University of Miami

Crypto Lab 3

ECE 576

Nigel John
10-16-2019

Objectives:

1. To understand and use symmetric and asymmetric key cryptography
2. To securely exchange a file over an unsecure channel

Task 0: Install python (v3), python IDE, python cryptography

For this lab, you are going to need to have python installed on your laptop. If you already have this then you can skip this step.

1. Download the latest python (v3) from www.python.org for your operating system
    a. Your SEEDs systems have python 2 but not python 3
    b. Your Kali has both, but you may need to update
2. Follow the installation instructions for your system
    a. See: https://wiki.python.org/moin/BeginnersGuide
3. For programming information:
    a. https://wiki.python.org/moin/HowToEditPythonCode
4. Install the python cryptography module
    a. Open a Terminal window (or Command shell)
    b. Type: pip install cryptography
        i. check https://cryptography.io for installation issues
5. Download and install an IDE:
    a. IDLE: python ships with a standard IDE, IDLE, written in pure python
    b. PyCharm: www.jetbrains.com/pycharm
    c. Eclipse with PyDev: www.eclipse.org (my personal preference)
    d. Xcode on the Mac also supports python
6. Familiarize yourself with the platform
7. For this lab you also need openSSL for the backend to get access to the PK algorithms
8. The python documentation is at:
    a. Main documents: https://docs.python.org/3/
    b. Tutorial: https://docs.python.org/3/tutorial/index.html

Task 1: Certificates

So far the RSA keys that you have used were only useable on your system. This is fine for the private key, but you need to be able to move the public key to another system and still maintain authentication of the key. We will initially do this with a self-signed certificate and then with a CA-signed certificate.

Note that you will have to maintain a directory structure with files that contains (this will be a keystore):

1. Your encrypted private key
2. Your public X509 certificate
3. Public X509 certificates for everyone with whom you want to exchange information

Task 1.1: Create a self-signed certificate for User 1

1. Generate and save an RSA private/public key pair as in CryptoLab 2: Task 2
    a. You can use the ones you did for CryptoLab2 or create new ones
2. Create a python file
    a. Remember that you can import your previous files and use the functions there to simplify these steps
3. Import
    a. from cryptography import x509
    b. from cryptography.x509.oid import NameOID
    c. from cryptography.hazmat.primitives import hashes, serialization
    d. from cryptography.hazmat.backends import default_backend
4. Load your private key and public_key from step 1
    a. Or access it if you still have it loaded
5. Create the subject and issuer of the certificate as the same person
    a. subject = issuer = x509.Name([
    b. x509.NameAttribute(NameOID.COUNTRY_NAME, u"US"),
    c. x509.NameAttribute(NameOID.STATE_OR_PROVINCE_NAME, u"Florida"),
    d. x509.NameAttribute(NameOID.LOCALITY_NAME, u"Coral Gables"),
    e. x509.NameAttribute(NameOID.ORGANIZATION_NAME, u"University of Miami"),
    f. x509.NameAttribute(NameOID.ORGANIZATIONAL_UNIT_NAME, u"ECE Depti"),
    g. x509.NameAttribute(NameOID.COMMON_NAME, u"User 1"),])
6. Create a Certificate builder object
    a. builder = x509.CertificateBuilder()
7. Set the subject and issuer
    a. builder = builder.subject_name(subject)
    b. builder = builder.issuer_name(issuer)
8. Set the date
    a. builder = builder.not_valid_before(datetime.datetime.today() - one_day)
    b. builder = builder.not_valid_after(datetime.datetime(2018, 8, 2))
9. Set a random serial number
    a. builder = builder.serial_number(x509.random_serial_number())
10. Add the public key
    a. builder = builder.public_key(public_key)
11. Add the basic extensions
    a. builder = builder.add_extension(
    b. x509.BasicConstraints(ca=False, path_length=None), critical=True,)
12. Sign the certificate

a. certificate = builder.sign(
b. private_key=private_key, algorithm=hashes.SHA256(),
c. backend=default_backend())

13. Save the certificate
    a. cert_name = 'user1_cert.pem'
    b. with open(cert_name, 'wb') as file:
    c. file.write(certificate.public_bytes(serialization.Encoding.PEM))
14. You should now have a self-signed certificate for User 1

Task 1.2: Create a self-signed certificate for User 2

Repeat Task 1.1 for User 2 making sure that:

1. You create a new set of keys and store them in different files from User 1
2. You change the COMMON_NAME to User 2
3. Save the certificate to a file user2_cert.pem

Task 2: Signing and Verifying

Now let us use these new keys and certificates to sign and verify a file

Task 2.1: Signing a file

1. Use the signing code from CryptoLab2 Task 3.1 to create a signature for a data file
    a. Use the private key for User 1 as the private key for signing
2. To simplify operations later, change the padding on the key from PSS to PKCS1v15
    a. pad = padding.PKCS1v15()
3. Sign the file using the code from CryptoLab2 Task 3.1
    a. Load and hash the data from the file
    b. Load the private key
    c. Set the padding
    d. Sign the data
4. Save the signature

