INTRODUCTION

So what we are doing in project we would like to encrypt the data before they are uploaded to the server so we're not going to upload via a secure channel & then encrypt the data on the server or leave it unencrypted there **we want to encrypt the stuff before it leaves the browser** so that's the basic requirement & to use cryptographic functions on the browser we're gone use the **Web Crypto API interface(JAVASCRIPT API) allowing a script to use cryptographic operations such as hashing, signature verification, encryption & decryption in browser directly this is purely a browser based API only.**

Before starting discussion on web crypto API let's clear some the terminologies related to the cryptography.

TERMINOLOGIES RELATED TO CRYPTOGRAPHY

Private Key Encryption (Symmetric Key Encryption) :

- In private key encryption, data is encrypted & decrypted using a single key that only the sender and the receiver knows. That is why private key encryption is also called **symmetric key encryption** because the same key is used during both encryption and decryption of the transmitted data.The private key encryption is a simple method because both parties use the same key to decrypt the secret.

  METHOD -:

  1. **STREAM CIPHER** : stream cipher works on a single bit at a time it encrypts the data bit by bit this method is not used in modern cryptography.
  2. **BLOCK CIPHER** : block cipher is an algorithm operating on a fixed length group of bits called blocks to encrypt the data & the usual size of each block is 64 bits,128 bits ,256 bits.
     Eg : DES, Triple DES, AES, Blowfish etc.

Public Key Encryption (Asymmetric Key Encryption)

- In **public key encryption**, two different keys are used to encrypt and decrypt data.One is the public key and other is the private key. These two keys are mathematically related. They come as a pair.
- The public key encryption is also called **asymmetric key encryption** because two different keys are used.

How does the hash function work?

- Hash function used to create the unique digital fingerprint of the data called **digest.**

- Hash algorithms are primarily used for **comparison** purposes and **not for encryption** since the **hash functions are irreversible.**
- Characteristics of hash algorithms : **1) secured and non-reversible** i.e, the digest can not be reversed to determine the original data. **2) fixed size** whether the data is short or long it will produce the digest with the same length **3) unique digest** two separate data sets cannot produce the same digest value.
- hashing method commonly used for password hashing & you know the reason why.
- Go through dictionary attack and brute force attack.

Hashing Functions

- Cryptographic hash functions
    1. MD5
    2. SHA family
- Password hashing & key stretching functions
    1. **PBKDF2** (we will be using these function)
    2. Bcrypt
    3. Scrypt

How to salt your passwords? How to add "pepper" to salted passwords?

- Hackers can still hack the passwords using dictionary attacks or brute force attacks or rainbow table attacks even if they are hashed to make the task of the hacker difficult or the actions will become costlier. We use salt and pepper.
- A salt is a random value generated for each password, in other words each user has his own salt for his password.
- **SALTING** :

```
let digest value = "ed56g", password = "MyPassword" ;
hashing_function("MyPassword", "ed56g") = Hash_value or Digest
```

- remember the digest value needs to be kept along with each password.
- Pepper is not stored in a database & it is equal for all the passwords.
- Go through **confidentiality, integrity & authenticity.**

Encryption Algorithms

- Two of the most widely used encryption algorithms today are **AES** and **RSA**. Both are highly effective and secure, but they are typically used in different ways. Let's study both of them.

AES (Advanced Encryption Standard)

- The more popular and widely adopted **symmetric encryption algorithm.**
- **AES-256 GCM**
    - **AES -** kind of standard for symmetric key encryption.
    - **256 -** size of a key to encrypt and decrypt the data with.

    - **GCM -** is a specific mode of AES to provide the authenticated decryption or encryption means if we try to decrypt stuff that's encrypted with the key & that key is not correct in that case the operation will fail.
- The AES algorithm successively applies a series of mathematical transformations to each 128-bit block of data since it is based on **block cipher**.
- AES is a symmetric algorithm which uses the same 128, 192, or 256 bit key for both encryption and decryption (the security of an AES system increases exponentially with key length).

RSA Encryption

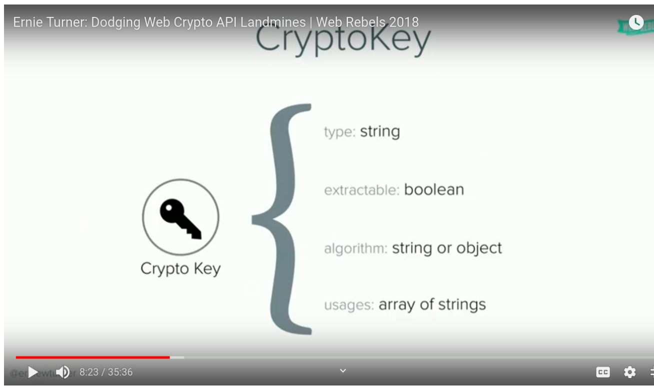
- It is an **asymmetric algorithm** that uses a publicly known key for encryption, but requires a different key, known only to the recipient, for decryption.
- RSA Encryption works on two different keys i.e. **Public Key** and **Private Key**.

