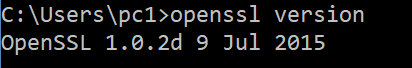
# Exercises: Playing with Cryptography, Public / Private Keys, Hashing, Blockchain Addresses and OpenSSL

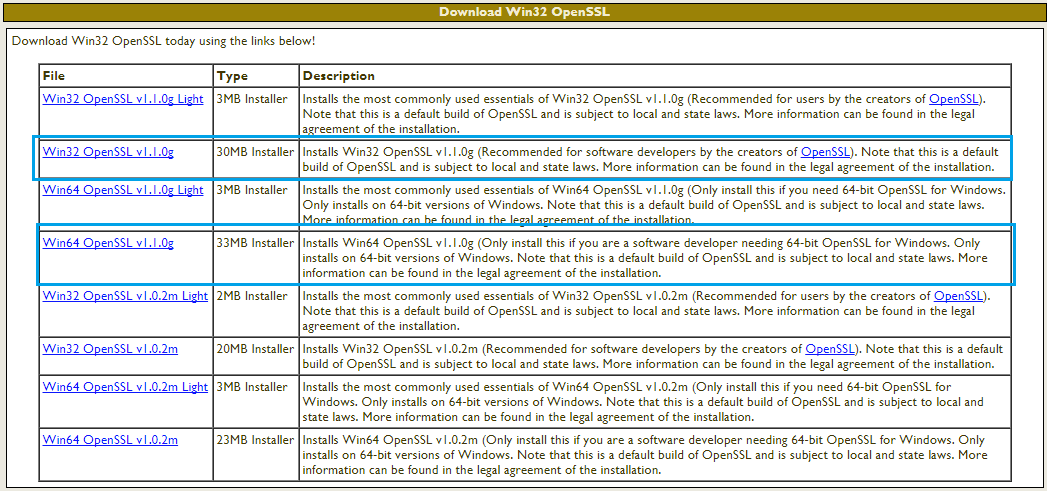
## Install OpenSSL

**OpenSSL** is an open-source implementation of the **SSL** and **TLS** protocols. The core library, written in the C programming language, implements the basic **cryptographic** functions and provides various utility functions.

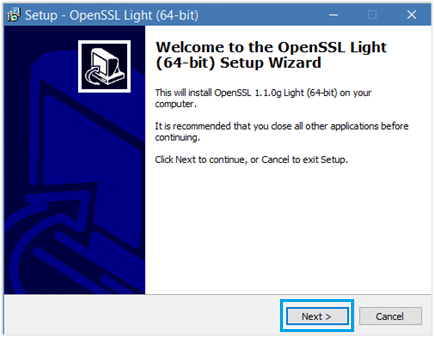
* 1. Open Command Prompt/Power Shell/Git Bash or some kind of command line interpreter
  2. Then write the command: “**openssl version”**, “**openssl version –a**” will show you more information
  3. If you have OpenSSL on your computer then you will see its version, eg. “**OpenSSL 1.0.2d 9 Jul 2015”**



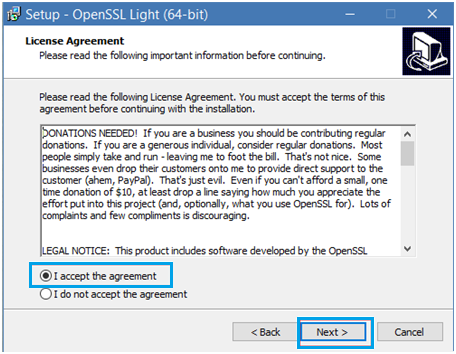
* 1. Then you can go to Exercise 2 and Generate Public and Private Key Pair
  2. If you do not have OpenSSL then you have to install it
  3. First download the OpenSSL file from: <https://slproweb.com/products/Win32OpenSSL.html> depends on your windows version



