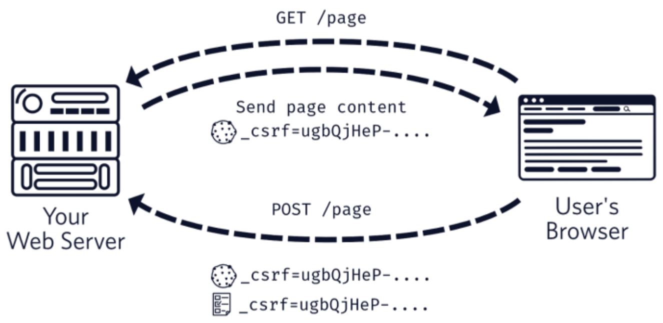
Cross Site Request Forgery aka CSRF/XSRF

An attack that tricks a user to execute unwanted actions on a web application in which they're authenticated. Typically executed on the functionality that uses form-based submissions like POST requests and cookie-based authentication. A hidden form into a malicious page automatically performs a POST request to your page's endpoint. The browser then automatically sends all the cookies stored for that page along with the request. If a user is logged into a current session, the attacker could, for example, post a message on behalf of the logged in user without them noticing. The attacker never has to have access to the page's cookies for this.

The most popular implementation to **prevent** Cross-site Request Forgery (CSRF), is to make use of a token that is associated with a particular user and can be found as a hidden value. This token, called a *CSRF Token* or a *Synchronizer Token*, works as follows:

1. The client requests an HTML page that contains a form.
2. The server includes two tokens in the response. One token is sent as a cookie. The other is placed in a hidden form field. The tokens are generated randomly so that an adversary cannot guess the values.
3. When the client submits the form, it must send both tokens back to the server. The client sends the cookie token as a cookie, and it sends the form token inside the form data. (A browser client automatically does this when the user submits the form.)
4. If a request does not include both tokens, the server disallows the request.



The cookie-parser middleware will check that the token is available in the cookies and csurf will be the automatic guard for any POST, PUT, PATCH or DELETE operations by checking that the _csrf token is present in both the cookies and the request body and that they match.

Key exchange in a HTTPS handshake

Anyone can intercept every single one of the messages you exchange with a server, including the ones where you are agreeing on the key and encryption strategy to use, and still not be able to read any of the actual data you send.

The client generates a random key to be used for the main, symmetric algorithm. It encrypts it using an algorithm agreed upon during the Hello phase, and the server's public key (found on its SSL certificate). It sends this encrypted key to the server, where it is decrypted using the server's private key, and the interesting parts of the handshake are complete.
HTTP requests and responses can now be sent by forming a plaintext message and then encrypting and sending it. **The other party is the only one who knows how to decrypt this message,** and so **Man in the Middle Attackers are unable to read or modify any requests that they may intercept**.

Password Salting

A form of password encryption that involves appending a password to a given username and then hashing the new string of characters. Salts defend against dictionary attacks since they are different in each case. They also protect commonly used passwords, or those users who use the same password on several sites, by making all salted hash instances for the same password different from each other.