Key derivation functions

- KDF's or Key Derivation Functions are functions to derive key from other secret information & used to generate a key with enough randomness in it.
- We will be using PBKDF2(Password base key derivation function 2.) with 250000 iterations and this operation which is a way of strengthening that password by repeatedly hashing it over and over again until it takes a long time in this way the key is hard to brute force by attackers.
- Eg. pbkdf2

INTRODUCTION TO WEB CRYPTO API

1. **CRYPTOKEY :** it's actually an opaque representation of cryptographic keys so when we have a key and we're using it for encryption and decryption operations we have to use the cryptokey. The cryptokey actually hides the actual bytes of that key so that it can't be seen by the user or developer, we can only view what this crypto key is used for & what kind of operations can i do with it.

   
   Ernie Turner: Dodging Web Crypto API Landmines | Web Rebels 2018
   CryptoKey
   Crypto Key
   type: string
   extractable: boolean
   algorithm: string or object
   usages: array of strings
   8:23 / 35:36

   Type : what kind of key is this whether it is a symmetric or asymmetric key.
   Extractable : indicating whether or not the key may be extracted.
   Algorithm : An object describing the algorithm for which this key can be used.
   Usage : indicating what can be done with the key.

   OPERATIONS :
   Operations that we have here :
   1. encrypt
   2. decrypt

3. importKey
4. exportKey
5. wrapKey
6. unwrapKey

PROJECT DISCUSSION :

INTRODUCTION :

Now let's dive into how our password manager application works. Let's start with an introduction.

Vault.IN is A password manager, digital vault with group management based on WebCryptoAPI http://www.w3.org/TR/WebCryptoAPI/. Vault.IN remembers all your passwords for you to help keep account information safe.it allows you to share individual records with any other users.

Vault.IN is based on **Zero-Knowledge Architecture.**

1. Data is encrypted & decrypted locally on the user's device(not on the server.)
2. The server never stores your record in plaintext.
3. The keys to encrypt & decrypt the record are derived on the user's device from the user's Master password.
4. Sharing of data uses **Public Key Cryptography**.

Before moving forward let's discuss some of the terminologies

Zero-Knowledge

It means that whenever we send our data to the server, the server should not have any knowledge about my data(i.e., what exactly zero-knowledge is). This is simple.

But, How to achieve that?

To achieve that we have our second Terminology client-Side Encryption.

Let's talk about it then..

Client-Side Encryption

Client side Encryption is the act of encrypting the data before sending it to the server.

Now, How to implement that??

To enable Client-side Encryption we have our **Web-Crypto API** made available to all the browsers as we discussed above.

NOTE* : *Let's start discussing the project. We will discuss the algorithm's use and terminologies as we move forward.

How it works

details of how Vault.IN works. Initially, you need to register with a username and password and an RSA-OAEP key pair is generated. (for public-key cryptography).then your private key is wrapped using AES-GCM-256 key instead of generating a random AES-GCM key we're gonna let you provide the password which we're then going to strengthen and harden and convert it into a CryptoKey.

Please understand RSA Key pairs i.e, the public & private keys are only generated at the time when users create an account or register.

Now the question arises how to manage/store these keys? On the server obviously because the key pairs are generated once for the user at time of registration.since we are storing most confidential information on the server so the **private key must be in an encrypted form.**

How do we encrypt or wrapp the private key? Answer is using **AES-GCM-256** bit Key.

Now let's walk through the flow of **how the AES-GCM key is generated?**. First of all, we're gonna get your password as bytes now we need to **import** these into CryptoKey (according to https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto/importKey.) we need to do this in order to derive an AES-GCM key and then your AES-GCM key is derived from your **PasswordKey**.

& this is how the PasswordKey is generated from users password.

```
const PasswordKey = await crypto.subtle.importKey(
                    "Raw",
                    passwordInBytes,
                    "PBKDF2",
                    false,
                    ["deriveBits", "deriveKey"]
            );
```

`deriveKey` Function takes salt and PasswordKey and then we're gonna run it through PBKDF2(Password base key derivation function 2.) with 250000 iterations and this operation which is a way of strengthening & hardening that password by repeatedly hashing it over and over again until it takes a long time meaning this way the key is hard to brute force so for this hashing we're gonna use **SHA-256** now we have an AES-GCM-256 key so we actually have something that we can actually wrap and unwrapped your **private key** with.

& this is how the AES-KEY is generated from passwordKey.

```
let aesKey = await crypto.subtle.deriveKey({
                  name:"PBKDF2",
                  salt,
                  iterations: 250000,
                  hash: {name: 'SHA-256'}
}, passwordCryptoKey, aesAlg, false, [ "wrapKey", "unwrapKey" ]);
```

When you create a secret, you specify a title and a secret data content.**The secret data content is encrypted using AES-GCM-256 key which is randomly generated intermediate key.**finally, this intermediate key is wrapped with your public key and concatenated with the iv used for encryption of that secret.

& this is how the AES-KEY-256 random key will be generated let's call it intermediate key.

```
   return crypto.subtle.generateKey(
       name: "AES-GCM",
       length: 256,
       true,
       ['encrypt', 'decrypt']
      );
```

Any time you want to access a secret, you need to type your master password then from your password bytes PasswordKey is generated & from passwordKey AES-GCM-256 key will be generated as discussed above, that will decrypt your private key then that will decrypt the intermediate key then finally decrypt the secret.

Using this method, it's easy to share a secret. You just need to know the exact username of your friend so you can find his public key to encrypt the intermediate key of the secret.

NOTE : ***The secret field is only decrypted when you try to access the secret.***

The "unshare" feature modifies the intermediate key of that secret data record, so it also needs to decrypt the secret to re-encrypt it with the new intermediate key.