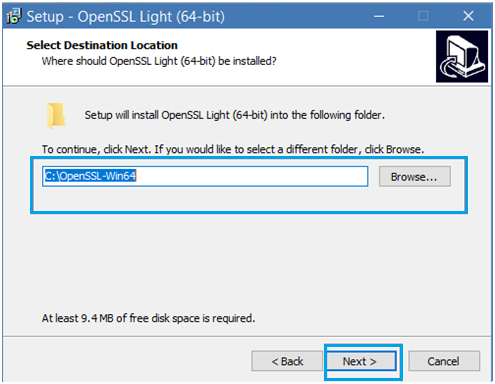
* 1. Then Open the .exe file. **Next**



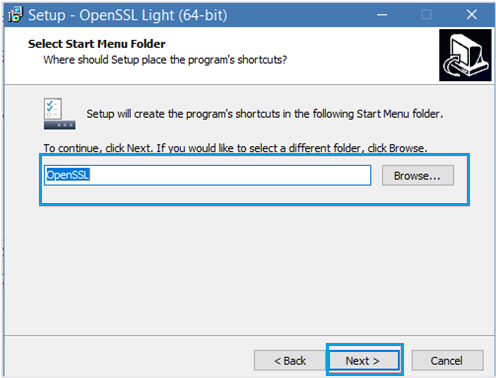
**Accept the agreement and Next**



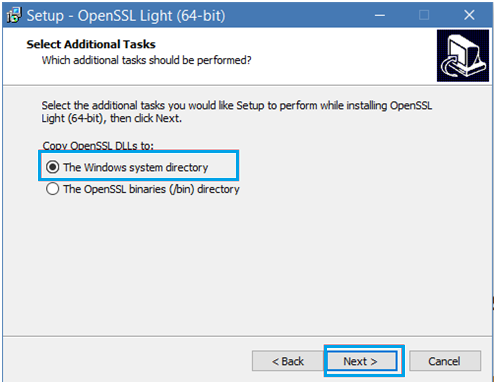
**Choose the default directory and Next**



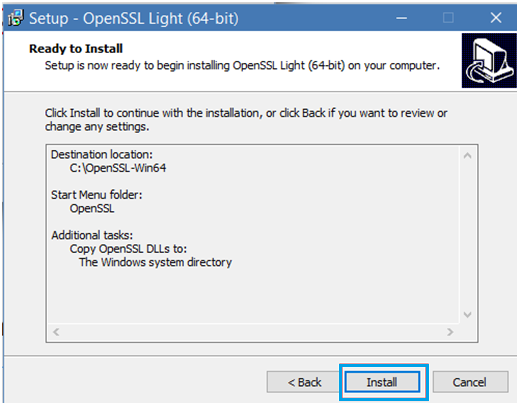
**Select the default folder and Next**



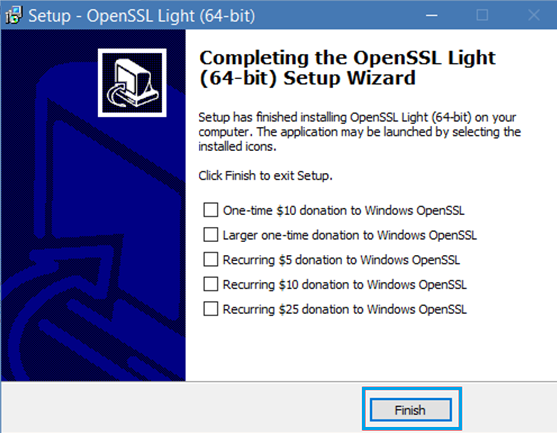
Choose to Copy OpenSSL DLLs to **“The Windows system directory” and Next**



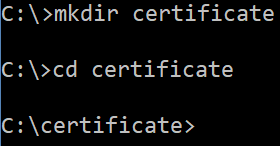
**Install**



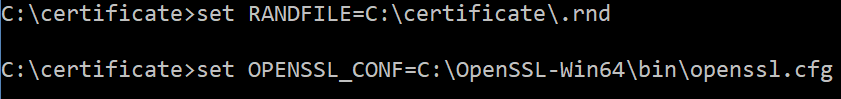
And **Finish**



* 1. Open command line and navigate to main folder **C:\**
  2. Make directory certificate : **mkdir certificate**
  3. Navigate to it: **cd certificate**



