Assignment 3 - Part 2 - Report

Joseph May

1. **PayMe!**:

a. A nonce is a value that is used only once, typically it is a random number. The purpose of a nonce in a protocol is to ensure that each transaction request is unique and two requests will not be identical. If someone were to send a new message with a new nonce, attackers cannot simply resend an old message to trick the server into duplicating an earlier transaction.

   The two main attacks that a nonce helps to prevent are replay attacks and reflection attacks. For replay attacks an attacker that obtains a valid transaction without a nonce could resend that message to the server and cause it to process that transaction again. A nonce prevents this as the server would detect that the message is old or that the nonce is incorrect. Reflection attacks are prevented as if the attacker tries to trick the server into reflecting a message, the server will again notice that the nonce is being reused as with a nonce, each request is unique.

b. In the protocol, the digital signature only covers X and n but not the entire message including the recipient's identity B. Since B is not signed if an adversary intercepts the transaction before it reaches the server, it could change the recipient's field to its own identifier.

   Now if the adversary sends the updated transaction to the server with its own identifier, the server will not be able to detect that the recipient was altered as the recipient is not included in the digital signature and the money would be sent to the adversary instead of the original destination. To fix this issue the receiver would also need to be included in the digital signature in order for the server to verify the destination.

2. **PompousPass™** :

a. No, this system is not secure. The issue with this system is that the encryption of the password, ID and nonce is done using the user's private key. The ID and the nonce are sent over plaintext as well. Since public keys are known to everyone, this means that anyone that intercepts the message can use the public key to decrypt the user's password. By raising $E_{x^{-}}(Idx|Pwx|r)$ to the users public key, anyone can obtain (Idx|Pwx|r). Since the nonce and the Id are sent by plaintext as well, anyone can obtain the user's password.

b. In a revised protocol to make the system secure, the user must first send its Id, then the server must challenge the user, and finally the user responds with its password that is encrypted with the server's public key.

User sends its Id to the server
x→S: Idx
Server sends challenge nonce r back to the user, r ensures protection against replay attacks
S→x: r
The user responds with two things, first the password and the nonce encrypted using the server's public key, meaning that only the server can decrypt the password ensuring that it is secure. Secondly it sends its Id and the nonce as a digital signature encrypted with the user's private key, meaning that only the user could have sent this.
x→S: ES+(Pwx || r), Sigx−(Idx || r)
The server can use its own private key to check if the password is a match, and also use the user's public key to verify the digital signature.

c. The reason that using public key cryptography, is most often used to distribute session keys in practice is for the fact that public key cryptography (asymmetric) is much more computationally expensive then symmetric cryptography. Key distribution is one of the biggest challenges that symmetric cryptography faces and public key cryptography simplifies this. Once the session key is distributed, a fast and efficient symmetric cipher such as AES can be used to encrypt the data giving better performance than public key cryptography.

3. **Kerberos-ish**:

a. Yes, the ChompChomp mechanism achieves client authentication. When the client sends the service the ticket, the service will decrypt it using its long-term key and recover the session key $K_{c,v}$ and the client's identity. Since the client encrypts ID_c,IP_c, and t under Kc,v, the service knows that whoever sent the message must know $K_{c,v}$. And the only party who should know $K_{c,v}$ (besides V) is the legitimate client, to whom the KDC sent it. Thus, the service is assured it is talking to the same entity the KDC designated as ID_c.

b. Yes ChompChomp achieves service server authentication. When the service replies to the client with $E_{Kc,v}(t+1)$ this informs the client that the server must know $K_{c,v}$ as it encrypted t+1 with it. If the service knows $K_{c,v}$ then it must know that the service could decrypt the $ticket_V$ and hence knows K_V. This means that the service is authenticated back as it must know the service's private key.

c. The principal goal of Kerberos that ChompChomp does not achieve is providing single sign-on via a separate ticket granting service. With Kerberos once a user obtains a Ticket-Granting ticket (TGT) from the KDC, the user can request service tickets from the TGS without having to re-enter or re-expose a long term secret such as their password or key, each time. With ChompChomp there is no TGS meaning that the client must always go back to the KDC with each new service request. This forces frequent exposure of their long term key and loses the convenience that SSO brings.