#### **Authentication strategies -**

1. **Basic Auth** - it is a method for an HTTP User agent (e.g. a web browser) to provide a username and password when making a request. In basic HTTP authentication, a request contains a header field in the form of Authorization: Basic <credentials>, where credentials is the Base64 encoding of ID and password joined by a single colon :

**For example, if the browser uses *Aladdin* as the username and *open sesame* as the password, then the field's value is the Base64 encoding of *Aladdin:open sesame*, when we convert to base64 it becomes *QWxhZGRpbjpvcGVuIHNlc2FtZQ==*. Then the *Authorization* header field will appear as: Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==**

**Pros of Basic Auth**

1. It is very easy to implement.
2. It is very easy for the user to understand, users only have to put in their username and password.

**Cons of Basic Auth**

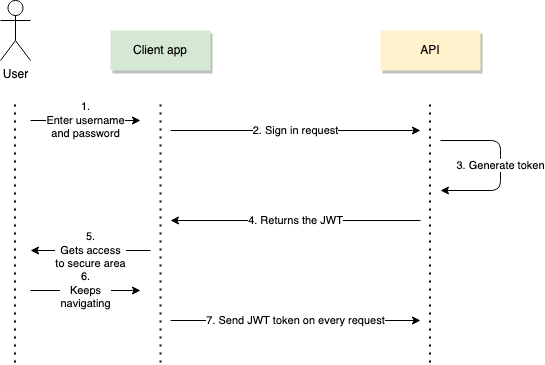
1. The BA mechanism does not provide confidentiality protection for the transmitted credential
2. Transmitted string is merely encoded with base64, it is not encrypted or hashed.
3. We have to cache the credential at client side which is a bit risky because stored string is merely encoded with base64.
4. We have to use this with conjunction with https to provide confidentiality protection

#### **2. JWT Token -** is a [proposed Internet standard](https://en.wikipedia.org/wiki/Internet_Standard#Proposed_Standard) for creating data with optional [signature](https://en.wikipedia.org/wiki/Signature_(cryptography)) and/or optional [encryption](https://en.wikipedia.org/wiki/Encryption) whose [payload](https://en.wikipedia.org/wiki/Payload_(computing)) holds [JSON](https://en.wikipedia.org/wiki/JSON) that asserts some number of [claims](https://en.wikipedia.org/wiki/Claims-based_identity). The tokens are signed either using a private secret or a [public/private key](https://en.wikipedia.org/wiki/Public-key_cryptography).

There are 3 parts of token which is elaborated in below diagram:-

| **Header** | {  **"alg"**: "HS256",  **"typ"**: "JWT"  } | Identifies which algorithm is used to generate the signature  HS256 indicates that this token is signed using HMAC-SHA256.  Typical cryptographic algorithms used are [HMAC](https://en.wikipedia.org/wiki/HMAC) with [SHA-256](https://en.wikipedia.org/wiki/SHA-256) (HS256) and [RSA signature](https://en.wikipedia.org/wiki/Digital_signature) with SHA-256 (RS256). JWA (JSON Web Algorithms) RFC 7518 introduces many more for both authentication and encryption.[[8]](https://en.wikipedia.org/wiki/JSON_Web_Token#cite_note-8) |
| --- | --- | --- |
| **Payload** | {  **"loggedInAs"**: "admin",  **"iat"**: 1422779638  } | Contains a set of claims. The JWT specification defines seven Registered Claim Names which are the [standard fields](https://en.wikipedia.org/wiki/JSON_Web_Token#Standard_fields) commonly included in tokens.[[1]](https://en.wikipedia.org/wiki/JSON_Web_Token#cite_note-rfc7519-1) Custom claims are usually also included, depending on the purpose of the token.  This example has the standard Issued At Time claim (iat) and a custom claim (loggedInAs). |
| **Signature** | HMAC\_SHA256(  secret,  base64urlEncoding(header) + '.' +  base64urlEncoding(payload)  ) | Securely validates the token. The signature is calculated by encoding the header and payload using [Base64url Encoding](https://en.wikipedia.org/wiki/Base64#The_URL_applications) [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [4648](https://datatracker.ietf.org/doc/html/rfc4648) and concatenating the two together with a period separator. That string is then run through the cryptographic algorithm specified in the header, in this case HMAC-SHA256. The *Base64url Encoding* is similar to [base64](https://en.wikipedia.org/wiki/Base64), but uses different non-alphanumeric characters and omits padding. |

**Flow As per UI perspective :**

****

**Pros of JWT Token**

1. This is stateless authentication as the user state is never saved in the server.
2. It is very much efficient database wise as verification of token is done without hitting the database
3. If token is theft like men in the middle attack, then it doesn't guarantee lifetime access unlike password.

