Computer Networks Laboratory – Assignment 11

Subhash S – 111801042
Computer Science and Engineering, IIT Palakkad

Generating public and private keys for users A and B

File: q1.py and utils.py

Aim

To generate public, private key pairs for users A and B.

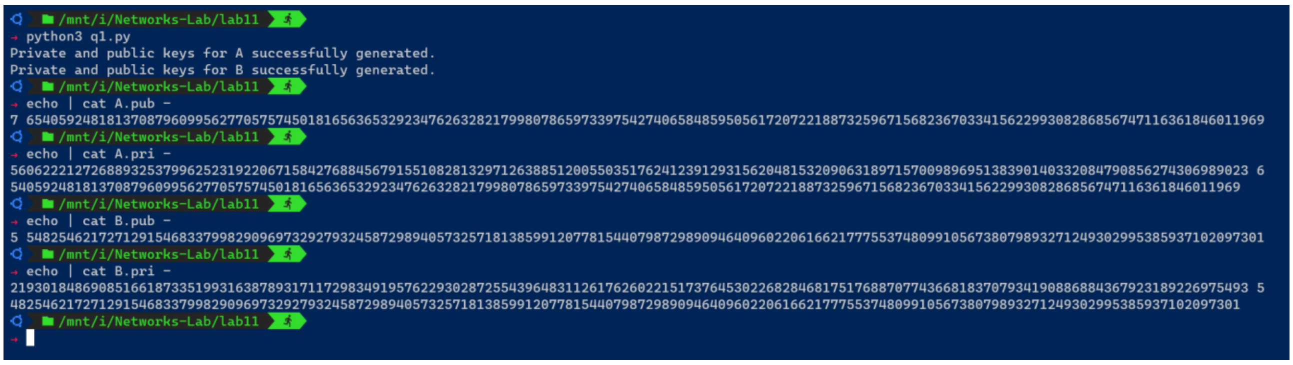
Functions

1. *is_prime:* Given a number n it checks if the number is prime.
2. *generate_prime:* Give the number of bits n, it generates a random n bit prime number.
3. *read_public_key* and *read_private_key:* Used to read public and private keys of a user respectively.

Program Flow

1. I generate two large prime numbers p and q of size 256 bits each, this ensures that the modulus $n = p \times q$ comes to 512 bits.
2. Now I store the modulus p x q in the local variable n.
3. I compute the totient of n i.e. $\varphi(n) = (p - 1) \times (q - 1)$
4. Now I find the public exponent $(e, 5 < e < \varphi(n))$ iteratively by starting from 5 and incrementing by 1 until it is relatively prime to the totient computed in 3.
5. I find the modular inverse of $e \equiv \varphi(n)$
6. Now the public key pair is (e, n) and the private key pair is (d, n).

Figure 1: The contents of all the four files after q1.py was run



```
 /mnt/i/Networks-Lab/lab11
→ python3 q1.py
Private and public keys for A successfully generated.
Private and public keys for B successfully generated.
 /mnt/i/Networks-Lab/lab11
→ echo | cat A.pub -
7 6540592481813708796099562770575745018165636532923476263282179980786597339754274065848595056172072218873259671568236703341562299308286856747116361846011969
 /mnt/i/Networks-Lab/lab11
→ echo | cat A.pri -
5606222127268893253799625231922067158427688456791551082813297126388512005503517624123912931562048153209063189715700989695138390140332084790856274306989023 6
540592481813708796099562770575745018165636532923476263282179980786597339754274065848595056172072218873259671568236703341562299308286856747116361846011969
 /mnt/i/Networks-Lab/lab11
→ echo | cat B.pub -
5 5482546217271291546833799829096973292793245872989405732571813859912077815440798729890946409602206166217775537480991056738079893271249302995385937102097301
 /mnt/i/Networks-Lab/lab11
→ echo | cat B.pri -
2193018486908516618733519931638789317117298349195762293028725543964831126176260221517376453022682846817517688707743668183707934190886884367923189226975493 5
482546217271291546833799829096973292793245872989405732571813859912077815440798729890946409602206166217775537480991056738079893271249302995385937102097301
 /mnt/i/Networks-Lab/lab11
→ 
```

Sign and Encrypt the message using the keys generated in q1

File: q2.py and utils.py

Aim

To sign and encrypt the message using the keys generated in the previous questions so that it can be sent to be in a secure manner.

