# **Types of Authentication**

A servlet-based web application can choose from the following types of authentication, from least secure to most:

* Basic authentication
* Form-based authentication
* Digest authentication
* SSL and client certificate authentication

“Authentication” and “Authorization”. Authentication can be defined as the process of verifying someone’s identity by using pre-required details (Commonly username and password). Authorization is the process of allowing an authenticated user to access a specified resource (eg:-right to access a file).

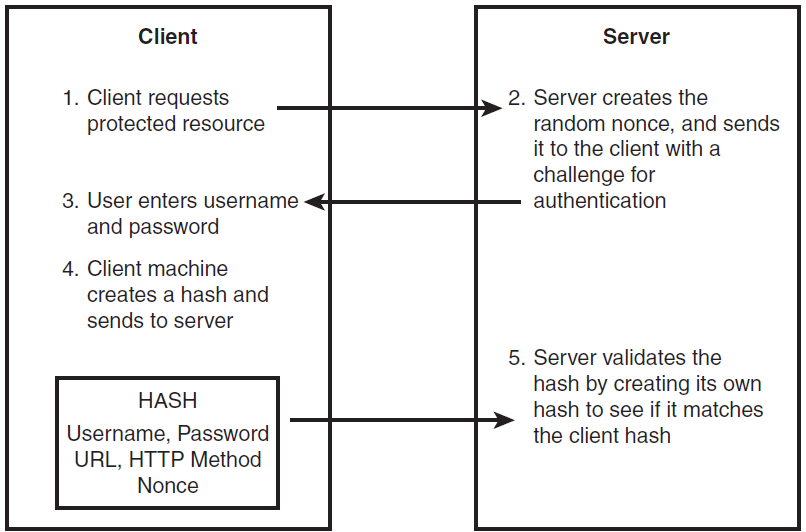
## HTTP Basic Authentication

* One solution is that of **HTTP Basic Authentication**. In this approach, an HTTP user agent simply provides a **username**and **password** to prove their authentication.
* This approach does not require cookies, session IDs, login pages, and other such specialty solutions, and because it uses the **HTTP header** itself, there’s no need to handshakes or other complex response systems.
* HTTP is not [encrypted](https://nordicapis.com/securing-your-datastream-with-p2p-encryption/) in any way. It is encapsulated in base64, and is often erroneously proclaimed as encrypted due to this

## Digest authentication

The difference between digest authentication and basic authentication is that in digest authentication, the username and password are never sent over the wire. Instead, a hash is created made up of the following pieces of information:

* The username
* The password
* The URL
* The randomly generated string (the nonce)
* The HTTP method being used

