Crypto Lab 8b – Digital Signatures

# Manual “Schoolbook” approach

This lab will create a signature using a simple method to illustrate the concept. It has several vulnerabilities and should not be used where security of the signature matters!

Graphical user interface

Description automatically generated with low confidence

## Key Generation

In Crypto Lab 4, where we computed RSA encryption manually, we used a very small value for n. We chose p = 131 and q = 157, which gave n = 20567. That n is only 15 bits long. We are going to encrypt a SHA-256 hash which is 256 bits long. If we want to fit our encrypted hash into one block, the size of n must be larger than 256 bits.

PyCryptodome has a function, getPrime(x), where x is the size in bits of the prime number we want to generate. If we use getPrime(150) for both p and q, n = p\*q will be about 300 bits long.

Text

Description automatically generated

Once we have p, q, and n, we can select our encryption exponent e and compute our decryption exponent d just as we did in lab 4.

Text

Description automatically generated

Note: instead of phi = (p – 1) \* (q -1), we could have used  
phi = (p – 1) \* (q -1) // GCD(p – 1, q – 1)  
Both work.

Record the values of the private [e, n] and public [d, n] keys so you can use them in the rest of the lab.

## Create a signature.

### Read a file.

Instead of signing a short text message, we will sign a file. Pick some random file from your Python directory and read it into a variable. It is important to read the file as binary since the hash function wants bytes as input.

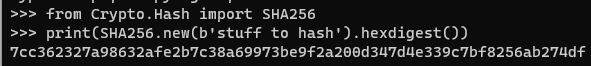


### Hash the file data you read.

The documentation for PyCryptodome’s hash functions is here: [https://pycryptodome.readthedocs.io/en/latest/src/hash/hash.html#](https://pycryptodome.readthedocs.io/en/latest/src/hash/hash.html) There is also a hash library that is part of the Python distribution. <https://docs.python.org/3/library/hashlib.html> Since we have been using PyCryptodome for the other functions, we will continue to use it for hashes. After importing the function,  
from Crypto.Hash import SHA256  
the function has three steps.

1. Create a hash object. You can give the new() function your data you want to hash. If the data is stored in a variable use  
   hash\_obj = SHA256.new(variable)  
   If you just want to hash characters use  
   hash\_obj = SHA256.new(b’this is what I want to hash’)  
   Note the b at the beginning of the data. SHA256 wants bytes, not strings.
2. Optionally, add more data to be hashed. If you had a huge file to hash and didn’t want to read it all into memory, you could add to the hash line by line. We won’t use this.  
   hash\_obj.update(b’add another line of stuff to hash’)
3. Output the hash. You have the choice of outputting in bytes or in a hex string.  
   hash\_obj.digest() outputs bytes  
   hash\_obj.hexdigest() outputs a hex string

Note: You can combine all the steps into one line (not counting the import statement) if you like. This is a handy way to compute hashes from the interactive prompt.



Graphical user interface, text, application

Description automatically generated

### Problem: we must change myHash from a hex string to and integer

The output of the hashdigest() function is a hex number in string format like this:  
'8204a4791f9ecdcb0ce9cea000b32fb10adf64771fa28a045238ded64a382a67'

The pow() function wants the input as an integer, so we must convert it. One way to do this is:  
>>> myHash

'8204a4791f9ecdcb0ce9cea000b32fb10adf64771fa28a045238ded64a382a67'

>>> int(myHash, 16)

58808872855968173572145847689955514554808283390921265099292751484176242846311

Hex is base 16, so that is where the 16 came from in the int() function.

### Compute the signature.

Compute the signature by encrypting the hash with the private key. Since we have the private key, and no one else has it, the signature had to have been created by us. Everyone has access to our public key, so anyone can verify that the signature is correct, and that we created it.

Encryption is simple: take the hash to the power of the exponent, d, from our private key, modulus n.

Text

Description automatically generated

## Send the file and hash to the recipient.

This can be done by mailing the file and the hash to the recipient, posting them to a website, or any number of ways. We also assume that the recipient has our public key, e and n from the key generation script.

## Recipient verifies the signature.

To verify the signature, the recipient:

1. Hashes the file they received.
2. Decrypts the signature with the public key.
3. If the signature is valid, the decrypted signature and the hash will be the same.

The first few lines of the verification script are the same as the signature script: read the file and create a SHA256 hash.

The decryption part of the script is like the signature script, except:

1. Use the public key, n and e.
2. Decrypt the signature.

If the decrypted signature and the hash we made are the same, the signature is verified. Otherwise, it fails.

Text

Description automatically generated

# Secure method

This method is taken from examples in the PyCryptodome documentation <https://pycryptodome.readthedocs.io/en/latest/src/signature/pkcs1_pss.html>. It uses the method shown in slide 7 of the class notes. If you have time, use this example to create a secure signature.

You will have use your code from Lab 5a to generate keys and save them to files (private.der and pem.der) so that the code in the example can use them.

Graphical user interface, text, application, email

Description automatically generated

# Hand in

Show screenshots of the following process:

1. Generate your own private and public keys.
2. In a separate script, use your private key to sign a random file of your choosing.
3. In another script, verify the signature. It should pass.
4. Change one character in the file you signed, and then verify the signature again. It should fail because the file has been altered since you created the signature.

If you have time, try the secure method. If not, read the code in the secure method so that you can answer this question. “Of the two methods, Schoolbook and Secure, which on teaches you more about how signatures work?”