* 1. Now we need to set **2 environment variables**



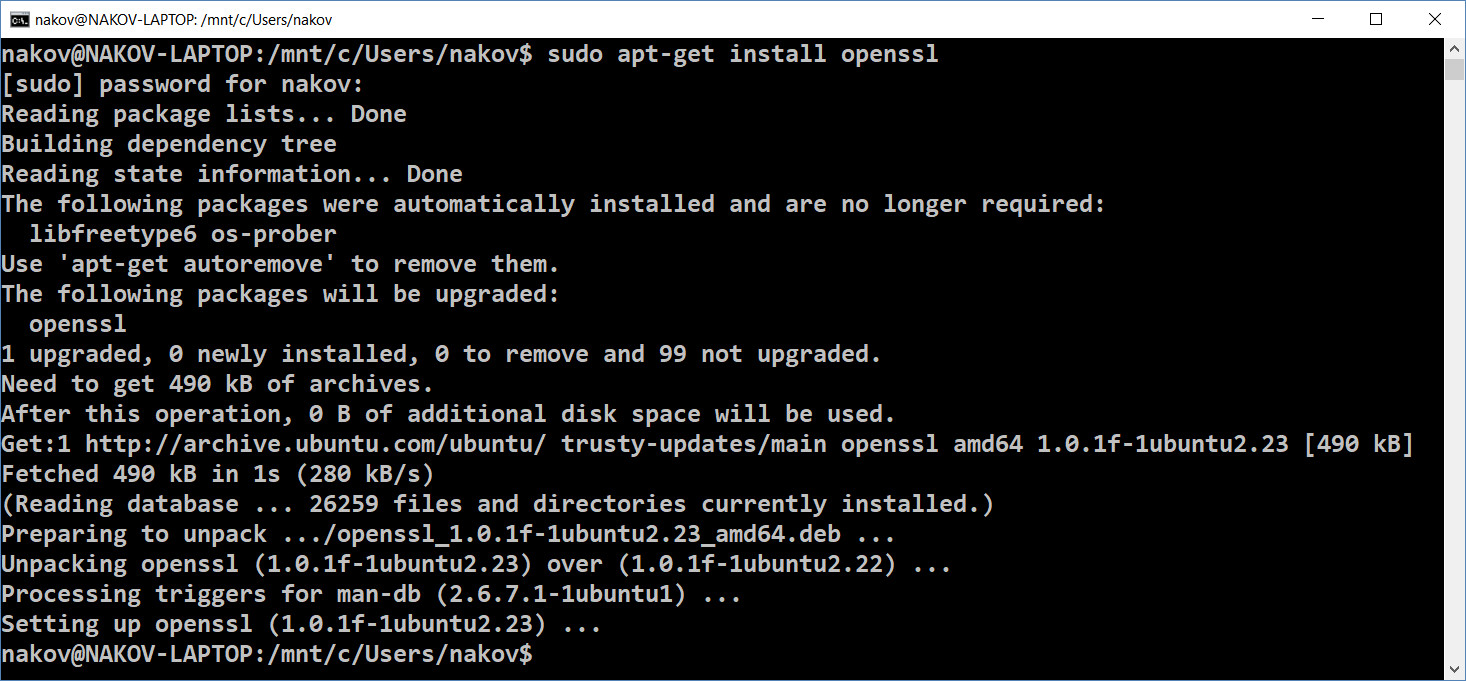
* 1. Commands:

**Set RANDFILE=C:\certificate\.rnd**

**Set OPENSSL\_CONF=C:\OpenSSL-Win64\bin\openssl.cfg**

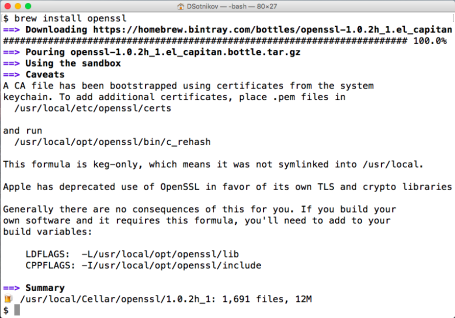
* 1. You can start it by typing: **C:\OpenSSL-Win64\bin\openssl.exe**

## Install OpenSSL on Linux

1. Run the command: **sudo apt-get install openssl**

## Install OpenSSL on Mac

* 1. Run the command: **brew install openssl**

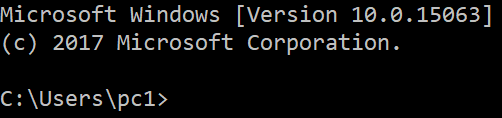


## Generate a Public / Private Key Pair using OpenSSL

Your task is to generate **2048 Bit RSA Key** using **OpenSSL**.

**RSA** (Rivest–Shamir–Adleman) is one of the first practical public-key cryptosystems and is widely used for secure data transmission. In such a cryptosystem, the **encryption key is public** and it is different from the **decryption key** which is kept secret (**private**).

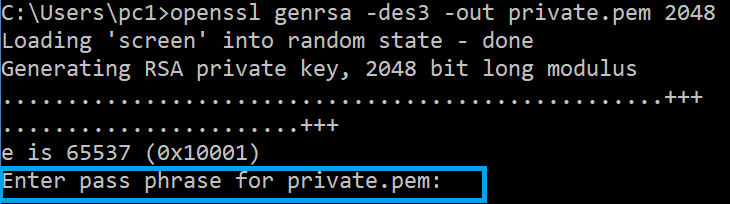
1. Open Command Prompt (cmd)/Power Shell/ Git Bash



