Session ID or Session Token:

- Is generated by the server and needs to be sent along every request.
- If any session ID is lost, then a hacker can make requests pretending to the server that the hacker is with the right session ID.

How do we send this secret token?
Standard way is to send it as an HTTP header parameter: Authorization.

Do we need to send the token manually with every single request?

- Cookies are special HTTP headers, that once set, the browser will keep sending along with every request to that server.
- Cookies are tied to a specific domain.
- Cookies have an expiry.
- A server can request the client to set a cookie with a particular name and value.
- The client may or may not decide to respect that.
- If the client does agree, then it sends a cookie header that contains the name and value.
- Cookies are completely under the control of a browser.
- Cookies max size: 4kb
- Session cookie expires when browser is closed.
- Persistent cookie expires at specific date.

Encoding:

- The purpose of encoding is to transform data so that it can be properly (and safely) consumed by a different type of system, e.g. binary data being sent over email, or viewing special characters on a web page. The goal is not to keep information secret, but rather to ensure that it's able to be properly consumed.
- Encoding transforms data into another format using a scheme that is publicly available so that it can easily be reversed. It does not require a key as the only thing required to decode it is the algorithm that was used to encode it.

E.g. ASCII, UNICODE, URL Encoding, BASE 64

Encryption:

- The purpose of encryption is to transform data in order to keep it secret from others.
- Goal is to ensure the data cannot be consumed by anyone other than the independent recipients.
- Encryption transforms data into another format in such a way that only specific individual(s) can reverse the transformation. It uses a key, which is kept secret, in conjunction with the plaintext and the algorithm, in order to perform the encryption operation. As such, the ciphertext, algorithm, and key are all required to return to the plaintext.

E.g. AES (Advanced encryption standard), 3DES (or Triple DES or TDES) (triple Data encryption algorithm), RSA, BLOWFISH

Hash Function:

- Is a function that converts any string into an utterly random fixed length string.
- Hashes are amazing mathematical functions because it is extremely hard to convert that random string back to original text. A one way function.
- The same input will always produce the same output.
- Multiple disparate input should not produce the same output.
- Any modification to the given input should result in a drastic change to the hash.

Usage:
Passwords can be hashed.

E.g. SHA-3, MD5, etc

Obfuscation:

- The purpose of obfuscation is to make something harder to understand, usually for the purposes of making it more difficult to attack or to copy.
- One common use is the the obfuscation of source code so that it's harder to replicate a given product if it is reverse engineered.
- It's important to note that obfuscation is not a strong control (like properly employed encryption) but rather an obstacle. It, like encoding, can often be reversed by

Single Sign On:

- Is a property of access control of multiple related, yet independent, software systems.
- With this property, a user logs in with a single ID and password to gain access to a connected systems.

Benefits:

- Mitigate risk for access to 3rd party sites.
- Reduce password fatigue from username and password combinations.

using the same technique that obfuscated it. Other times it is simply a manual process that takes time to work through.
- Another key thing to realize about obfuscation is that there is a limitation to how obscure the code can become, depending on the content being obscured. If you are obscuring computer code, for example, the limitation is that the result must still be consumable by the computer or else the application will cease to function.

E.g. javascript obfuscator, proguard

- Reduce time spent re-entering passwords for the same identity.
- Reduce IT costs due to lower number of IT help desk calls about passwords.

Basic AuthN:
- is a simple authentication scheme built into the HTTP protocol.
- The client sends HTTP requests with the Authorization header that contains the word Basic word followed by a space and a base64-encoded string username:password.
- E.g. Authorization: Basic ZGVtbzpwQDU1dzByZA==
- Because base64 is easily decoded, Basic authentication should only be used together with other security mechanisms such as HTTPS/SSL.

Authentication Server:
- An application that facilitates authN of an entity that attempts to access over a n/w.
- An authn server can reside in a dedicated computer, an ethernet switch, an access point or a b/w access server.

AuthN:
- Authentication is the process of determining whether someone or something is actually who or what it declares itself to be.

