# Do a raw signature of a Bitcoin transaction as a wallet would do

Daniel Arturo Casals Amat, Estudiante de Magister en Universidad Técnica Federico Santamaría. Magister en Ciencias de Ingeniería Informática

Felipe Beroiza, Estudiante de Doctorado en Universidad Técnica Federico Santamaría. Doctorado en Ciencias de Ingeniería Informática.

**Resumen**—El proceso de generar una transacción rela para ser distribuida y agragada al blockchain de la red de bitcoin consta de 3 pasos. En este reporte se desglosan los detalles de implementación para generar una transacción de bitcoin firmada y lista para ser distribuida entre la red de p2p de nodos de bitcoin. Como resultado se obtiene scripts para generar transacciones y descerializar estas para mostrar el proceso.

**Index Terms**—Bitcoin, Blockchain, transactions.

## 1. Introduction

Las transacciones son la parte más importante del sistema bitcoin. Todo lo demás en bitcoin está diseñado para garantizar que las transacciones puedan crearse, propagarse en la red, validarse y, finalmente, agregarse al libro mayor global de transacciones (la cadena de bloques). Las transacciones son estructuras de datos que codifican la transferencia de valor entre los participantes en el sistema de bitcoin. Cada transacción es una entrada pública en la cadena de bloques de bitcoin, el libro contable global de doble entrada.[1]

El proceso de generar una transacción firmada y lista para ser distribuida en una red p2p de nodos que utilizan el protocolo de bitcoin consta de varios pasos. Este se encuentra lleno de detalles de implementación que deben ser tenidos en cuenta para que la transacción sea procesada e incluida en un bloque que pase a formar parte del blockchain.

El alcance de este trabajo solo cubre los detalles para generar la transacción firmada válida desde el punto de vista estructural semántica. No se validan otros detalles como: la existencia de el/los UTXO a gastar, la correspondencia entre el monto del/los UTXO con respecto a los montos de salidas, entre otros detalles que pueden terminar invalidando la inclusión de la transacción en el blockchain. Además se provee un script que realiza la descerialización de una transacción firmada siguiendo la estructura planteada en [2]

Mayo, 2019

### 1.1. Objectives

- Do a raw signature of a bitcoin as a wallet would do.
- Explain what part of the signature corresponds to what.

### 1.2. Pasos para generar una transacción firmada

The steps to create a real transaction at a high level can be the following:

- Create a Raw Transaction.
- Take a Raw Transaction from first step and sign with the private key.
- Use the signed key from second step to create a real raw transaction.

• *A report for the Cryptocurrencies and Smart Contracts course offered by Pontificia Universidad Católica de Chile*
*E-mail: daniel.casals@sansano.usm.cl*

As a result, a string representing the signed transaction must be obtained. In the following sections we will break down each of these steps in detail.

#### 1.2.1. Create a Raw Transaction

The next transaction to be created complies with the instructions of [3]. This spends a previous pay-to-pubkey output paying to a new pay-to-pubkey-hash (P2PKH) output. The previous dimension is because the Script language that will be used later comprises several commands more than those detailed here. The structure used in this project only comprises generating a transaction type P2PKH.

We start creating a new raw transaction:

1. **Add four-byte version field.**
2. **Add n bits that represent The Number of Inputs**. The numbers of bytes used varies in orther to use the least number of bytes needed. By sinplicity we use 1 byte for this inplementation that allow represent 252 inputs. The full specification that sould be applied you can consult in [4]
3. **Add txIn**. The bytes to use in that step varies in order to represent N inputs defined in the before item. In the next epigraf we describe that in detail.
4. **Add N bits that represent The Number of Outputs**. The numbers of bytes used varies in orther to use the least number of bytes needed. By sinplicity we use 1 byte for this inplementation that allow represent 252 inputs. The full specification that sould be applied you can consult in [4]
5. **Add txOut**. The bytes to use in that step varies in order to represent N inputs defined in the before item. In the next epigraf we describe that in detail.
6. **Add 4 bytes of lock_time**. For simplicity we use default value proposed for bitcoint core(0xffffffff). But a complete explanation of this field could be finded in [5]

The raw_tx class written in python defines the previously structured one. It could be found in [6]. An implementation detail is that all values sould be written in little-endian format.[7]. The flip_byte_order is used to invert the order of bytes secuences for that reason.

