# PROJECT 5A
# TESTING OF RSA 2048 ENCRYPTION/DECRYPTION IN MICRO-PYTHON

## PROJECT DOCUMENTATION

## <u>Main Code Explanation.</u>

- **Main.py**
  - **read_settings():** reads data from the settings.json file, and returns ssid, password, atSign, and privateKey.
  - **read_key(atSign):** reads data from the keys file, and returns aesEncryptPrivateKey, aesEncryptPublicKey, aesPkamPrivateKey, aesPkamPublicKey, and selfEncryptionKey.
  - **aes_decrypt(aesEncryptedData, aesKey):** decrypts aesEncryptedData using aesKey and returns the decrypted data.
  - **sync_time():** syncs the device's time with an NTP server.
  - **find_secondary(atSign):** returns the IP address of the secondary corresponding to atSign.
  - **connect_to_secondary(secondary):** connects to the secondary at the specified secondary IP address and returns a socket object ss.
  - **send_verb(ss, verb):** sends verb over the ss socket and returns the response and the next command to be sent.
  - **send_verbs(ss, verb):** sends verb over the ss socket and returns the response and the next command to be sent.
  - **b42_urlsafe_encode(data):** encodes data using base 64 and returns the encoded data in a URL-safe format.
  - **get_pem_parameters(pem_key):** extracts the private key parameters from the given pem_key and returns a list containing the parameters.
  - **get_pem_key(pkamPrivateKey):** returns the PEM-formatted key corresponding to the given pkamPrivateKey.
  - **main():** the main function that performs the following operations:
    - reads the ssid, password, atSign, and privateKey from settings.json
    - reads the aesEncryptPrivateKey, aesEncryptPublicKey, aesPkamPrivateKey, aesPkamPublicKey, and selfEncryptionKey from the keys file
    - decrypts the aesPkamPrivateKey using the selfEncryptionKey to obtain the pkamPrivateKey
  - Connects to the Wi-Fi network specified by ssid and password
  - syncs the device's time with an NTP server
    - displays a menu and performs the corresponding action based on the user's input:
    - if opt is 1 or 2, connects to a secondary and waits for user input to send over the socket
    - if opt is 3, generates a new private key and saves it to settings.json
    - if opt is 4, displays a temperature sensor menu and performs the corresponding action based on the user's input
    - if opt is 5, runs a test
    - if opt is 6, exits the program.

- **Pkcs1.py**
  - **_pad_for_encryption(message, target_length):** This function pads the given message for encryption, returning the padded message. The function receives two parameters: message, the bytes to be padded, and target_length, the length of the output, and returns a byte string in the following format: 00 02 RANDOM_DATA 00 MESSAGE.
  - **_pad_for_signing(message, target_length):** This function pads the given message for signing, returning the padded message. The padding consists of repetition of FF bytes. The function receives two parameters: message, the bytes to be padded, and target_length, the length of the output, and returns a byte string in the following format: 00 01 PADDING 00 MESSAGE.

- **HASH_ASN1:** A dictionary that maps the names of hash functions to their corresponding ASN.1 codes.
- **HASH_METHODS:** A dictionary that maps the names of hash functions to their corresponding hashing methods.
- **CryptoError:** A base class for all exceptions in this module.
- **DecryptionError(CryptoError):** Raised when decryption fails.
- **VerificationError(CryptoError):** Raised when verification fails.

**Test cases Explanation.**

- **Micropython_Test.py**
  - **read_settings():** This function reads settings from a configuration file and returns them as a tuple. The test_read_settings() function tests this function by calling it and asserting that it returns a tuple with a length of 4.
  - **read_key(at_sign):** This function reads a set of keys from a file using an @sign as a parameter and returns them as a tuple. The test_read_key() function tests this function by calling it with an @sign as a parameter and asserting that it returns a tuple with a length of 5.
  - **find_secondary(at_sign):** This function searches for a secondary server using an @sign as a parameter and returns the IP address of the secondary server as a string. The test_find_secondary() function tests this function by calling it with an @sign as a parameter and asserting that it returns a string.
  - **connect_to_secondary(ip_address):** This function connects to a secondary server using an IP address as a parameter and returns a boolean indicating whether the connection was successful or not. The test_connect_to_secondary() function tests this function by calling it with the IP address of a secondary server and asserting that it returns True.
  - **send_verbs(ss, at_sign):** This function sends a command to a secondary server and returns the response and the command as strings. The test_send_verbs() function tests this function by calling it with a secondary server connection and an @sign as parameters and asserting that it returns two strings.
- **testcase.py**
  - **test_encrypt_decrypt():** This function tests the encryption and decryption functionality of the RSA algorithm using the encrypt() and decrypt() functions from the rsa.pkcs1 module. It generates a new key pair, encrypts a plaintext message using the public key, decrypts the resulting ciphertext using the private key, and checks that the decrypted message matches the original plaintext. The function takes no parameters and returns nothing.
  - **test_sign_verify():** This function tests the signing and verification functionality of the RSA algorithm using the sign() and verify() functions from the rsa.pkcs1 module. It generates a new key pair, signs a message using the private key and a specified hash function, and verifies the resulting signature using the public key. The function takes no parameters and returns nothing.
  - **test_sign_hash():** This function tests the signing and verification of a hash value using the sign_hash() and verify() functions from the rsa.pkcs1 module. It generates a new key pair, computes the hash value of a message using a specified hash function, signs the hash value using the private key, and verifies the resulting signature using the public key. The function takes no parameters and returns nothing.
  - **test_decrypt_fail():** This function tests the behavior of the decrypt() function when it is passed an invalid ciphertext. It generates a new key pair, encrypts a plaintext message using the public key, modifies the resulting ciphertext to introduce an error, and attempts to decrypt the modified ciphertext using the private key. The function expects the decryption to fail and raises a DecryptionError exception if it does not. The function takes no parameters and returns nothing.

## Library Explanation

- **unittest.py** (optimized library to deal with memory issues of Pico-W)
  - It is a library that contains several classes for testing purposes.
  - **SkipTest:** This is an exception class that can be raised to indicate that a test should be skipped.
  - **AssertRaisesContext:** This is a context manager class that can be used to test whether a particular exception is raised by a function.
  - **TestCase:** This is a base class that provides various methods for testing, such as assertEqual, assertTrue, and assertRaises. Test cases are created by subclassing this class and defining methods that start with the prefix test_.
  - **skip:** This is a decorator that can be used to skip a test if a condition is true.
  - **skipIf:** This is a decorator that can be used to skip a test if a condition is true.
  - **skipUnless:** This is a decorator that can be used to skip a test if a condition is false.
  - **TestSuite:** This is a container class for test cases.
  - **TestRunner:** This is a class that runs a suite of test cases and reports the results.
  - **TestResult:** This is a class that collects the results of running a suite of test cases. It keeps track of the number of tests run, the number of failures, the number of errors, and the number of tests that were skipped.

- **base64.py**
  - Micro-python library for b64decode(s, altchars=None, validate=False)
  - This library decodes a Base64 encoded string or bytes object into its original binary form.