AuthZ:
- Authorization is the process of giving someone permission to do or have something.
- Assuming that someone has logged in to a computer operating system or application, the system or application may want to identify what resources the user can be given during this session. Thus, authorization is sometimes seen as both the preliminary setting up of permissions by a system administrator and the actual checking of the permission values that have been set up when a user is getting access.

Single Sign On protocols:
- Based on the principle of an authN server, a lot of SSO standards have been created.
- Popular ones are:
  - CAS
  - SAML2
  - OpenID connect

Digest AuthN:
- Is an enhanced method of single-factor authentication (SFA).
- a request from a potential user is received by a network server and then sent to a domain controller.

Steps Involved:
- A client requests access to a website with a username and a password.
- The server responds with a digest session key, a nonce and 401 authentication.
- The client answers with a response array with a composition of (username:realm:Password) which is encrypted using MD5.
- The server employs the username and realm to look up the password in the db, then uses that password to create a md5 key using (username:realm:passwordFromDB)
- Then the server compares its generated MD5 key to clients submitted MD5 key. If it matches, the client is authenticated.If not, the client is denied access.

realm: A String which will be used with the UI and as part of the hash.
qop: Can be auth and auth-int and has influence on how the hash is created. We use auth.
nonce: A unique code, which will be used with the hash and needs to be sent back by the client.
Opaque: This can be treated as a session id. If this changes the browser will deauthN the use

Drawbacks:
- Fact that the single factor (the password or user response) is relatively easy for an experienced hacker to discover and exploit.
- Is vulnerable to replay attacks, to a limited extent.

Superior security can be obtained by the use of a two-factor authentication scheme, in which a physical token such as smart card is employed in addition to the password or keyboard-generated response to verify the identity of a potential user.

SAML2:

- Security Assertion Markup Language (SAML) is a standard for logging users into applications based on their sessions in another context. This single sign-on (SSO) login standard has significant advantages over logging in using a username/password:
    - No need to type in credentials
    - No need to remember and renew passwords
    - No weak passwords

How SAML works?

- SAML SSO works by transferring the user's identity from one place (the identity provider) to another (the service provider). This is done through an exchange of digitally signed XML documents.
- Consider the following scenario: A user is logged into a system that acts as an identity provider. The user wants to log in to a remote application, such as a support or accounting application (the service provider). The following happens:
    - The user accesses the remote application using a link on an intranet, a bookmark, or similar and the application loads.
    - The application identifies the user's origin (by application subdomain, user IP address, or similar) and redirects the user back to the identity provider, asking for authentication. This is the authentication request.
    - The user either has an existing active browser session with the identity provider or establishes one by logging into the identity provider.
    - The identity provider builds the authentication response in the form of an XML-document containing the user's username or email address, signs it using an X.509 certificate, and posts this information to the service provider.
    - The service provider, which already knows the identity provider and has a certificate fingerprint, retrieves the authentication response and validates it using the certificate fingerprint.
    - The identity of the user is established and the user is provided with app access.

SAML SSO Flow:

SAML2 protocol:

OpenID connect:

- OpenID Connect is a simple identity layer built on top of the OAuth 2.0 protocol, which allows clients to verify the identity of an end user based on the authentication performed by an authorization server or identity provider (IdP), as well as to obtain basic profile information about the end user in an interoperable and REST-like manner. OpenID Connect specifies a RESTful HTTP API, using JSON as a data format.
- OpenID Connect is an increasingly common authentication protocol: when an app prompts you to authenticate using your Facebook or Google+ credentials, the app is probably using OpenID Connect.
- OpenID Connect allows a range of clients, including web-based, mobile, and JavaScript clients, to request and receive information about authenticated sessions and end-users.