#### 1.2.2. Input Transactions structure

The txIn mencioned in the before empigraf refers to transactions that contain UTXO outputs that are going to be spent by the current transaction. It is necessary to remember that the outputs of the transactions that have not been spent become the inputs for future transactions. The sum of the set of bitcoins specified in the UTXOs associated with a specific key represents the balance available to the owner of the key.

An input in a transaction which contains three fields: an outpoint, a signature script, and a sequence number. The outpoint references a previous output and the signature script allows spending it.[8]

1. **Add the outpoint.**. The data structure used to refer to a particular transaction output, consisting of a 32-byte TXID and a 4-byte output index number (vout). Each of this should be in little-endian.
2. **Add length of the script**. The bytes to use in that step varies in order to represent the length of the script specified in the next item.[4]
3. **Add signature script**. Data generated by a spender which is almost always used as variables to satisfy a pubkey script. Signature Scripts are called tx_in["script"] in code. As the type of transaction used in this work is a P2PKH The script implemented follow the structure: **OP_DUP OP_HASH160 PUBICKEY_HASH OP_EQUALVERIFY OP_CHECKSIG**. Each Script language command used is represented in code as the exadecimal code that is defined in [9]
4. **Add 4 bytes of secuence field**. For simplicity we use default value proposed for bitcoint core(0xffffffff). But a complete explanation of this field could be finded in [5]

The PUBLICKEY_HASH puede ser obtenido de dos formas: Si se realiza el proceso utilizando la llave pública descomprimida(resultante de generar una llave privada de 32 bytes y concatenar 0x04 + llave pública descomprimida de 64 bytes) el proceso es el siguiente:

1. Crear llave privada de 32 bytes.

2. Obtener la lleve publica de 64 bytes y concatenarle al inicio 0x04 para obtener la llave pública que utiliza bitcoin. El prefijo 0x04 se utiliza para identificar que la llave se encuentra descomprimida. Ver [10]
3. Hash utilizando SHA-256 sobre prefijo + publicKey del paso anterior.
4. Hash utlizando RIPEMD160 sobre el resultado en hexadecimal del paso anterior. Ver [11]

Si el proceso se realiza desde la BitcoinAddress:

1. Decodificar utilizando Base58 Decoder la BitcoinAddress, este paso debe entregar 25 bytes que corresponden al formato PREFIX + CompressedPrublicKey + CHECKSUM.
2. Eliminar el primer byte correspondiente al PREFIX, en este caso como se está utilizando P2PKH debe corresponder al byte 0x00 que representa a una dirección de bitcoin.[12]
3. Eliminar los últimos 4 bytes correspondientes al CHECKSUM.

The raw input called txIn is the concatenation of this fields( all in little-endian format) for each UTXO to spent.

#### *1.2.3. Transactions Output structure*

TxOut is a sequence of output structures defined below:

1. **Add the value field**. In 8 bytes it represents the amount in satochies to be transferred in the output. A satochie represents 1 BTC is represented as 1000000 Satochies.
2. **Add the output script**. As in the input it is a string with the structure: **OP_DUP OP_HASH160 PUBLICKEY_HASH OP_EQUALVERIFY OP_CHECKSIG**. Defines the conditions which must be satisfied to spend this output.

The raw transaction must be this estructure:

Listing 1. First Raw Transaction

```
raw_tx_string = (
      rtx.version
      + rtx.tx_in_count
      + rtx.tx_in["txouthash"]
      + rtx.tx_in["tx_out_index"]
      + rtx.tx_in["scrip_bytes"]
      + rtx.tx_in["script"]
      + rtx.tx_in["sequence"]
      + rtx.tx_out_count
      + rtx.tx_out1["value"]
      + rtx.tx_out1["
         pk_script_bytes"]
      + rtx.tx_out1["pk_script"]
      + rtx.tx_out2["value"]
      + rtx.tx_out2["
         pk_script_bytes"]
      + rtx.tx_out2["pk_script"]
      + rtx.lock_time
      + struct.pack("<L", 1) #
         hashCode
)
```

The last field is not described in the previous documentation, but must be included in order to create the raw transaction. Corresponds to the hash code, the same one that identifies the type of script is a P2PKH. This is represented with 4 bytes equally in little-endian format.

