# Exercises: Sign and Verify Transaction in JavaScript

In this exercise, you shall write code on how to **sign** and **verify** transactions using the **secp256k1-based ECDSA** cryptography in JavaScript. You will use the following JS libraries from NPM:

* **Crypto-JS** – a JavaScript crypto-library
* **Elliptic –** a fast elliptic-curve cryptography implementation for **secp256k1**

In short, your goal is to write code to **sign transactions**, given as **JSON**:



# 1. Implement and Test the Code

1. We should import some **libraries**: **crypto-js**, **elliptic**

First, initialize an empty **package.json**:

|  |
| --- |
| npm init -y |

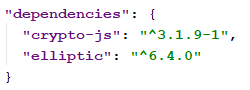
Then, install **cryptojs**:

|  |
| --- |
| npm install --save crypto-js |

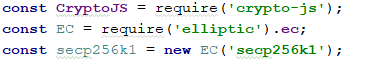
Install **elliptic**:

|  |
| --- |
| npm install --save elliptic |

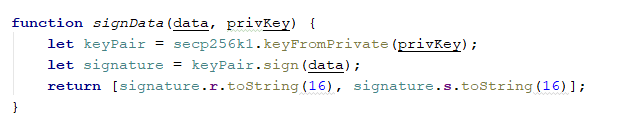
The dependencies in **package.json**:



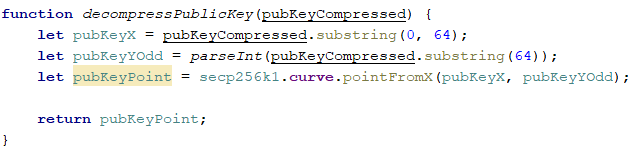
Create a JavaScript file and write the following constants:



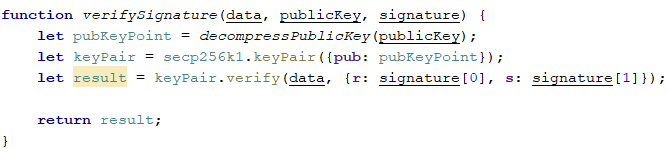
1. To **sign** any data we need data and a private key. Build a key pair with **secpk256k1** using **keyFromPrivate** with which we will **sign** the data. Return the signature's parameters in hex:



1. In order to verify a signature we will need the data, **public key** of the signer and the signature itself. To create a key pair we need to **decompress** the public key:

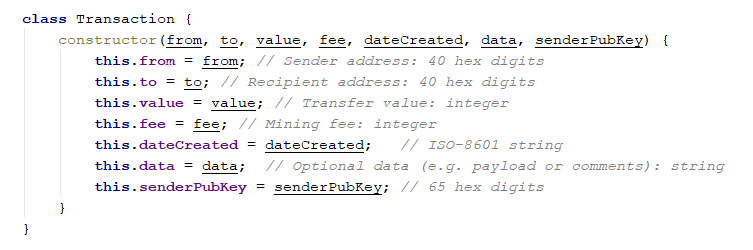


1. Verifying signature needs to create a key pair from the public key and **verify** the **data** with the **signature**:

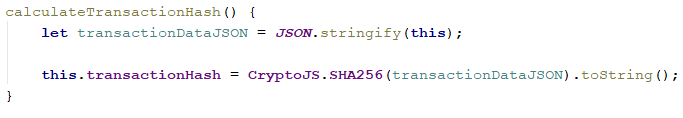


Now it is time to use these methods but for transactions:

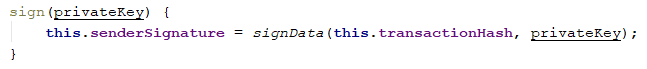
1. Create a class **Transaction** that stores transaction information and has methods to sign and verify it. The class will take several parameters:
   * Sender address: 40 hex digits
   * Recipient address: 40 hex digits
   * Transfer value: integer
   * Mining fee: integer
   * Date created: ISO-8601 string
   * Data: Optional data (e.g. payload or comments): string
   * Sender public key: 65 hex digits



1. Before signing a transaction, we must first calculate **SHA256** of the JSON transaction and then sign it.

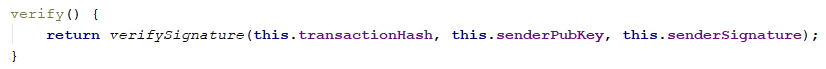


1. **Sign** method that takes private key, calls **signData** method with the transaction hash and saves the signed signature:



1. **Verify** method that calls **verifySignature** with the **transaction hash** as data, sender **public key** and **signature**.

The method returns whether the signature is valid or invalid.



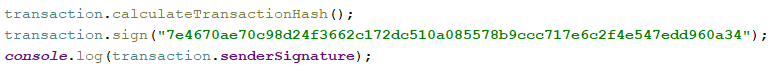
1. Create **transaction** with the following **parameters**:

|  |
| --- |
| **let** transaction = **new** Transaction(  **"c3293572dbe6ebc60de4a20ed0e21446cae66b17"**,  **"f51362b7351ef62253a227a77751ad9b2302f911"**,  25000,  10,  **"2018-02-10T17:53:48.972Z"**,  **"Send to Bob"**,  **"c74a8458cd7a7e48f4b7ae6f4ae9f56c5c88c0f03e7c59cb4132b9d9d1600bba1"** ); |

1. Finally, **test** the code. **The sender address must always match the sender public key and private signer**.

First, calculate the hash of the transaction, **sign** the hash with

|  |
| --- |
| 7e4670ae70c98d24f3662c172dc510a085578b9ccc717e6c2f4e547edd960a34 |





1. **Verify** the signature:





# What to Submit?

Create a **ZIP file** (e.g. username-sign-verify-js-exercise.zip) holding your JavaScript file and the package.json.