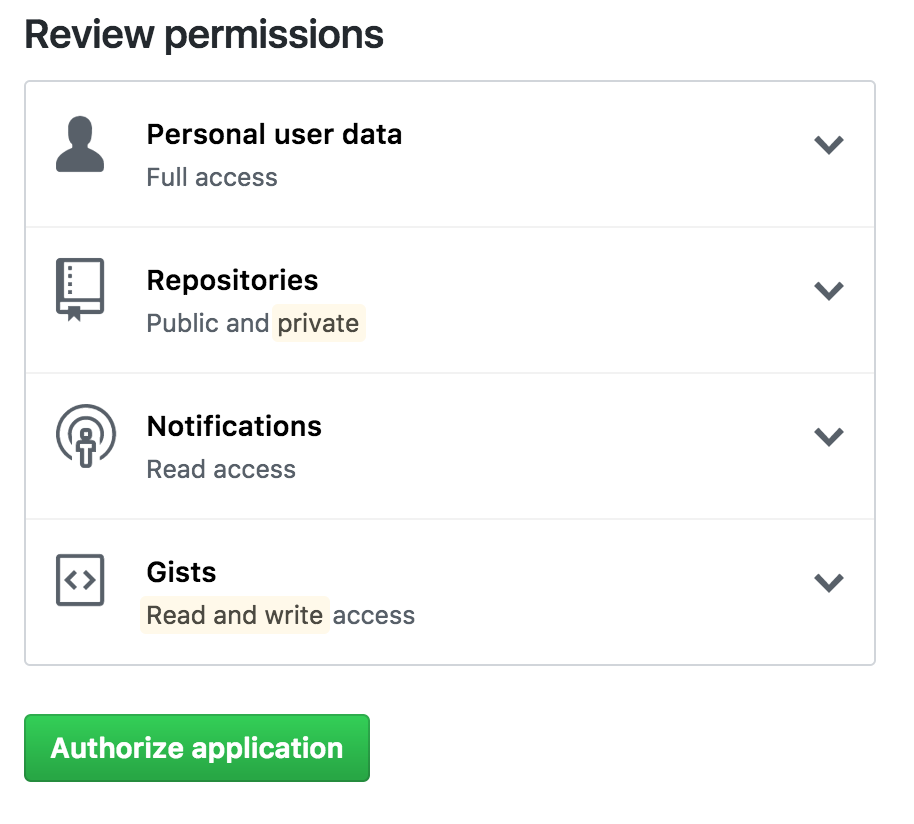


## API Keys : for Developer Quickstart

* To access Bitbucket in Hygieia, we will generate API key, and we will place that key in properties file.
* API Keys can be used as Basic HTTP Authentication credentials and provide a substitute for the account's actual username and password.
* The best thing about an API key is its simplicity. You merely log in to a service, find your API key (often in the settings screen), and copy it to use in an application, test in the browser, or use with one of these [API request tools](https://zapier.com/engineering/api-request-tools/)
* Typically, an API key gives full access to every operation an API can perform, including writing new data or deleting existing data. If you use the same API key in multiple apps, a broken app could destroy your users' data without an easy way to stop just that one app.
* Many API keys are sent in the query string as part of the URL, which makes it easier to discover for someone who should not have access to it. A better option is to put the API key in the Authorization header. In fact, that’s the [proposed standard](https://tools.ietf.org/html/rfc7235?utm_source=zapier.com&utm_medium=referral&utm_campaign=zapier):  
  Authorization: Apikey 1234567890abcdef

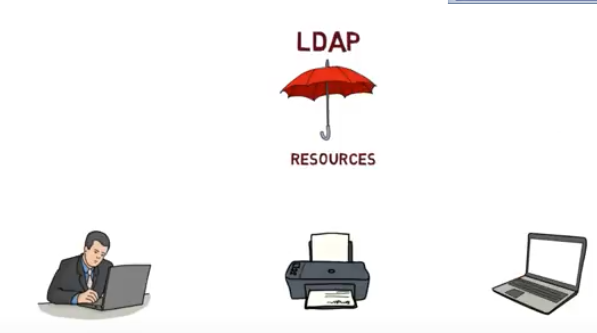
## OAuth Tokens: Great for Accessing User Data

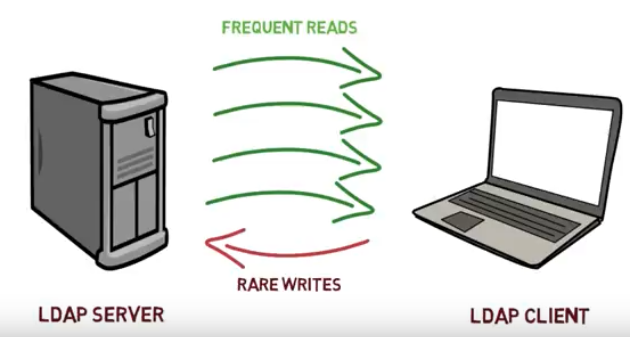
* OAuth is the answer to accessing user data with APIs.
* users simply click a button to allow an application to access their accounts.

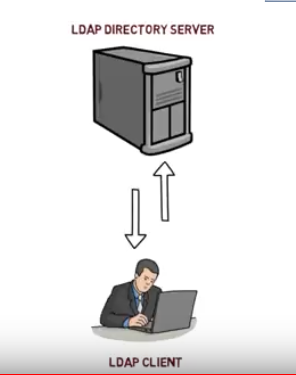


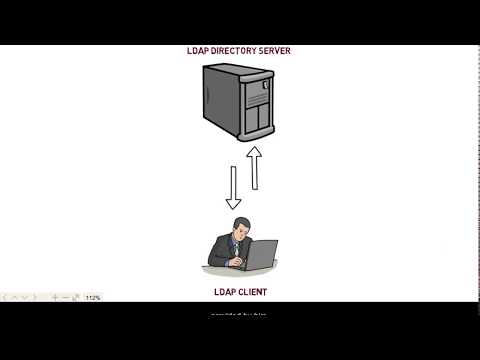
## LDAP









[](https://www.youtube.com/watch?v=lp5z8HQGAH8)

<https://www.youtube.com/watch?v=lp5z8HQGAH8>

import javax.naming.\*;

import javax.naming.directory.\*;

import java.util.Hashtable;

class Simple {

public static void main(String[] args) {

Hashtable authEnv = new Hashtable(11);

String userName = "johnlennon";

String passWord = "sushi974";

String base = "ou=People,dc=example,dc=com";

String dn = "uid=" + userName + "," + base;

String ldapURL = "ldap://ldap.example.com:389";

authEnv.put(Context.INITIAL\_CONTEXT\_FACTORY,"com.sun.jndi.ldap.LdapCtxFactory");

authEnv.put(Context.PROVIDER\_URL, ldapURL);

authEnv.put(Context.SECURITY\_AUTHENTICATION, "simple");

authEnv.put(Context.SECURITY\_PRINCIPAL, dn);

authEnv.put(Context.SECURITY\_CREDENTIALS, passWord);

try {

DirContext authContext = new InitialDirContext(authEnv);

System.out.println("Authentication Success!");

} catch (AuthenticationException authEx) {

System.out.println("Authentication failed!");

} catch (NamingException namEx) {

System.out.println("Something went wrong!");

namEx.printStackTrace();