**Cons of JWT Token**

1. It is a bit developer dependent.
2. Developer should choose the correct configuration to verify and sign the token, there should be no loop hole in that.

**2FA : -** Two Factor Authentication (2FA or TFA) is the technical term for the process of requiring a user to verify their identity in *two unique ways* before they are granted access to the system.

2FA extends the authentication paradigm by adding an additional step to the authentication process, most commonly requiring the user to enter a one-time token that is dynamically generated and delivered through a method that only the user has access to. Another common method is to use the users biometric data such as fingerprints or retina as a second factor.

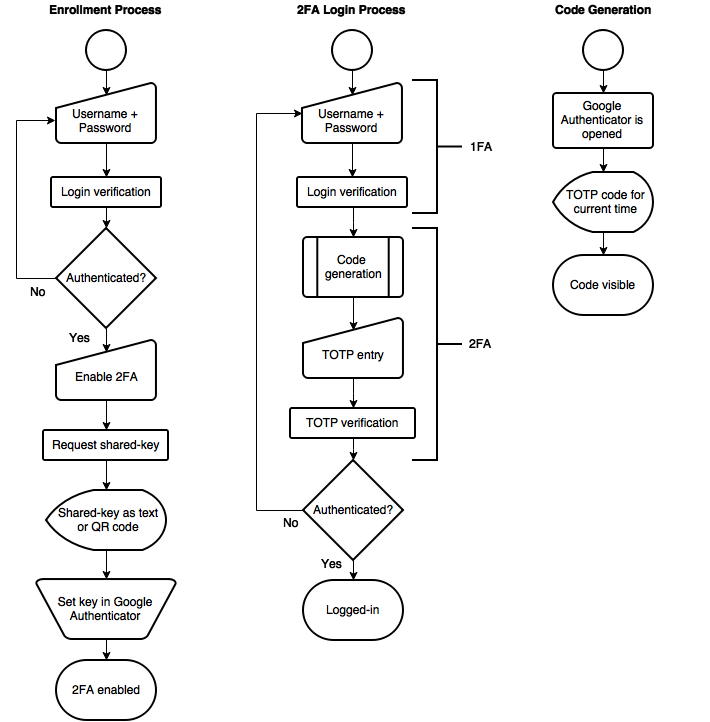
Two factor authentication is based on the user providing two of the following three ***“somethings”***:

* Something you Know – the password or pin for an account
* Something you Have – a physical device such as a mobile phone or a software application that can generate one-time passwords
* Something you Are – a biologically unique feature to you such as your fingerprints, voice or retinas

Learning the password or pin for an account is what most hackers go after. Accessing a physical to

**Types of 2FA**

1. SMS Token - This method sends the user a unique token via SMS ***text message***, normally a 5-10 digit code, after they have successfully entered their username and password.
2. Email Token - This method is very similar to the SMS method above but common implementations include having the user enter a 5-10 alpha-numeric token or clicking a link provided in the email.
3. Software Token - Software tokens require the user to download and install an application that runs on their computer or mobile device that dynamically generates tokens for the user. With the rise of smartphones – this method is gaining popularity. Software tokens work similarly to hardware tokens in that they are randomly generated and last a brief period of time before changing but developers can choose a number of different implementations to meet the business needs.
4. Phone Call - This method of 2FA calls the user once they have authenticated their username and password and provides them with the token. This is perhaps the most inconvenient method for the end-user but is a viable and common method of delivering dynamic tokens to the user.
5. Biometric Verification - This method of 2FA is unique and different from the others we mentioned so far. Biometric verification relies on the actual user being the token. A unique feature such as the users ***fingerprints*** or ***retina*** is used to verify that the user is who they say they are.



**Pros of 2FA**

1. It provides extra layer of security against weak password, same password over multiple site

**Cons of 2FA**

1. We should have backup strategies as user can lost or reset mobile phone.
2. Implementation becomes tough.

**Document Upload With Security**

**How to secure file upload**

### **Option 1: Use a third party system**

Using an off-the-shelf file upload system can be a fast way to achieve highly secure file uploads with minimal effort. If there are no special storage requirements or legacy systems to migrate, this option can be a great way for organizations to support file uploads by users. Especially if development resources are constrained.

