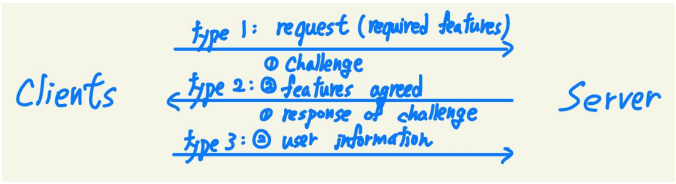
   I implement the Man-in-the-middle attack to get the flag.

   b) CNS{r&omne$$$hould_B_unp#ff0000ic\\tle}
   Since the LCG of Alice isn't totally random but generate r in a fixed order each time we connect to the server, we can simply compute x by sending different c for the same r/a.

c) CNS{CDH_f@!l_wHEn_tHE_'_is_uns@Fe}
The Pohlig-Hellman algorithm is very useful in practice if the order of the group in which we would like to solve a given discrete logarithm problem is smooth, that is it has only "small" prime divisors. Thus, I utilize the online tools from https://shrek.unideb.hu/~tengely/crypto/section-6.html#p-204-part9 to solve w and get the flag.

6. **Clandestine Operation II**

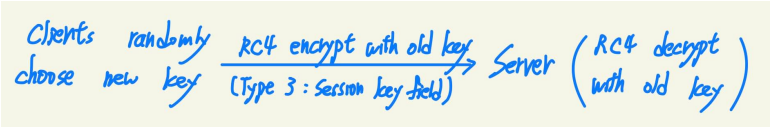
a) The diagram of messages exchange is as follow:



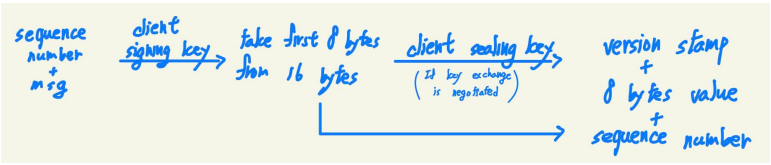
The three types of messages is listed below:

- Type 1: Negotiation ⇒ The client indicates supported options via the flags.
- Type 2: Challenge ⇒ The server sends challenge to test if the client knows the password.
- Type 3: Authentication ⇒ The client responds to the challenge to prove his knowledge about the password without leak the information of password and provides his domain or server name and username if necessary.

b) The following figure shows the rough process for key exchange.



The following figure shows the rough process of signing. Clients use client signing/sealing key while server use server signing/sealing key. All Signing keys are generated from the unweakened master key.



7. **So Anonymous, So Hidden**

a) CNS{H3Y_Y0u_Ar3_A_m1x3R_ma573R}
Since we have to make sure that there is less than a 1% chance that the adversary correctly, I buffer 101 packets and shuffle them before I send them to their destination. guesses which incoming message corresponds to a specific outgoing message

b) CNS{1_8r04dc457_7H15_m59_51nCe_y0U_D1dn7_91vE_7He_re7uRN_4ddRE55}
First, I implement *create()* to make a packet with the information of "Give me flag, now!". In this function, data would be encrypted with *StreamCipher* and *PublicKeyCipher* sequentially. Then, I called *add_next_hop()* several times to prepend the targets one by one to the data.