Why OpenID:

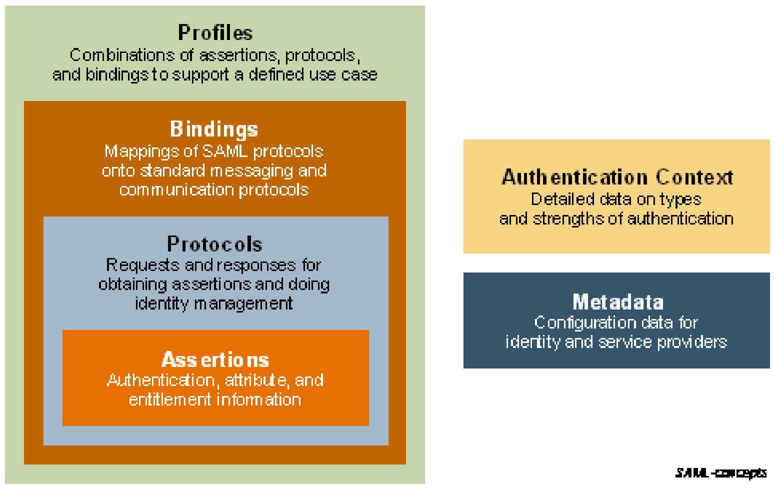
- OpenID Connect is easier to integrate than SAML, and it can work with a wider variety of apps. Specifically, it provides:
    - Easily consumed identity tokens. Client apps receive the user's identity encoded in a secure JSON Web Token (JWT) called the ID token. JWTs are elegant and portable and support a range of signature and encryption algorithms.
    - The OAuth 2.0 protocol. Clients use OAuth 2.0 flows to obtain ID tokens, which work with web apps as well as native mobile apps. OAuth 2.0 also means that you have a single protocol for authentication and authorization (obtaining access tokens).
    - Simplicity with capability. OpenID Connect is simple enough to integrate with basic apps, while also offering features and security options that can meet demanding enterprise requirements.

Grant Types:
OpenID Connect supports the following authentication flows:

- The Implicit Flow is required for apps that have no "back end" logic on the web server, like a Javascript app.
- The Authentication (or Basic) Flow is designed for apps that have a back end that can communicate with the IdP away from prying eyes.
- The Resource Owner Password Grant does not have an login UI and is useful when access to a web browser is not possible.
- The Client Credentials Grant is useful for machine to machine authorization.

Implicit Flow:

- The Implicit flow is required for apps and websites that have no back end logic on the web server, and


Profiles
Combinations of assertions, protocols, and bindings to support a defined use case
Bindings
Mappings of SAML protocols onto standard messaging and communication protocols
Protocols
Requests and responses for obtaining assertions and doing identity management
Assertions
Authentication, attribute, and entitlement information
Authentication Context
Detailed data on types and strengths of authentication
Metadata
Configuration data for identity and service providers
SAML-concepts

SAML profiles:

- SAML profiles are defined to satisfy a particular business use case, for example the Web Browser SSO profile.
- Profiles typically define constraints on the contents of SAML assertions, protocols, and bindings in order to solve the business use case in an interoperable fashion.
- There are also Attribute Profiles, which do not refer to any protocol messages and bindings, that define how to exchange attribute information using assertions in ways that align with a number of common usage environments.

Metadata:

- Metadata defines a way to express and share configuration information between SAML parties. For instance, an entity's supported SAML bindings, operational roles (IDP, SP, etc), identifier information, supporting identity attributes, and key information for encryption and signing can be expressed using SAML metadata XML documents. SAML Metadata is defined by its own XML schema.

AuthN context:

- In a number of situations, a service provider may need to have detailed information regarding the type and strength of authentication that a user employed when they authenticated at an identity provider.
- A SAML authentication context is used in (or referred to from) an assertion authentication statement to carry this information.
- An SP can also include an authentication context in a request to an IdP to request that the user be authenticated using a specific set of authentication requirements, such as a multi-factor authentication.

everything that is passed between the app or site and the IdP can be viewed using browser development tools.

Steps:

- the user attempts to start a session with your client app and is redirected to the OpenID Provider (OneLogin), passing in the client ID, which is unique for that application.
- the OpenID Provider authenticates and authorizes the user for a particular application instance.
- user details are encoded by the OpenID Provider into an id_token (JWT) that contains user information (scopes, in OAuth terms) and signature (using RS256), which is passed to a preconfigured Redirect page on the web server.
- the client app confirms the JWT id_token and confirms the signature using the public key. If everything is fine, a session is established for the user.