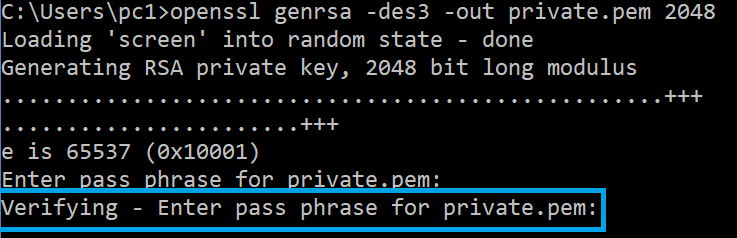
1. You can start OpenSSL with the command **C:\OpenSSL-Win64\bin\openssl.exe** or just type in the in front of the command **openssl**
2. You can generate a **public** and **private** RSA key pair with command:

sddsaasd.png

1. That generates a **2048-bit RSA key pair**, encrypts them with a password you provide, and writes them to a file.



1. After entering **pass** phrase you need to verify it:



Next extract the **public key file**. You will use this, for instance, on your web server to encrypt content so that it can only be read with the private key.

1. Export the RSA **public** key to a file:

sddsaasd.png

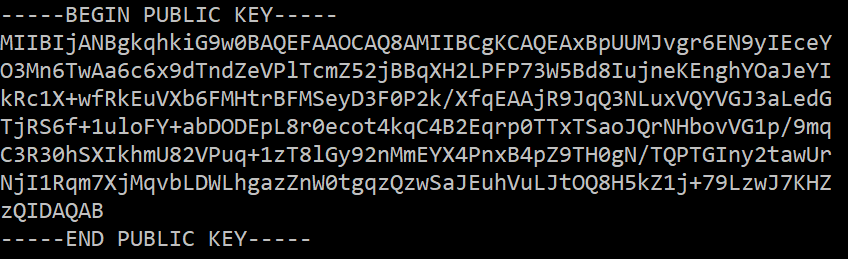
The ***–pubout*** flag is really important.

1. Open the **public.pem.** To check it from the command prompt use: **less public.pem** command:

sddsaasd.png

1. Ensure that it starts with a **-----BEGIN PUBLIC KEY-----**.

This is how you know this file is the **public key** of the pair and **not a private key**.



1. Export the RSA **private** key to a file (Not a good idea to export private key):sddsaasd.png
2. Open the **private\_unencrypted.pem.** To check it from the command prompt use: **less private\_unencrypted.pem** command:

sddsaasd.png

1. Ensure that it starts with a **-----BEGIN PRIVATE KEY-----**.

This is how you know this file is the **private key** of the pair and not a **public key**.



**DO NOT SHOW IT!**

**less private.pem** to verify that it starts with a **-----BEGIN RSA PRIVATE KEY-----**

**less public.pem** to verify that it starts with a **-----BEGIN RSA PUBLIC KEY-----**

Depending on the nature of the information you will protect, it’s important to keep the **private key backed up and secret**.

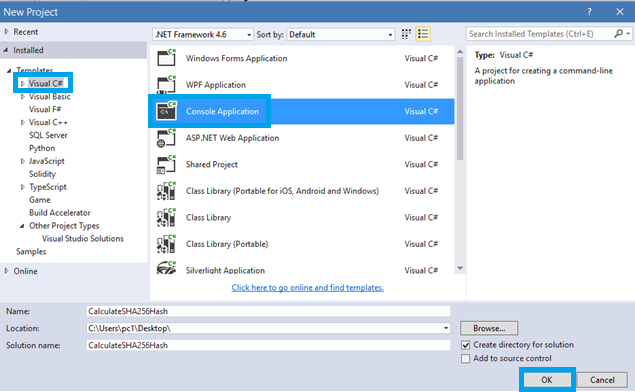
The **public key can be distributed** anywhere or embedded in your web application scripts.

Again, **backup your keys!**

## Calculate a SHA256 Hash using C#

Now we will learn how to hash a string by using **C#** programming language.

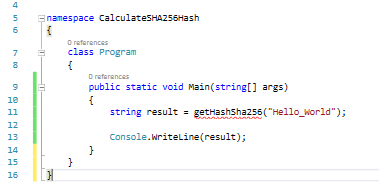
1. Firstly you will need **C# IDE** (Visual Studio).
2. Then Start the IDE and go to **File** -> **New** -> **Project** and choose “**Console Application**”



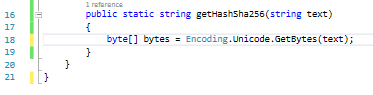
1. We have to include the following **usings**:

weq.png

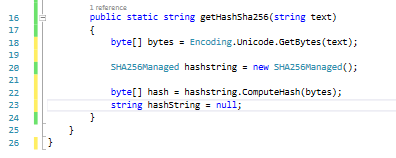
1. In the **main** method we declare the **string** variable “**result**” which is the **SHA256** **hash** of “**Hello\_World**” and then print it.