These services usually offer enterprise-grade security with features such as antivirus scanning, user access controls, and well-documented and updated libraries in many common languages. Creating a secure file upload system using a third party is as easy as integrating their JavaScript library into the application’s frontend, then using the backend libraries to access the files as necessary by the system.

[Filestack](https://www.filestack.com/) – Previously Filepicker.io, Filestack was/is one of the first popular file uploading services. Since then, they’ve added advanced enterprise-level features while still maintaining an accessible free plan.

[Transloadit](https://transloadit.com/) – Transloadit offers an open-source uploader widget that can optionally pair with their paid file hosting and processing services.

[Cloudinary](https://cloudinary.com/documentation/upload_widget) – Specializing in publishers where uploaded content mostly comes from within the organization, Cloudinary offers an image and video upload service that specializes in advanced transformations.

[Uploadcare](https://uploadcare.com/) – A more affordable upstart that provides uploads, media processing, and adaptive delivery for web and mobile.

**AWS S3 -** We can use AWS S3 as a document storage and build all api infrastructure using API gateway and lambda function to trigger s3 upload.

### **Option 2: secure file upload in house**

The other option is to handle file uploads in-house and build protections against the most common vulnerabilities. Choosing an in-house solution is a good option for organizations that have special requirements for their data that prevent third-party vendors from being used (for example, highly sensitive documents). An in-house solution is also a good option if the file upload system is part of a larger legacy system that would result in a prohibitively expensive migration. Below are basic 8 rules that we should consider when exposing an endpoint for file upload to our server

#### **1. Create a new file name**

#### Do not use the user supplied file name as a file name on your local system. Instead, create your own unpredictable file name. Something like a hash (md5/sha1) works as it is easily validated (it is just a hex number). Maybe add a serial number or a time stamp to avoid accidental collisions. You may add a secret to the name to make it harder to guess the file name. If you need to keep the original file name: use a look-up table to link the validated user supplied file name to the server created name.

#### **2. Store the file outside of your document root**

#### If your document root is /var/www/html, create a directory /var/www/uploads and use it to store uploaded files. That way, an attacker will not be able to retrieve the file directly. This will allow you to provide fine grained access control. The file will not be parsed by the server's application language module but the source of the file will be streamed.

#### **3. Check the file size**

#### You should set a maximum file size in the upload form, but remember: It is just advisory. Make sure to check the file size after the upload completed. Be in particular careful if you allow the upload of compressed files and later uncompress them on the server. This scenario is very hard to secure.

#### **4. Extensions are meaningless**

#### The motivation for this post is the ';' issue in IIS. However, even Apache doesn't always behave the way you expect it to. Try 'something.php.x' in Apache and chances are that php code will be executed. Its a feature ? . If you stream a file back to the user, the extension isn't what matters, but the Content-Type header and the file's header. It is best to use the "file" command on unix to check the file type. But even this is not fool proof. It will just check the first few bytes. In PHP for example, a file may start with a GIF header, but later if the PHP engine sees a "*<?php*" tag, it will happily interpret an embedded PHP script.

#### **5. Try a malware scan**

#### The extension is right, and you checked that the file is actually a valid JPEG file per it's header. However, it could still be a malicious JPEG using one of the many image parser bugs to exploit clients downloading the file. There is no great defense against this as far as I am aware. One possible work around is to "rebuild" the file. Convert the JPEG to a GIF and back to a JPEG. This will likely strip out any malicious feature. But this technique could expose your servers to just the same image parser bugs.

#### **6. Keep tight control of permissions**

#### Any uploaded file will be owned by the web server. But it only needs read/write permission, not execute permissions. After the file is downloaded, you could apply additional restrictions if this is appropriate. Sometimes it can be helpful to remove the execute permission from directories to prevent the server from enumerating files.

#### **7. Authenticate file uploads**

#### File uploads, in particular if these files are viewable by others without moderator review, have to be authenticated. This way it is at least possible to track who uploaded an objectionable file.

#### **8. Limit the number of uploaded files**

#### Many developers limit the file size, but not all limit the number of files uploaded in a request. Make sure to apply reasonable limits. But be also ready for a DoS attack that just uploads a large number of small files. Pick an appropriate directory structure to limit the number of files per directory and pick an appropriate file system.

#### 