Authentication Flow:

- The Authentication (or Basic) flow is an option for apps that have web-server logic that enables back-end communication with the IdP (OneLogin). It functions like a traditional three-legged OAuth flow and results in a traditional OAuth access token being returned in secret to the web application via calls made on the back end. In this flow, rather than transmit the user details, the provider sends a special, one-time-use code that can be exchanged by the back-end web service for an OAuth access token. This exchange needs to include the client id and client secret in addition to the code, just like a traditional OAuth 2.0 flow. It is more secure than the Implicit flow, because tokens are not visible through the browser and the client app can also be authenticated.

Steps:

- the user attempts to start a session with your client app and is redirected to the OpenID Provider (OneLogin), passing in the client ID, which is unique for that application.
- the OpenID Provider authenticates and authorizes the user for a particular application instance. So far, it looks like the Implicit flow.
- a one-time-use code is passed back to the web server using a predefined Redirect URI.
- the web server passes the code, client ID, and client secret to the OpenID Provider token endpoint, and the OpenID Provider validates the code and returns a one-hour access token.
- the web server uses the access token to get further details about the user (if necessary) and establishes a session for the user.

SAML components:

SAML Bindings:

- Assertions
- Protocols
- Bindings

SAML Assertions:

- SAML allows for one party to assert security information in the form of statements about a subject. For instance, a SAML assertion could state that the subject is named "John Doe", has an email address of john.doe@example.com, and is a member of the "engineering" group.
- An assertion contains some basic required and optional information that applies to all its statements, and usually contains a subject of the assertion (if not present, the identity determined through other means, e.g. the certificate used for subject confirmation), conditions used to validate the assertion, and assertion statements.
- SAML defines three kinds of statements that can be carried within an assertion:
    - Authentication statements: These are created by the party that successfully authenticated a user. At a minimum, they describe the particular means used to authenticate the user and the specific time at which the authentication took place.
    - Attribute statements: These contain specific identifying attributes about the subject (for example, that user "John Doe" has "Gold" card status).
    - Authorization decision statements: These define something that the subject is entitled to do (for example, whether "John Doe" is permitted to buy a specified item).

SAML Protocols:
SAML defines a number of generalized request/response protocols:

- Authentication Request Protocol: Defines a means by which a principal (or an agent acting on behalf of the principal) can request assertions containing authentication statements and, optionally, attribute statements. The Web Browser SSO Profile uses this protocol when redirecting a user from an SP to an IdP when it needs to obtain an assertion in order to establish a security context for the user at the SP.
- Single Logout Protocol: Defines a mechanism to allow near-simultaneous logout of active sessions associated with a principal. The logout can be directly initiated by the user, or initiated by an IdP or SP because of a session timeout, administrator command, etc.
- Assertion Query and Request Protocol: Defines a set of queries by which SAML assertions may be obtained. The Request form of this protocol can ask an asserting party

SAML bindings detail exactly how the various SAML protocol messages can be carried over underlying transport protocols. The bindings defined by SAML V2.0 are:

- HTTP Redirect Binding: Defines how SAML protocol messages can be transported using HTTP redirect messages (302 status code responses).
- HTTP POST Binding: Defines how SAML protocol messages can be transported within the base64-encoded content of an HTML form control.
- HTTP Artifact Binding: Defines how an artifact (described above in the Artifact Resolution Protocol) is transported from a message sender to a message receiver using HTTP. Two mechanisms are provided: either an HTML form control or a query string in the URL.
- SAML SOAP Binding: Defines how SAML protocol messages are transported within SOAP 1.1 messages, with details about using SOAP over HTTP.
- Reverse SOAP (PAOS) Binding: Defines a multi-stage SOAP/HTTP message exchange that permits an HTTP client to be a SOAP responder. Used in the Enhanced Client and Proxy Profile to enable clients and proxies capable of assisting in IDP discovery.
- SAML URI Binding: Defines a means for retrieving an existing SAML assertion by resolving a URI (uniform resource identifier).