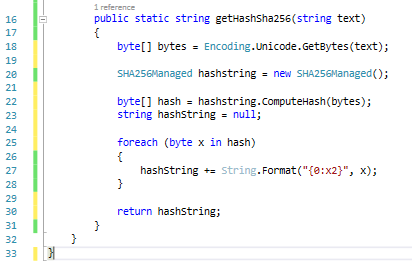
1. Then define the **getHashSha256** method which gets **string** and returns **string**. We need to **encode** the **text** in **byte array**.



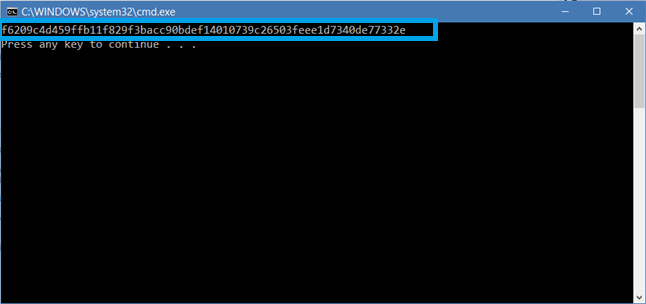
1. We should use the **SHA256Manager** from the **System.Security.Cryptography** library which will **compute** the hash of the **string** and return **byte array**. Then we define the **hashString** we will return at the end.



1. Finally in a **foreach loop**, which get one by one the **bytes** from the **byte array**, we concatenate the formatted string to **hashString**. **{0:x2}** means that the **byte** will be in **hexadecimal** format. Then return the **hashString**.



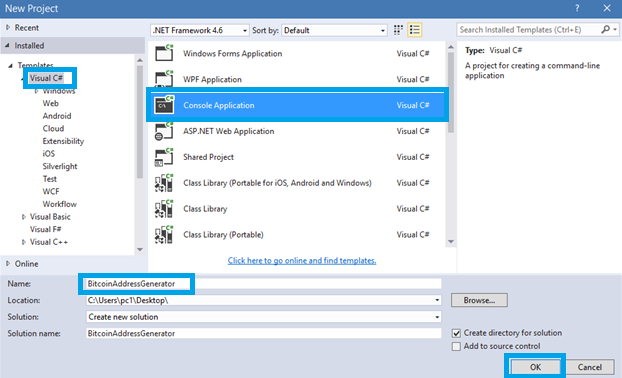
1. And the result should be **64 length alphanumeric string**:



The full source code is available here: <https://github.com/sMustafov/CalculateSHA256Hash>

## Bitcoin Address Generator (C# Edition)

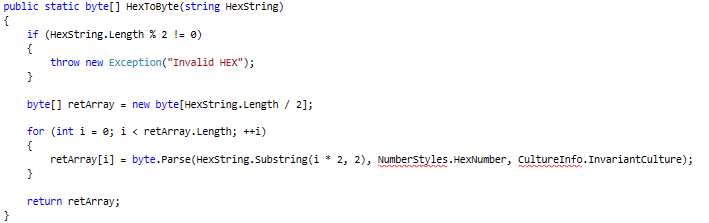
* 1. First you need to open Visual Studio 2017 or an IDE to write C# code.
  2. Then **File** -> **New** -> **Project** and select “**Console Application”** give name and **OK**



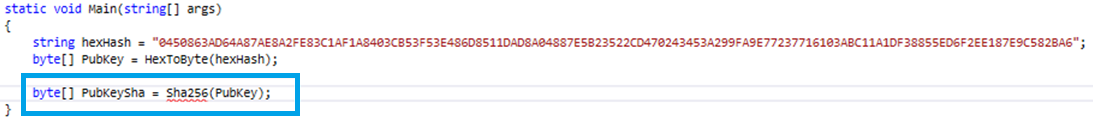
* 1. We give a default **hexHash** (alphanumeric) and calculate **Public Key** by the function HexToByte. Hex to Byte converts the Hexidecimal string to a byte array, it’s simple as taking the 2 characters and converting it into base 256.
  2. **Note:** The alphanumeric string is written by you, do not use the capital letters “O” and “I” because they are very similar looking to the numbers “0” and “1”.
  3. You can use the following string: **“0450863AD64A87AE8A2FE83C1AF1A8403CB53F53E486D8511DAD8A04887E5B23522CD470243453A299FA9E77237716103ABC11A1DF38855ED6F2EE187E9C582BA6”**

****

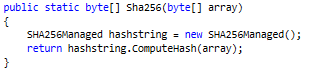
* 1. We have to implement the **HexToByte** method and we need to add **using System.Globalization** because of the **NumberStyles** and **CultureInfo**



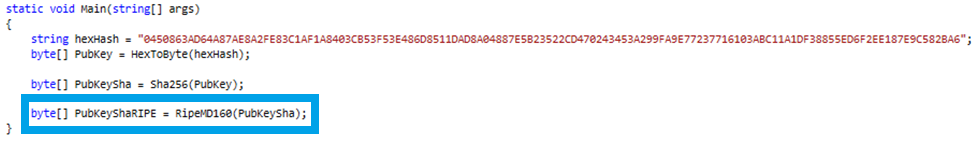
* 1. Then we need the **Sha256 Public Key**, so we have to implement the **Sha256** method.