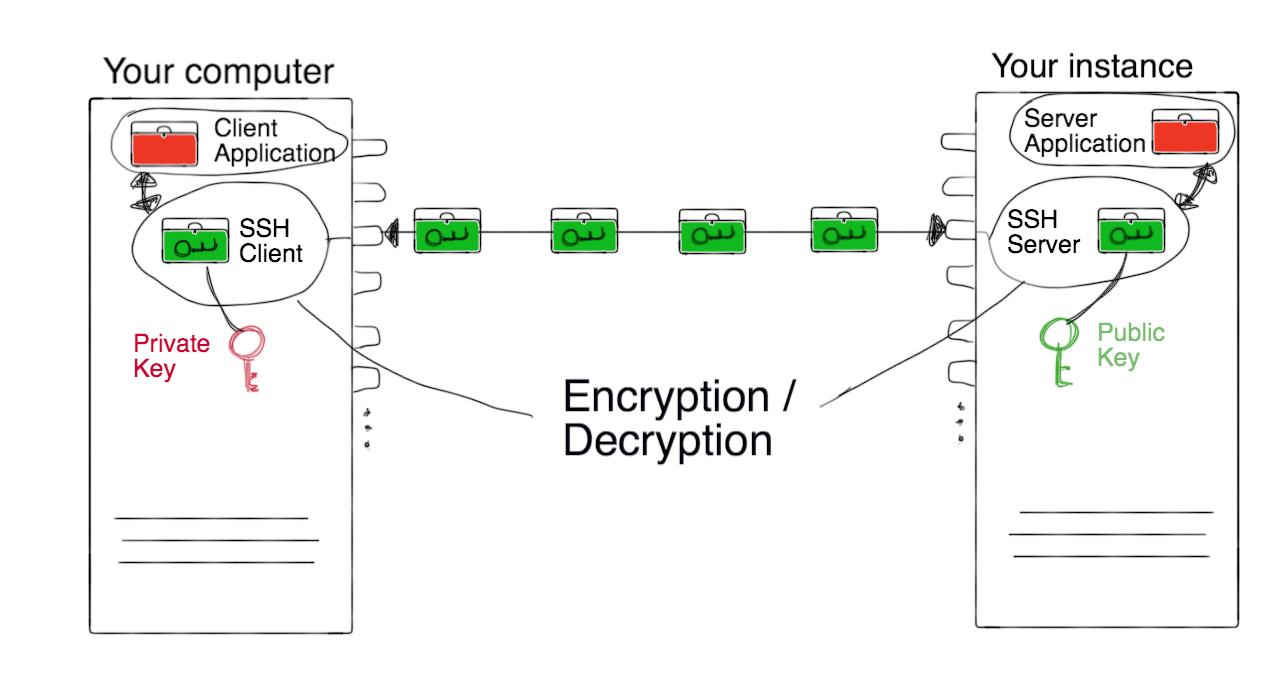
}

}

}

## SSH – Only for LINUX Server / CommadLine(git) related Access

* SSH, or secure shell, is an encrypted protocol used to administer and communicate with servers. When working with a Linux server, chances are, you will spend most of your time in a terminal session connected to your server through SSH.
* An SSH server can authenticate clients using a variety of different methods. The most basic of these is password authentication, which is easy to use, but not the most secure.
* SSH key pairs are two cryptographically secure keys that can be used to authenticate a client to an SSH server. Each key pair consists of a public key and a private key.
* The **private key** is retained by the client and should be kept absolutely secret. Any compromise of the private key will allow the attacker to log into servers that are configured with the associated public key without additional authentication. As an additional precaution, the key can be encrypted on disk with a passphrase.
* The associated public key can be shared freely without any negative consequences. The public key can be used to encrypt messages that only the private key can decrypt. This property is employed as a way of authenticating using the key pair.
* The public key is uploaded to a remote server that you want to be able to log into with SSH. The key is added to a special file within the user account you will be logging into called ~/.ssh/authorized\_keys.
* When a client attempts to authenticate using SSH keys, the server can test the client on whether they are in possession of the private key. If the client can prove that it owns the private key, a shell session is spawned or the requested command is executed.



## Base64 – not Authentication

represent [binary data](https://en.wikipedia.org/wiki/Binary_data) in an [ASCII](https://en.wikipedia.org/wiki/ASCII) string format

Each Base64 digit represents exactly 6 bits of data.

Steps:

* take three ASCII numbers 155, 162, and 233
* Convert into binary stream formate 100110111010001011101001
* groupings of six characters: 100110 111010 001011 101001.
* The binary string 100110 converts to the decimal number 38: 0\*2^01 + 1\*2^1 + 1\*2^2 + 0\*2^3 + 0\*2^4 + 1\*2^5 = 0+2+4+0+0+32.
* Base64 6-bit values 38, 58, 11 and 41.
* Using the Base64 conversion table:
  + 38 is m
  + 58 is 6
  + 11 is L
  + 41 is p