Audrey Long

JHU Web Security

Crypto/Digital Certificate Module Assignment

02/02/2021

Virtual Machine Setup

If you don't have it running already, download and install a VM software (e.g. Oracle's VirtualBox), then download the JHU_WebSec_Ubuntu18.ova (it is a fresh Ubuntu install in a VirtualBox VM) and importing it by opening VirtualBox

and Click 'Machine' -> 'Add' to import the VM you downloaded.

DELIVERABLE: Please provide a PDF or Word Doc with the following questions answered and annotated processes.

Question 1: Understanding Public/Private Keypairs

This week we talked a bit about public/private key pairs. The public key is available to everyone and is also called the "encryption key" because the public can use it to encrypt messages for the owner of the keypair. Conversely the private key, which should only ever be known to its owner and NOT the public, is known as the "decryption key".

For this question let's assume Alice wants to send a secure message m to Bob and Bob wants to be able to verify the message m was sent by Alice.

Alice has two keys: {k_Apub (public), k_Aprv (private)}

Bob has two keys: {k_Bpub (public), k_Bprv (private)}

For each of the following explain why/why not whether or not the encryption schemes accomplish: (I gave you one example)

a) Confidential (the message contents cannot be read by anyone other than the recipient)

b) Verifiable (the recipient of the message can verify the sender of the message)

1. Alice encrypts m using Bob's public key: Alice sends k_Bpub(m) = m' -> Bob K_Bpub(m) = m' -> Bob

Confidential: This encryption scheme does accomplish the goal of confidential, because once the message has been encrypted, it cannot be decrypted by anyone other than Bob, the intended recipient. This is because only Bob's private key can decrypt a message which has been encrypted with Bob's public key.

Verifiable: This encryption scheme does not accomplish the goal of verifiable, because the recipient of the message cannot verify the sender of the message. In this case the sender could be ANYONE that has established a connection, and there is no way to tell if Alice sent the key to Bob, or if Eve (MiTM) was the one establishing a connection instead.

2. Alice encrypts m using Alice's private key: Alice sends k_Aprv(m) = m' -> Bob K_Aprv(m) = m' -> Bob

Confidential: This encryption scheme does accomplish the goal of confidentiality because Bob has no awareness of what Alice's private key is, but Bob can decrypt the incoming message with Alice's public key.

Verifiable: This encryption scheme is verifiable because Bob can be fairly certain that the data he has received comes from Alice because only Alice has the private key.

3. Alice encrypts m using Bob's public key, then encrypts the results with her public key: Alice sends k_Apub(k_Bpub(m)) = m' -> Bob

Confidential: This solution is not considered to be confidential because only Bob can decrypt using his private key but would need Alice's private key to decrypt the outer message, which he does not possess because she encrypted with her public key.

Verifiable: All we know from this solution is that it is not verifiable because Alice encrypted the message with her public key and you would need Alice's private key to decrypt. So we know without certainty that Alice generated this message because the public keys are completely in the open.

4. Alice encrypts m using her public key, then encrypts the result with Bob's public key: Page 2 of 2 Alice sends k_Bpub(k_Apub(m)) = m' -> Bob

Confidential: This solution is not confidential because Bob can open the outer layer message with his private key, but cannot open the inner message because he does not contain Alice's private key.

Verifiable: This is not verifiable because Bob sees that the message was encrypted with Alice's public key which is known by everyone, Only Alice can decrypt this message.

5. Alice encrypts m using Bob's public key then encrypts the result with Alice's private key: Alice sends k_Aprv(k_Bpub(m))

Confidential: This encryption scheme does accomplish the goal of confidential, because Alice's public key can decrypt the outer layer and Bob's private key can decrypt the inner layer.

Verifiable: This encryption scheme is verifiable because Bob can be fairly certain that the data he has received comes from Alice because only Alice has the private key. But Alice's private key message can be decrypted with the public key that everyone has. Bob's public key message can only be decrypted with Bob's private key, so integrity still stands.