### 1.3. Signature and sign the raw transaction

The signature process consists of the following steps:

1. **Perform double hashing using SHA-256 over the key the raw _transaction**
2. **Use ECDSA to generate the signature**. The Elliptic Curve Digital Signature Algorithm (ECDSA) with the secp256k1 used by Bitcoin uses the private key of the author of the transaction as seed. The signature of the result of the previous step is obtained
3. **Obtain the unlocked public key**. As previously mentioned, the public key must start with the concatenation of 0x04 + public key.
4. **Get the length of the public key in hexadecimal**
5. **Create the structured string**. The structured string must be returned in this step as SIGNATURE + 0x01 in 1 Byte + Length of PublicKey + PublicKey.

Un detalle de implementación acá es que tanto el Signature como el PublicKey son los únicos elementos que no están escritos in little-endian.

#### 1.3.1. Getting the real transaction output.

The last step is to generate the result transaction output its very símilar to the first transaction output:

Listing 2. Create real Transaction

```
real_tx = (
      rtx.version
      + rtx.tx_in_count
      + rtx.tx_in["txouthash"]
      + rtx.tx_in["tx_out_index"]
      + struct.pack("<B", len(
         sigscript) + 1)
      + struct.pack("<B", len(
         signature) + 1)
      + sigscript
      + rtx.tx_in["sequence"]
      + rtx.tx_out_count
      + rtx.tx_out1["value"]
      + rtx.tx_out1["
         pk_script_bytes"]
      + rtx.tx_out1["pk_script"]
      + rtx.tx_out2["value"]
      + rtx.tx_out2["
         pk_script_bytes"]
      + rtx.tx_out2["pk_script"]
      + rtx.lock_time
      # hashCode is deleted
)
```

If we compare the previous code with the first one shown in section 1.2.1 we can notice subtle differences. The script and script lenth are replaces by the fields length of sigscript, length of signature an the signscript generated in the previous section. It has also been eliminated the 'hash code' field.

### 1.4. Descerialization

The tester.py script implement a deserialization process for a P2PKH script signed. The result for the generation of a real transaction described before could be tested here.

### 1.5. Recomendations

Some elements such as the representation of the integers in CompacSize have been simplified for the use of 1 byte for simplicity. It is proposed to extend the behavior so that it covers more than 253 transactions of inputs and outputs.

## REFERENCIAS


[1] "Mastering Bitcoin 2nd Edition: Programming the Open Blockchain - bitcoinbook/bitcoinbook," Mastering Bitcoin. [Online]. Available: https://github.com/bitcoinbook/bitcoinbook

[2] Bitcoin Developer Reference - Bitcoin. [Online]. Available: https://bitcoin.org/en/developer-reference#raw-transaction-format

[3] Developer Examples - Bitcoin. [Online]. Available: https://bitcoin.org/en/developer-examples#simple-raw-transaction

[4] Bitcoin Developer Reference - CompactSize Unsigned Integers. [Online]. Available: https://bitcoin.org/en/developer-reference#compactsize-unsigned-integers

[5] Transactions guide - bitcoin locktime. [Online]. Available: https://bitcoin.org/en/transactions-guide#locktime-and-sequence-number

[6] D. A. C. Amat, "Dacasals/bitcoin_project on GitHub." [Online]. Available: https://github.com/dacasals/bitcoin_project

[7] Little-Endian. [Online]. Available: https://learnmeabitcoin.com/glossary/little-endian

[8] Input, Transaction Input, TxIn - Bitcoin Glossary. [Online]. Available: https://bitcoin.org/en/glossary/input

[9] Script - Bitcoin Wiki Script language. [Online]. Available: https://en.bitcoin.it/wiki/Script

[10] 4. Keys, Addresses - Mastering Bitcoin, 2nd Edition [Book]. [Online]. Available: https://www.oreilly.com/library/view/mastering-bitcoin-2nd/9781491954379/ch04.html

[11] Hash160(Public Key). [Online]. Available: https://learnmeabitcoin.com/glossary/public-key-hash160

[12] BitcoinAddress. [Online]. Available: https://learnmeabitcoin.com/glossary/address