# Exercises: Sign / Verify Ethereum Messages

In this exercise you shall write code to **sign** and **verify** **messages**, using the secp256k1-based ECDSA cryptography and signatures in the **Ethereum** format (130 hex digits). You will read and produce **JSON documents**, in the same format, that is readable from the **MyEtherWallet** app.

## Sign Messages in Ethereum Style

Write a program toreada **256-bit private key** and a **text message** and digitally **sign the message**. Produce **JSON** document, in the same format, that is readable from the **MyEtherWallet** app. See the examples below:

|  |
| --- |
| **Input (Private Key + Message)** |
| 97ddae0f3a25b92268175400149d65d6887b9cefaf28ea2c078e05cdc15a3c0a  Message for signing |
| **Output (JSON)** |
| {  "address": "**0xa44f70834a711f0df388ab016465f2eeb255ded0**",  "msg": "**Message for signing**",  "sig": "**0x6f0156091cbe912f2d5d1215cc3cd81c0963c8839b93af60e0921b61a19c54300c71006dd93f3508c432daca21db0095f4b16542782b7986f48a5d0ae3c583d401**",  "version": "**1**"  } |

**Input**: two text lines

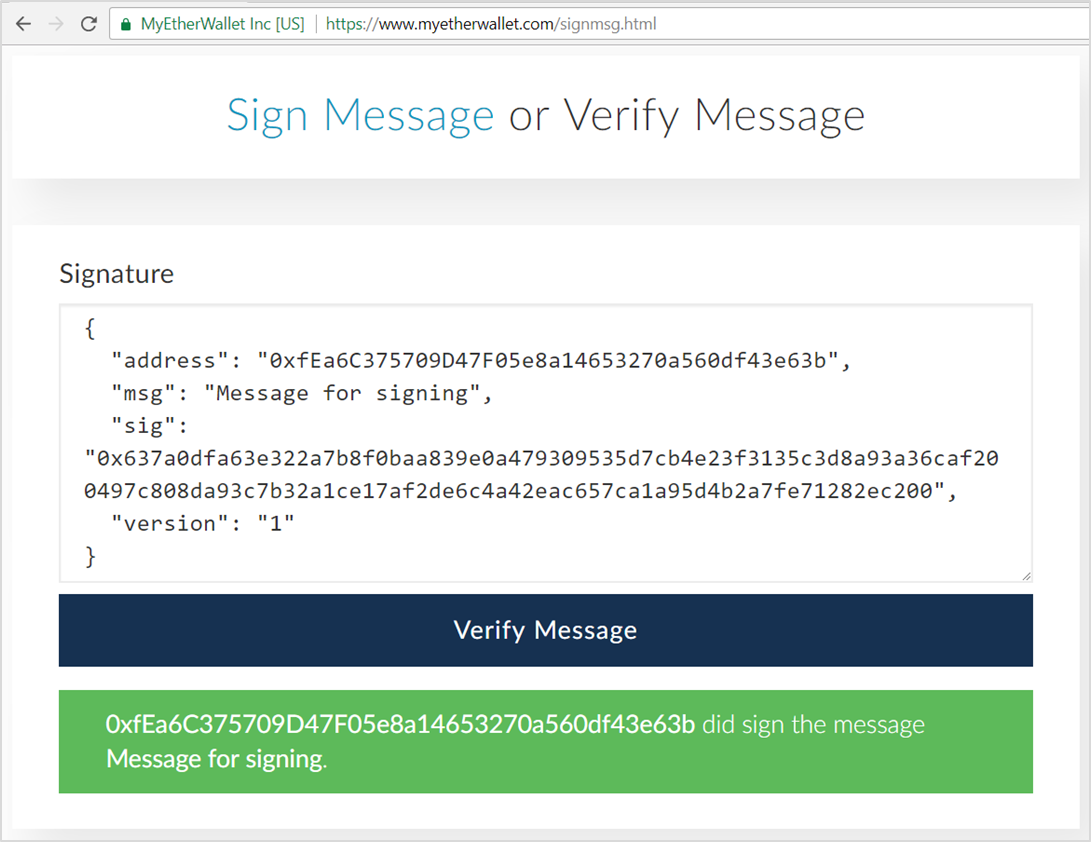
* 256-bit **private key**, encoded in **hex** (without the **0x** prefix)
* **text message** for signing (encoded in UTF8)

**Output**: JSON document holding the signed message, along with the signer address:

* **address**: holds the **Ethereum address**, derived from public key, corresponding to the given private key (the last 40 hex digits of the keccak256 hash of the uncompressed public key), prefixed with **0x**
* **msg**: holds the signed message (in UTF8 encoding)
* **sig**: holds the Ethereum-style ECDSA signature [**v**, **r**, **s**] – 130 hex digits, prefixed with **0x**
  + **r** – 64 hex digits
  + **s** – 64 hex digits
  + **v** – 2 hex digits (00 or 01)
* **version**: holds always "**1**"

Use programming language and cryptographic libraries of choice.

You can verify the signed message JSON at: <https://www.myetherwallet.com/signmsg.html>.



## Verify Signed Message in Ethereum JSON Format

Write a program to **verify signed message**, given as JSON document, in the exactly same format, like in the previous exercise. Example:

|  |
| --- |
| **Input (Signed Message JSON)** |
| {  "address": "**0xa44f70834a711f0df388ab016465f2eeb255ded0**",  "msg": "**Message for signing**",  "sig": "**0x6f0156091cbe912f2d5d1215cc3cd81c0963c8839b93af60e0921b61a19c54300c71006dd93f3508c432daca21db0095f4b16542782b7986f48a5d0ae3c583d401**",  "version": "**1**"  } |
| **Output (Valid / Invalid)** |
| Valid |

Another example, holding a tampered message:

|  |
| --- |
| **Input (Signed Message JSON)** |
| {  "address": "**0xa44f70834a711f0df388ab016465f2eeb255ded0**",  "msg": "**Tampered message**",  "sig": "**0x6f0156091cbe912f2d5d1215cc3cd81c0963c8839b93af60e0921b61a19c54300c71006dd93f3508c432daca21db0095f4b16542782b7986f48a5d0ae3c583d401**",  "version": "**1**"  } |
| **Output (Valid / Invalid)** |
| Invalid |

Use programming language and cryptographic libraries of choice.

# What to Submit?

Create a **ZIP file** (e.g. your-name-ethereum-sign-verify-exercise.zip) holding your source code for all problems. Submit your ZIP file as **homework** at the course Web site.