6. Alice encrypts m using her private key then encrypts the result with Bob's public key: Alice sends k_Bpub(k_Aprv(m)) = m' -> Bob

Confidential: This encryption scheme does accomplish the goal of confidential, because only Bob and Alice can decrypt this message completely. The first decryption can occur on Bob's end using Alice's public key, then the outer layer can be decrypted using Bob's private key.

Verifiable: This encryption scheme is verifiable because Bob can be fairly certain that the data he has received comes from Alice because only Alice has the private key. This scheme verifies that only Bob can receive this message, and only Alice can send it

Question 2: Generating Your Own Public/Private Keypairs
In your Ubuntu VM, if OpenSSL is not already installed, please install it. Next, complete the following:
Provide annotated screenshots of the major steps in the process:

1. Generate a 2048 bit RSA (AES-128) key pair. Name your key "JHED_ID.key".

```
root@student:/home/student# openssl genrsa -des3 -out JHU_ID.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................+++
...+++
e is 65537 (0x010001)
Enter pass phrase for JHU_ID.key:
Verifying - Enter pass phrase for JHU_ID.key:
```

2. What are the names of your public keyfile and private keyfile?

```
root@student:/home/student# ls
Desktop  Documents  Downloads  JHU_ID.key  Music  Pictures  Public  Templates  Videos
root@student:/home/student# openssl rsa -in JHU_ID.key -outform REM -pubout -out public.pem
rsa: Invalid format "REM" for -outform
rsa: Use -help for summary.
root@student:/home/student# openssl rsa -in JHU_ID.key -pubout -out public.pem
Enter pass phrase for JHU_ID.key:
writing RSA key
root@student:/home/student# ls
Desktop  Documents  Downloads  JHU_ID.key  Music  Pictures  Public  public.pem  Templates  Videos
root@student:/home/student#
```

Private key: JHU_ID.key
Public key: public.pem

3. Next create a Certificate Signing Request that is valid for 180 days using the private key you created above.

```
root@student:/home/student# openssl req -days 180 -out CSR.csr -key JHU_ID.key -new
Enter pass phrase for JHU_ID.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
```

4. Finally, use your private key and CSR to create and sign an SSL certificate that is good for 180 days.

```
root@student:/home/student#
root@student:/home/student# openssl x509 -req -days 180 -in CSR.csr -signkey JHU_ID.key -out cert.
crt
Signature ok
subject=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
Getting Private key
Enter pass phrase for JHU_ID.key:
root@student:/home/student# ls
cert.crt        CSR.csr  Documents  JHU_ID.key  Pictures  public.pem  Videos
clientcert.csr  Desktop  Downloads  Music       Public    Templates
```