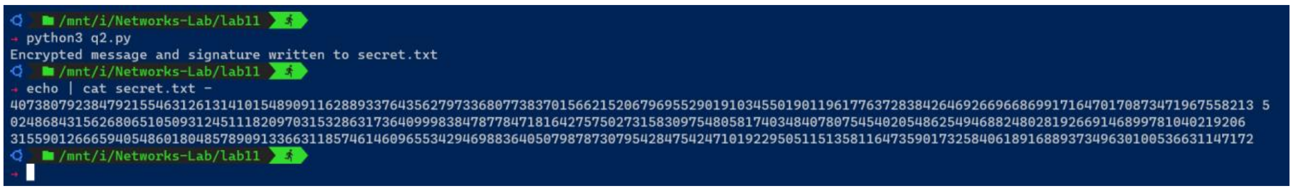
Functions

1. *msg2blocks:* Splits any message of arbitrary length into blocks of size k, where $2^k < n$ and returns an iterator over it.
2. *hash2int:* Given a string s, it converts the ASCII string into an integer by converting each character from ASCII to its integer value.
3. *encrypt:* This function is used to encrypt the message m by performing the operation $m^e \equiv n$ where m is the message, e is the public exponent and n is the modulus.
4. *sign:* This function is used to sign the message by performing the operation $s^d \equiv n$, where s is the message digest and d, is the secret exponent.

Program

1. I read the private key of A and the public key of B since they are used in encrypting the message and signing the message respectively.
2. I read the message from the file *message.txt.*
3. Since the message can be of any length, I divide it into blocks of appropriate size using the *msg2blocks* function mentioned above.
4. For each of the blocks,
    a. I encrypt the block using the *encrypt* function mentioned above.
    b. I append this encrypted message to the variable *encrypted_msg* which accumulates the final message from each of the blocks.
5. Now the encryption is done and hence the message has to be signed.
6. To generate the message digest, I use the SHA-1 hashing algorithm since I'm generating prime numbers of 256 bits each and SHA-1 has an output length of 160 bits, this ensures that there are no issues while encrypting.
   Note: I could have used the SHA-256 which has an output length of 256 but in that case, I would have to generate 1024 bit RSA keys which would increase the running time. Hence, for demonstration, I chose to go with SHA-1.
7. Now using the *sign* function mentioned above, I sign the above-generated message digest.
8. I write the encrypted message and the signature in two separate lines of the file *secret.txt*.

Figure 2: Partially shows the content of secret.txt after running q2.py



```
 /mnt/i/Networks-Lab/lab11
→ python3 q2.py
Encrypted message and signature written to secret.txt
 /mnt/i/Networks-Lab/lab11
→ echo | cat secret.txt -
4073807923847921554631261314101548909116288933764356279733680773837015662152067969552901910345501901196177637283842646926696686991716470170873471967558213 5
024868431562680651050931245111820970315328631736409998384787784718164275750273158309754805817403484078075454020548625494688248028192669146899781040219206
3155901266659405486018048578909133663118574614609655342946988364050798787307954284754247101922950511513581164735901732584061891688937349630100536631147172
 /mnt/i/Networks-Lab/lab11
→ 
```

In the ***cat*** command of the above image, the space-separated integers represent the various blocks of the encrypted message and the last line represents the signature of the message that was signed using the private key of A.

Verifying signature and decrypting message generated in q2

File: q3.py and utils.py

Aim

To verify the signature generated in q2 and to decrypt and print the encrypted message from q2 if the signature is valid.