SAML Profiles:
SAML profiles define how the SAML assertions, protocols, and bindings are combined and constrained to provide greater interoperability in particular usage scenarios. Some of these profiles are examined in detail later in this document. The profiles defined by SAML V2.0 are:

- Web Browser SSO Profile: Defines how SAML entities use the Authentication Request Protocol and SAML Response messages and assertions to achieve single sign-on with standard web browsers. It defines how the messages are used in combination with the HTTP Redirect, HTTP POST, and HTTP Artifact bindings.
- Enhanced Client and Proxy (ECP) Profile: Defines a specialized SSO profile where specialized clients or gateway proxies can use the Reverse-SOAP (PAOS) and SOAP bindings.
- Identity Provider Discovery Profile: Defines one possible mechanism for service providers to learn about the identity providers that a user has previously visited.
- Single Logout Profile: Defines how the SAML Single Logout Protocol can be used with SOAP, HTTP Redirect, HTTP POST, and HTTP Artifact bindings.
- Assertion Query/Request Profile: Defines how SAML entities can use the SAML Query and Request Protocol to obtain SAML assertions over a synchronous binding, such as SOAP.
- Artifact Resolution Profile: Defines how SAML entities

for an existing assertion by referring to its assertion ID. The Query form of this protocol defines how a relying party can ask for assertions (new or existing) on the basis of a specific subject and the desired statement type.

- Artifact Resolution Protocol: Provides a mechanism by which SAML protocol messages may be passed by reference using a small, fixed-length value called an artifact. The artifact receiver uses the Artifact Resolution Protocol to ask the message creator to dereference the artifact and return the actual protocol message. The artifact is typically passed to a message recipient using one SAML binding (e.g. HTTP Redirect) while the resolution request and response take place over a synchronous binding, such as SOAP.
- Name Identifier Management Protocol: Provides mechanisms to change the value or format of the name identifier used to refer to a principal. The issuer of the request can be either the service provider or the identity provider. The protocol also provides a mechanism to terminate an association of a name identifier between an identity provider and service provider.
- Name Identifier Mapping Protocol: Provides a mechanism to programmatically map one SAML name identifier into another, subject to appropriate policy controls. It permits, for example, one SP to request from an IdP an identifier for a user that the SP can use at another SP in an application integration scenario.

OAuth 2.0:

- OAuth 2 is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, such as Facebook, GitHub, and DigitalOcean. It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account. OAuth 2 provides authorization flows for web and desktop applications, and mobile devices.
- OpenID connect is the protocol built on OAuth2.0

JSON web token (JWT):

- JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.
- Signed tokens can verify the integrity of the claims contained within it, while encrypted tokens hide those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

can use the Artifact Resolution Protocol over a synchronous binding, such as SOAP, to obtain the protocol message referred to by an artifact.

- Name Identifier Management Profile: Defines how the Name Identifier Management Protocol may be used with SOAP, HTTP Redirect, HTTP POST, and HTTP Artifact bindings.
- Name Identifier Mapping Profile: Defines how the Name Identifier Mapping Protocol uses a synchronous binding such as SOAP.

Sample SAML Request:

```
<samlp:AuthnRequest
xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
ID="identifier_1"
Version="2.0"
IssueInstant="2004-12-05T09:21:59Z"
AssertionConsumerServiceIndex="1">
<saml:Issuer>https://sp.example.com/SAML2</saml:Issuer>
<samlp:NameIDPolicy
AllowCreate="true"
Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"/>
</samlp:AuthnRequest>
```

