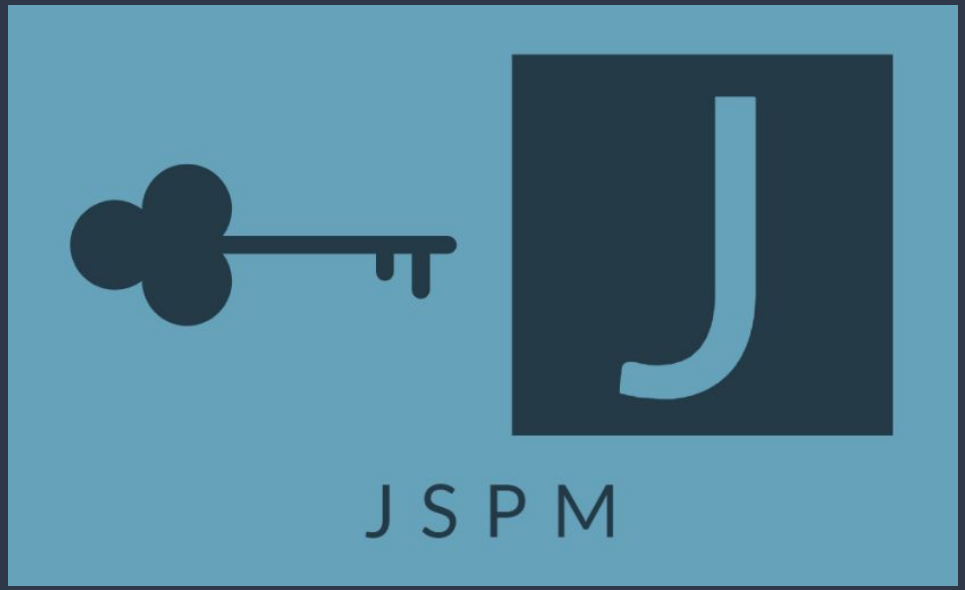

MSCS 710L Final Project

Jarett Sutula

Background

AES (Advanced Encryption Standard)

AES uses the same key for encryption and decryption.

AES requires the key and the plaintext to be encrypted to be exactly 16 bytes (in our case since we use AES-128).

JSPM was built in python using the foundation of the lab code we built in Java. The web app used is django and the database in which the encrypted data is stored is MongoDB.

Methodology

A JSPM account has a unique username and a password upon creation.

This password will serve as the key for later encryption on a user's credentials. It is hashed and salted using the python module bcrypt.

Website credentials will not always be 16 characters, so PKCS5 padding is used to make the plaintext 16 bytes.

Ex: plaintext = "my_password" of length 10.
m y _ p a s s w o r d
6D 79 5F 70 61 73 73 77 6F 72 64

PKCS5 padding takes the left over amount of bytes (16 - 10) = 6 and sets the padding hex digits to that (in this case, 0x06).
m y _ p a s s w o r d - - - - - -
6D 79 5F 70 61 73 73 77 6F 72 64 06 06 06 06 06 06

When changing the hex string back to plaintext during decryption, we can just read the last value (0x06) and remove the last 6 hex pairs on the string to give us the correct un-padded ciphertext.
With this in place, JSPM can encrypt and decrypt credentials of any length.

JSPM Demo

Results and Conclusion

A lot of famous credential leaks (Yahoo 2014, Facebook 2012) come from **passwords being saved in plaintext**.

Applying AES encryption to passwords before being pushed to databases would mitigate the dangers of database leaks significantly.

Additional hashing and salting of secret keys (in our case the **JSPM password**) **makes guessing the secret key a brute force attack on AES** which is well beyond our computation powers and time requirements.

With the right mindset, **AES encryption can still be relevant for credentials** (and some password managers use it too!)