5. Print your self-signed SSL certificate to the screen and provide a screenshot of the output.

```
root@student:/home/student# cat cert.crt
-----BEGIN CERTIFICATE-----
MIIDBjCCAe4CCQDuDdSats1/jzANBgkqhkiG9w0BAQsFADBFMQswCQYDVQQGEwJB
VTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UECgwYSW50ZXJuZXQgV2lkZ2l0
cyBQdHkgTHRkMB4XDTIxMDIwMTIzMjQyNloXDTIxMDczMTIzMjQyNlowRTELMAkG
A1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUxITAfBgNVBAoMGEludGVybmV0
IFdpZGdpdHMgUHR5IEx0ZDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
AMM2aiUT9C81IGcWcVOosWhhOG/2+WGZHHDGrhxeyp99HapMWwzwze6NHFxfHzl+
aFNWT3QMwum8QaBGQBKWL76/4B3cyNE7tysesYHKZqxYzKV5j8TRNcssX3oKc0c0
N/NscSq8dOrovMWJMEm22olccDcYWMOTROAivvb9jN6OcArhc9QF7iLpDsq7EICA
NZwJudGh4Cno/68CK3wsvYFnQiF7WuVLKqMLHEW6zBKGr/J+q+7PLI4XxjCqfZFB
26dmf2D6u8+6XwD/3VGDKqmz86elO9MMksr7hHGawvfjXkx/zPhVqXB65RBjW5Jg
vQDoTaefc5xF6ZsEN4/TqjkCAwEAATANBgkqhkiG9w0BAQsFAAOCAQEAod/9ycFg
cna6Jg7re68UbjFYU0FwJMWx7pKTvpgnRXTU/K9y7oW6OiM6F4dEgvajQbLDtqiq
4k8WYugaHB/D89RRxWBWqggHIlZNX6QOHi5wP+Wsdjva3J9XvM8URdduDpDF1ECZ
rdhSnk3UK+rGAVpQCt6gs8UEJmo/d2JX23CZ8ZRjGVLpUjfgL2XZ4Cr6Wc00lOz0
sGsRe+qjIcx4hsLXPWj0pICX1Mtoz53r1S1aOWZbQao1MkZSANj0nD58EATjoIJf
HgiSSigbgyDVt0/jBTFgrA8DYCiU+A917HHlj3/3R7Uezvk5kOlJ4nlVrDwEJKyu
knVm7L04CvBHvQ==
-----END CERTIFICATE-----
root@student:/home/student#
```

6. When interacting with untrusted websites on the internet, it is important to verify the authenticity of the server's certificate. However, we just generated our own keypair above. Name and describe a use case where self-signed certificate infrastructure would be useful.

A use case where self-signed certificate infrastructure could be useful would be an intranet where no outside internet connections can be made and there is virtually no chance that a man in the middle attack can occur to hijack the connection. Development servers can also be a nice place for a self-signed certificate infrastructure, as long as functionality exists and is tested with a self-sign infrastructure pre-production it can be replaced when going into production with a real secure certificate.

7. What are some of the advantages of using self-signed certificates rather than certificates issued by a third-party provider like VeriSign?

Some advantages of using self-signed certificates can be considered with the use case where both sides of the communication already know each other within the same system. In that use case the certificate can be used as credentials and contain a layer of security. Testing environments are also a great place to use self-signed certificates, because it keeps the development moving forward while placing all of the necessary checks for a quicker CI/CD environment.

Question 3: Main-in-the-Middle Attacks

SSL can be vulnerable to a "man-in-the-middle" attack.

1. Briefly describe what a MiTM attack is and how it works.

Generally speaking, a man in the middle attack works by an adversary listening in on a client to server transmission and secretly alters and relays information so the client and server think they are communicating with each other, but in fact the communication has been intercepted

2. What measures can be taken to detect/prevent a MiTM attack from occurring?

Man in the middle attacks can be prevented by a few means, one of which is authentication, which provides a layer of integrity that a message comes from an expected source. These attacks can be detected with some tamper detection mechanisms. Some smart checking by timing latency and hash calculations can tip off the user that something fishy is happening.

3. Can quantum computing do anything to detect these types of attacks? If so, briefly describe how.

In theory, quantum computing can detect man in the middle attacks with the help of the no-cloning theorem which states that it is impossible to create an independent and identical copy of an arbitrary unknown quantum state. This means that a man in the middle attack would be impossible because this theorem asserts that a quantum state cannot be copied, thus cannot be impersonated.

References

https://en.wikipedia.org/wiki/Man-in-the-middle_attack#Defense_and_detection [1]

https://www.techrepublic.com/article/when-are-self-signed-certificates-acceptable-for-businesses/ [2]

https://rietta.com/blog/openssl-generating-rsa-key-from-command/ [3]

https://www.sslshopper.com/article-most-common-openssl-commands.html [4]

https://support.f5.com/csp/article/K4146 [5]

https://www.sslshopper.com/article-how-to-create-a-self-signed-certificate.html [6]