Sample SAML Response:

```
<samlp:Response
xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
ID="identifier_2"
InResponseTo="identifier_1"
Version="2.0"
IssueInstant="2004-12-05T09:22:05Z"
Destination="https://sp.example.com/SAML2/SSO/POST">
<saml:Issuer>https://idp.example.org/SAML2</saml:Issuer>
<samlp:Status>
<samlp:StatusCode
Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
</samlp:Status>
<saml:Assertion
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
ID="identifier_3"
Version="2.0"
IssueInstant="2004-12-05T09:22:05Z">
<saml:Issuer>https://idp.example.org/SAML2</saml:Issuer>

<!-- a POSTed assertion MUST be signed -->
<ds:Signature
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...</ds:Signature>
<saml:Subject>

<saml:NameID
Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient">
3f7b3dcf-1674-4ecd-92c8-1544f346baf8
</saml:NameID>

<saml:SubjectConfirmation
Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">

<saml:SubjectConfirmationData
InResponseTo="identifier_1"
Recipient="https://sp.example.com/SAML2/SSO/POST"
NotOnOrAfter="2004-12-05T09:27:05Z"/>

</saml:SubjectConfirmation>

</saml:Subject>

<saml:Conditions
NotBefore="2004-12-05T09:17:05Z"
```

Where should we use JWT?

- Authorization: This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.
- Information Exchange: JSON Web Tokens are a good way of securely transmitting information between parties. Because JWTs can be signed—for example, using public/private key pairs—you can be sure the senders are who they say they are. Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

JWT structure:

- Header: The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.
  E.g.
  ```
  {
    "alg": "HS256",
    "typ": "JWT"
  }
  ```
- Payload: The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional data. There are three types of claims: registered, public, and private claims.
  - Registered claims: These are a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. Some of them are: iss (issuer), exp (expiration time), sub (subject), aud (audience), and others.
  - Public claims: These can be defined at will by those using JWTs. But to avoid collisions they should be defined in the IANA JSON Web Token Registry or be defined as a URI that contains a collision resistant namespace.
  - Private claims: These are the custom claims created to share information between parties that agree on using them and are neither registered or public claims.

  E.g.
  ```
  {
    "sub": "1234567890",
    "name": "John Doe",
    "admin": true
  }
  ```
- Signature: To create the signature part you have to take the encoded header, the encoded payload, a secret, the

```
NotOnOrAfter="2004-12-05T09:27:05Z">
<saml:AudienceRestriction>
<saml:Audience>https://sp.example.com/SAML2</saml:Audience>
</saml:AudienceRestriction>
</saml:Conditions>
<saml:AuthnStatement
AuthnInstant="2004-12-05T09:22:00Z"
SessionIndex="identifier_3">
<saml:AuthnContext>
<saml:AuthnContextClassRef>
urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
</saml:AuthnContextClassRef>
</saml:AuthnContext>
</saml:AuthnStatement>
</saml:Assertion>
</samlp:Response>
```

Asymmetric Cryptography:

- Asymmetric cryptography, also known as public key cryptography, uses public and private keys to encrypt and decrypt data. The keys are simply large numbers that have been paired together but are not identical (asymmetric). One key in the pair can be shared with everyone; it is called the public key. The other key in the pair is kept secret; it is called the private key. Either of the keys can be used to encrypt a message; the opposite key from the one used to encrypt the message is used for decryption.
- Many protocols like SSH, OpenPGP, S/MIME, and SSL/TLS rely on asymmetric cryptography for encryption and digital signature functions.

Digital signatures:

- Digital signatures are based on asymmetric cryptography and can provide assurances of evidence to origin, identity and status of an electronic document, transaction or message, as well as acknowledging informed consent by the signer.
- To create a digital signature, signing software (such as an email program) creates a one-way hash of the electronic data to be signed.
- The user's private key is then used to encrypt the hash, returning a value that is unique to the hashed data.
- The encrypted hash, along with other information such as the hashing algorithm, forms the digital signature.
- Any change in the data, even to a single bit, results in a different hash value.
- This attribute enables others to validate the integrity of the data by using the signer's public key to decrypt the hash.
- If the decrypted hash matches a second computed hash of the same data, it proves that the data hasn't changed since it was signed.
- If the two hashes don't match, the data has either been tampered with in some way (indicating a failure of integrity) or the signature was created with a private key that doesn't correspond to the public key presented by the signer (indicating a failure of authentication).

algorithm specified in the header, and sign that. For example if you want to use the HMAC SHA256 algorithm, the signature will be created in the following way:
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)	

Note: Information gathered in this document has been collected from various sources on the Internet.