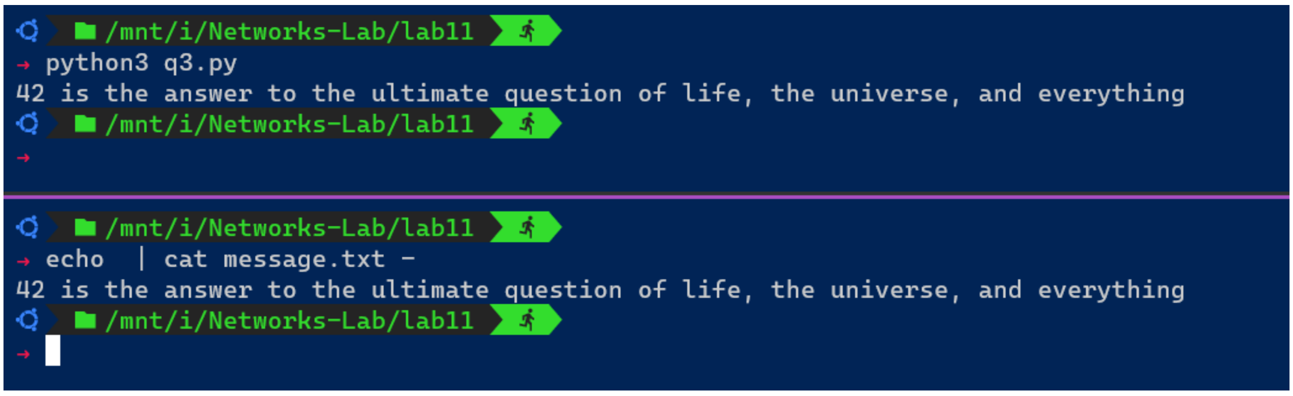
Functions

1. *bin2str:* Given a binary string s, it converts the binary string to ASCII characters.
2. *decrypt:* This function is used to decrypt the encrypted message c by performing the operation $c^d \ \equiv n$ where m is the message, d is the secret exponent and n is the modulus.

Program

1. I read the private key of B and the public key of A since they are used in decrypting the message and verifying the signature of the message respectively.
2. I read the contents of the file *secret.txt* that was generated in q2 and it contains the encrypted message and signature in two different lines.
3. The first step is to try and encrypt the message and since q2 separated each block by a space, for each of those encrypted blocks,
    a. I decrypt the block using the *decrypt* function mentioned above.
    b. I convert the decrypted message to a binary string and append this to the variable *decrypted_msg* which accumulates the final message from each of the blocks.
4. Now the *decrypted_msg* is a binary string and using the function *bin2str* mentioned above, I convert it to a string.
5. If the decoding fails during conversion from binary to string, the message was tampered and hence I exit the program with an error message.
6. If the message was successfully decrypted, I proceed to verify the signature to check if this message was sent by A.
7. I generate the SHA-1 has of the decrypted message obtained in 4.
8. I also decrypt the second line of the file *secret.txt* which contains the signed message digest from A. This decryption gives the SHA-1 has that was generated by A.
9. Now, I verify the signature by checking if the hash generated in 7 and 8 are the same, this ensures that the message was sent by A since this would be possible only if the message was signed using the private key of A which is held only by A (unless compromised).
10. If the signatures are verified successfully, then I print the decrypted message i.e. the original message obtained in 4, else I print the appropriate error message.

Figure 3: Top half shows the output from q3.py and the bottom half shows the actual content in the file.



```
/mnt/i/Networks-Lab/lab11
→ python3 q3.py
42 is the answer to the ultimate question of life, the universe, and everything
/mnt/i/Networks-Lab/lab11
→
```

```
/mnt/i/Networks-Lab/lab11
→ echo   | cat message.txt -
42 is the answer to the ultimate question of life, the universe, and everything
/mnt/i/Networks-Lab/lab11
→
```