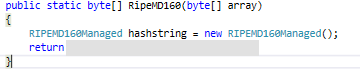
* 1. Sha256 uses Microsoft’s security cryptography include, and by default takes a byte array. We need **using System.Security.Cryptography;**



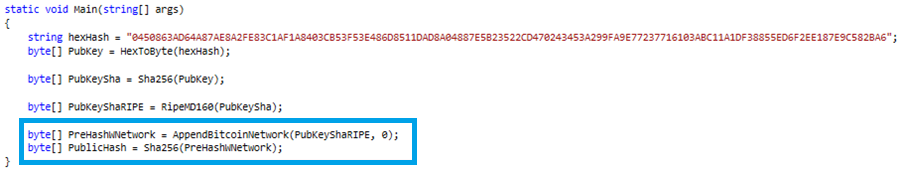
* 1. Then we have to implement the **RipeMD160** method.



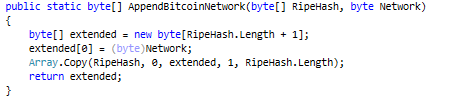
* 1. Again this function uses Microsoft’s security cryptography include and is pretty much identical to the **Sha256 function**.



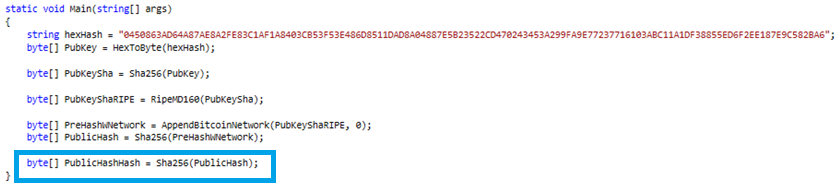
* 1. **AppendBitcoinNetwork** simply pre-appends a byte onto the beginning of an array of bytes and then hash the **PreHashWNetwork** with **Sha256**.



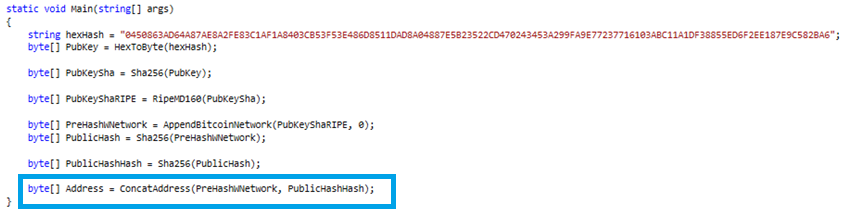
* 1. **AppendBitcoinNetwork** method:



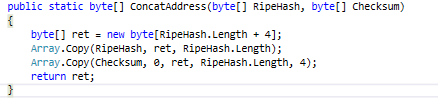
* 1. Then hash the **Public Hash**



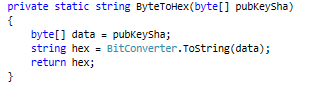
* 1. Finally we need **concat address** which appends the last 4 bytes of the hash onto the end of the RipeMD160 value.



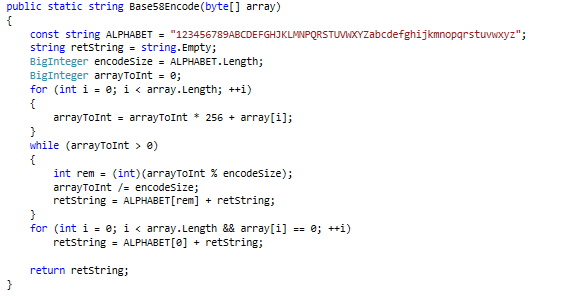
* 1. Contact address implementation:



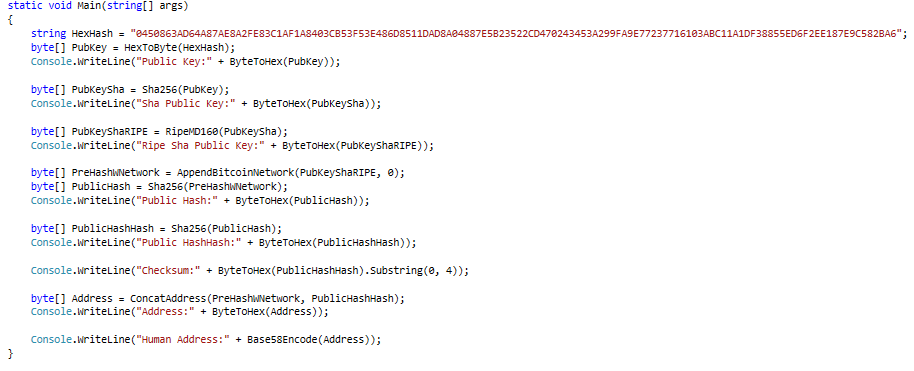
* 1. It is Console Application so if we want to see the result we have to print it on the console. We have to the method **ByteToHex** which **converts a byte into a hexadecimal string.**



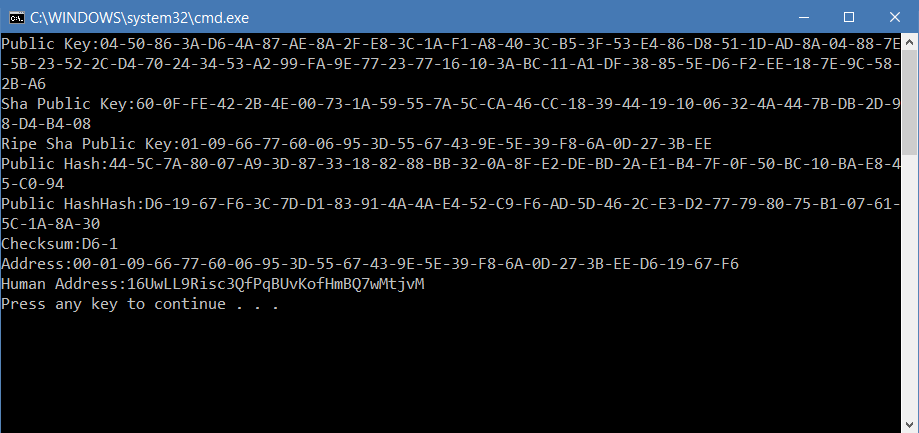
* 1. The last method we need is the **Base58Encode** to make **the Contact Address** human readable and we have to add Reference **System.Numerics** and then **using System.Numerics** to use BigInteger.
  2. **Note:** The alphanumeric string is written by you, do not use the capital letters “O” and “I” and the letter “l” the small letter of “L” and the number “0” because they are very similar looking to the numbers “0” and “1”.
  3. You can use the following string: **“123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz”**



* 1. Now we can add the print all the information on the console.



* 1. Now Run the Console Application (**CTRL + F5**) and the result should be the following:

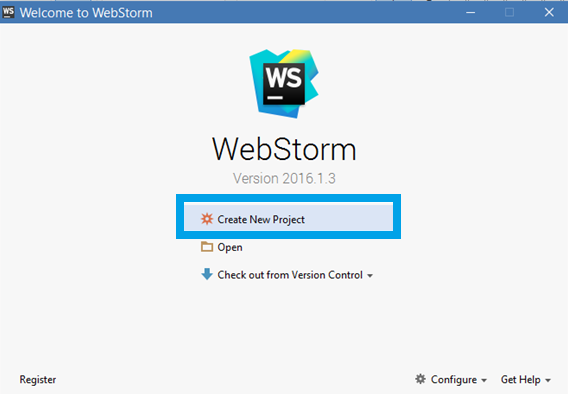


The full source code is available here: <https://github.com/sMustafov/BitcoinAddressGeneratorCSharpEdition>

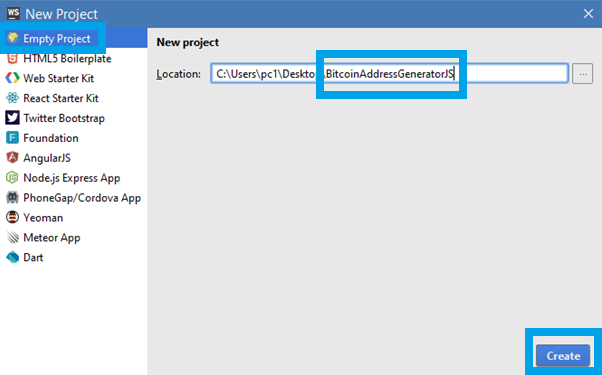
## Bitcoin Address Generator (JS Edition)

Now we will generate **bitcoin** address and transaction by using **JavaScript** programming language.

