<u>CSI4139</u>

Lab 2

Name: NamChi Nguyen
Student #: 7236760

Group members:
Julian Templeton - 8229400
Sean McAvoy - 8263210

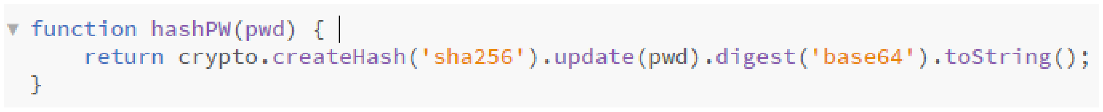
Two-Factor Authentication

For this lab, we used the given pre-existing solution 2FALab that utilized the Authy API through Twilio's services. Thus, the purpose of this report is to analyze the Authy implementation.

The programming language used was JavaScript, along with Node.js as the back-end to use the API Authy by Twilio and the database MongoDB. MongoDB is the database used to store the user's credentials and phone number in order to verify a returning user whereas Mongoose is the object modeling tool for MongoDB [1].

<u>Analysis of Authy implementation:</u>

SHA-256: Using the Hash class, it creates data hash digests in which the computed hash is then generated by the hash.update() and hash.digest() methods. This hashing function used SHA-256 to hash passwords, providing a security level of 256-bits. The method hash.update() takes the password to be hashed and the hash.digest() calculates the digest of the password as a string of base 64 [2].

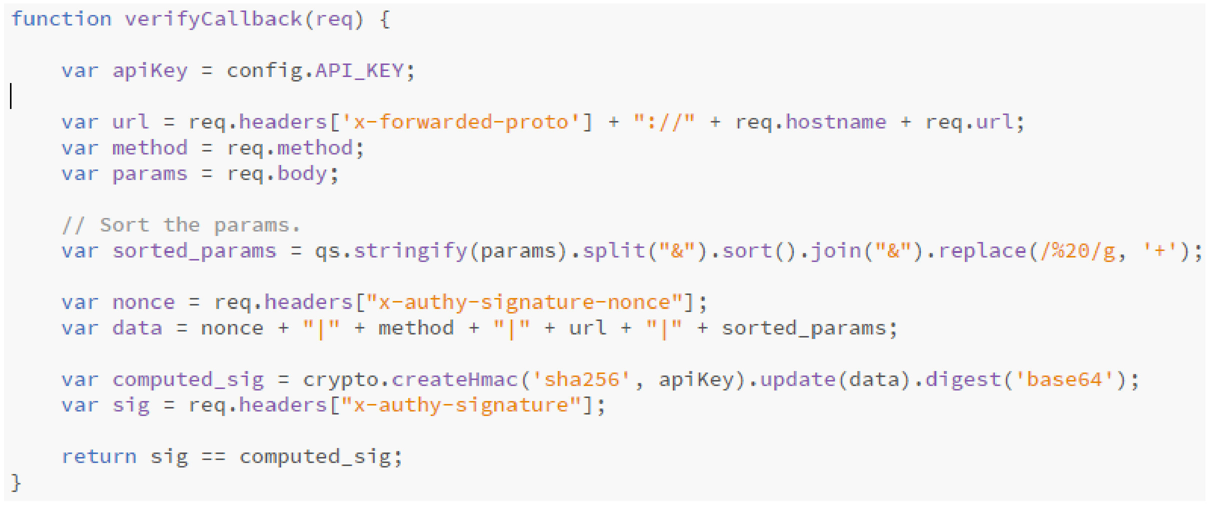


```
function hashPW(pwd) {
    return crypto.createHash('sha256').update(pwd).digest('base64').toString();
}
```

However, despite the use of SHA-256, there was no password salting to accompany the hashing. Even though SHA-256 is a secure hashing algorithm, it also makes fast computations. This can result in fast brute-force attacks through lookup tables by the attackers who could be using high-end graphic cards (GPUs) that can run over a billion hashes per second [3], [4].

So, in terms of password hashing, using cryptographic hash functions from the SHA-3 families are not ideal. A solution to diminish the probability of a successful attack is to make the hash function extremely slow which is also known as key stretching, and contrary to the computationally fast SHA-3 algorithms. Some standard key stretching algorithms are PBKDF2, bcrypt and scrypt that combine hashing and salting together. These algorithms specify the number of iterations as a parameter in which this value determines the computational speed of the hash function. Hence, a large iteration count would yield a slower hash function. In a web application, the iteration count can be lower to avoid a DoS [4].

HMAC: To verify the authenticity of the message and to confirm it was sent by Twilio, a HMAC digest was created using the hash function SHA-256 and the application API key [1].



```
function verifyCallback(req) {

    var apiKey = config.API_KEY;

    var url = req.headers['x-forwarded-proto'] + "://" + req.hostname + req.url;
    var method = req.method;
    var params = req.body;

    // Sort the params.
    var sorted_params = qs.stringify(params).split("&").sort().join("&").replace(/%20/g, '+');

    var nonce = req.headers["x-authy-signature-nonce"];
    var data = nonce + "|" + method + "|" + url + "|" + sorted_params;

    var computed_sig = crypto.createHmac('sha256', apiKey).update(data).digest('base64');
    var sig = req.headers["x-authy-signature"];

    return sig == computed_sig;
}
```

Encryption/digital signature: The usage of encryption and a digital signature weren't included in the Authy implementation although it would've been used following the protocol described in the lab manual if we had chosen to alternatively implement from scratch.

Public/private keys: As for the creation of the public and private key pairs, it is assumed that the private key is sent to the user's device whereas the public key of the Authy application is the Production API Key that can be found in an Authy application in the Twilio console.

PRODUCTION API KEY ••••••••••••••••••••••••••••

<u>2FA Flow:</u>

When a user first registers on the website, they provide a username, email, password and phone number. Before verifying the user's phone number, an authyID must be created by the Authy API client as stated in the User Schema (found in user_model.js and users.js) which is then stored in the database. Next, the user would receive a one-time password from Authy to verify the phone number and then after entering the one-time password, the registration is completed [2].

When a returning user logins, a SMS message containing a token is sent to the user's phone and then the user inputs the token to login. It is assumed that the method of the text message being sent to the user is secure.

Overall, to conclude, two factor authentication is crucial nowadays to augment the security of one's account and to protect personal data. The additional steps to validate that the correct person is logging in as well as confirming that the connection to the server is also secure will help to reassure the user that their accounts haven't been compromised and to make it more difficult for an attacker to gain access.

References:

[1] Twilio, “Account Verification with Authy, Node.js and Express”, 2018, [Online]. Available: https://www.twilio.com/docs/authy/tutorials/account-verification-node-express. [Accessed Oct. 5, 2018]

[2] Node.js, “Node.js v10.11.0 Documentation”, n.d., [Online]. Available: https://nodejs.org/api/crypto.html#crypto_hash_digest_encoding. [Accessed Oct. 5, 2018]

[3] D. Arias, “Hashing In Action Understanding Bcrypt”, May 31, 2018, [Online]. Available: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/. [Accessed Oct. 6, 2018]

[4] Defuse Security, “Salted Password Hashing - Doing it Right”, July 30, 2018, [Online]. Available: https://crackstation.net/hashing-security.htm. [Accessed Oct. 6, 2018]