Task 2.2: Verifying a signature

1. Use the verification code from CryptoLab2 Task 3.2 to verify the signature
    a. Do not load the public key from a file, but from the certificate
2. Load the certificate for User 1
    a. with open('user1_cert.pem', 'rb') as file:
    b. certificate = x509.load_pem_x509_certificate(
    c. data=file.read(),
    d. backend=backend)
3. Get the public key from the certificate
    a. public_key = certificate.public_key()
4. Use the PKCS1v15 padding
    a. pad = padding.PKCS1v15()
5. Verify the signature for the file as in CryptoLab2 Task 3.2
    a. Load and hash the data from the file
    b. Load the signature
    c. Load the certificate
    d. Access the public key
    e. Set the padding
    f. Verify the signature using the public key

Task 2.3: Verifying a certificate

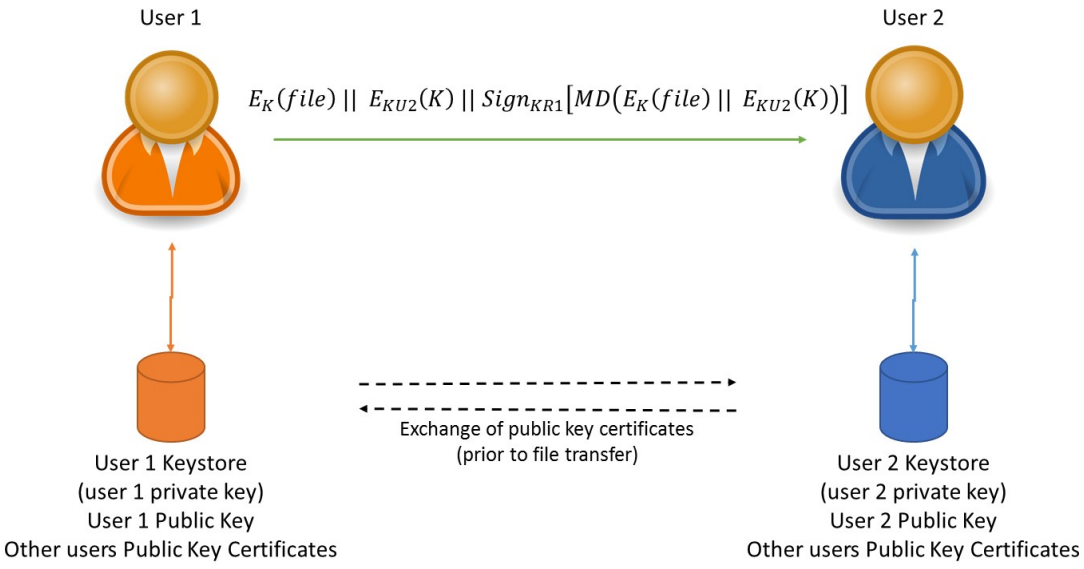
It is also possible to verify the signature on a certificate. In a self-signed certificate, the public key in the certificate in used to verify the signature. In a CA-signed certificate, the public key in the CA certificate is used to verify the signature of a certificate signed by the CA.

1. Load the certificate
    a. with open('user1_cert.pem', 'rb') as file:
    b. certificate = x509.load_pem_x509_certificate(
    c. data=file.read(),
    d. backend=backend)
2. Get the public key from the certificate
    a. public_key = certificate.public_key()
3. Get the signature
    a. sig = certificate.signature
4. Get the data to be used to check the signature
    a. data = certificate.tbs_certificate_bytes
5. Hash the data
6. Set the padding
7. Verify the signature

Task 3: Incorporate the file encryption, message digest, signature and public key certificate to create a complete file exchange

This task requires you to:

1. Create two keystores, k1 and k2 with different keys to represent two users.
2. Copy the certificate from k1 and to k2 and vice versa.
    a. So k1 has
        i. User 1 encrypted private key file
        ii. User 1 certificate
        iii. User 2 certificate
    b. And k2 has
        i. User 2 encrypted private key file
        ii. User 2 certificate
        iii. User 1 certificate
3. Now as user 1:
    a. Create an encrypted file with a randomly generated secret key
        i. Symmetric encryption, AES-128
    b. Encrypt the secret key used for the file encryption with the public key of user 2
        i. From User 2 certificate in keystore k1
    c. Create a message digest of the encrypted file and the encrypted key
    d. Sign this message digest with user 1's private key
        i. From User 1 private key file in keystore k1
    e. Save each part as a separate file
4. Copy all files to user 2 and as user 2:
    a. Verify the signature using user 1's public key
        i. From User 1 certificate in keystore k2
    b. Decrypt the secret key using user 2's private key
        i. From User 2 private key file in keystore k2
    c. Decrypt the encrypted file using the secret key
5. Congratulations, you have now implemented a basic secure file exchange mechanism



Report

1. Statement of what was done for each task
2. All source code for each task
3. Screen shots of all operations for each task

Reference:

Python reference: https://docs.python.org/3/
Cryptography module reference: https://cryptography.io/en/latest/