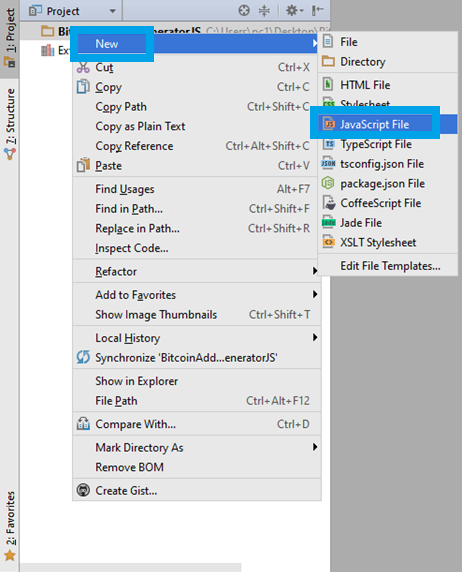
1. Firstly you will need **JS IDE** (**WebStorm**).
2. Then Start the IDE and choose “**Create New Project**”.



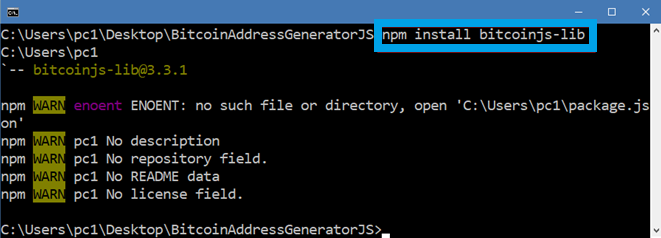
1. Choose “**Empty Project**”, give good name and “**Create**”.



1. Click Right button on project folder and choose **New** -> **JavaScript File.**



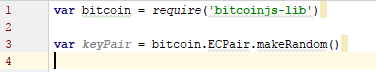
1. Open Command Line Interpreter go to the project folder and install **BitcoinJS library** there.



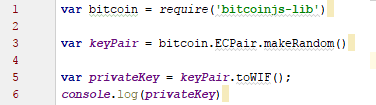
1. Require the **bitcoinjs-lib**.

555.png

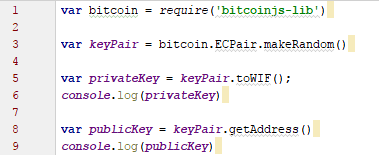
1. Create random key pair.



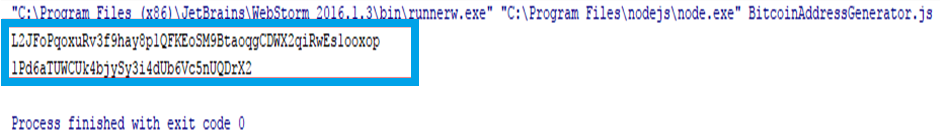
1. Create **private key** in WIF format and then print it.



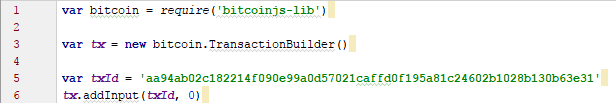
1. Then create **public key** and then print it.



1. The result should be something like this:



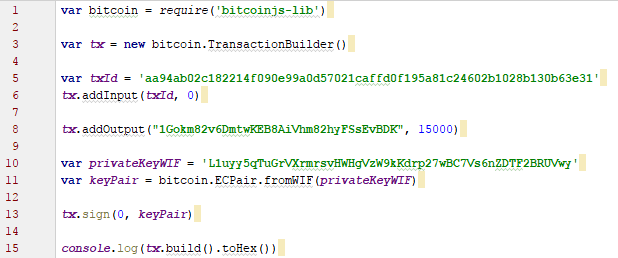
1. Now create new file and name it “**GenerateTransaction**”.
2. We need the **bitcoinjs-lib** again. Then create new **Bitcoin** **transaction**. Create custom transaction ID. Finally add the input (who is paying): previous transaction hash and index of the output to use.



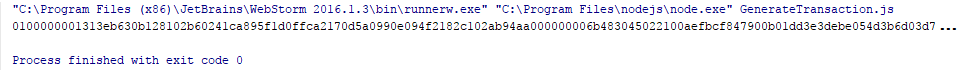
1. Then add the output (who to pay to): payee's address, amount in satoshis.



1. Now initialize a private key using WIF and then sign the first input with the new key. Finally build the transaction, convert it to hexadecimal and print it.



1. The result should look like:



The full source code is available here: <https://github.com/sMustafov/BitcoinAddressGeneratorJSEdition>

# What to Submit?

Create a **zip file** (e.g. your-name-cryptography-exercise.zip) holding your assets.

* Put in the file **screenshot** of **first** and **second** exercises.
  + **First exercise** – write “**openssl”** command on **cmd** and **screenshot** the result
  + **Second exercise** – **screenshot** the **public** and **private** keys.
* Put in the file also the **source codes** of **third** and **fourth** exercises (only the **.cs** files).

Submit your zip file as **homework** at the course Web site.