Transacciones en Bitcoin

Javier Domínguez Gómez
jdg@member.fsf.org
Fingerprint: 94AD 19F4 9005 EEB2 3384 C20F 5BDC C668 D664 8E2B

v0.1.05 - Septiembre 2019

Índice

1. Introducción

1.1. Formatos en el texto

A lo largo de este documento el lector encontrará algunas palabras o bloques de texto con diferentes formatos o tipografía. Se mostrarán en letra *cursiva* las palabras en inglés o que hagan referencia a algún término técnico. Resaltadas o en **negrita** aquellas palabras que describen una propiedad, una variable o un nombre de fichero, y finalmente con tipografía `Courier` aquellos datos que representen un valor hexadecimal o base 16, o fragmentos de código fuente en diferentes lenguajes de programación.

1.2. Objetivo

Este documento presenta en detalle el la estructura de datos, composición, funcionamiento y tipos de las transacciones en el protocolo *Bitcoin*[1], además el lector podrá encontrar información acerca del lenguaje *Bitcoin Script* que se emplea para el desarrollo e inclusión de programas llamados *scripts* en las transacciones. Puesto que las transacciones son el elemento base donde ubicar los *scripts* es necesario realizar un análisis forense a bajo nivel sobre cada una de las partes de las que se componen dentro del criptosistema definido por el protocolo *Bitcoin*. Todos los datos utilizados en los ejemplos, así como en los cálculos, son datos reales pertenecientes al bloque número #286819 de la cadena de bloques de *Bitcoin* en la red principal *mainnet*. No existe ninguna razón especial en la elección de este bloque, es uno al azar. El lector puede consultar los datos de este o cualquier otro bloque, bien en la cadena de bloques descargada en su computadora local o bien en un *explorador de bloques online*[2], y realizar los mismos cálculos en todos los casos, obviamente obteniendo resultados diferentes.

1.3. A quién va dirigido

El documento va dirigido a principalmente a lectores que quieren desarrollar programas en lenguaje *Bitcoin Script* para posteriormente utilizarlos como *Smart Contracts* en la red del protocolo *Bitcoin*. También va dirigido a lectores familiarizados con la criptología en cualquiera de sus dos ramas: la criptografía y el criptoanálisis, o a estudiantes de ciencias de la computación o matemáticas, en general a lectores que todavía no se han adentrado en el campo de del análisis forense sobre la información que contienelas transacciones de la cadena de bloques de *Bitcoin*, pero que quieren comprender su funcionamiento al más bajo nivel, a nivel de *bits*. El texto trata de facilitar al lector la información sobre los métodos empleados para poder identificar las transacciones dentro de un bloque de datos de *Bitcoin*. Para ello será necesario tener de antemano unos conocimientos mínimos sobre conversión de datos en diferentes bases como decimal y hexadecimal.

[1] https://bitcoin.org/bitcoin.pdf
[2] https://www.blockchain.com/en/btc/block-height/286819

2. Las transacciones en los bloques

Si se visualizan los datos que hay dentro de un bloque en la cadena de bloques de Bitcoin se encontrarán varios segmentos difícilmente identificables a simple vista. A continuación se detallan, siguiendo con el ejemplo del bloque #286819, y en condiciones normales, es decir, siempre y cuando a la cadena de bloques descargada no se le haya aplicado algún tipo de *prune*[3] o podado, los datos se encuentran en el archivo **blk00116.dat**. El contenido de todos estos archivos con extensión *.dat* comienza con los datos de un bloque, pero los datos del bloque que se quiere analizar no tienen por qué encontrarse en esa posición, es bastante probable encontrarlos muchos *bytes* más adelante o en un *offset* mucho mayor desde el inicio del archivo. Es el caso del bloque #286819, los datos comienzan a partir del *byte* número 93725126, en el *offset* número `059621c6`. En el siguiente ejemplo se ejecuta el programa **hexdump** con la opción **-C** para obtener una salida *standard* por pantalla, también con la opción **-s** seguido del número de *bytes* desde el que se quiere empezar a visualizar datos, y la opción **-n** para indicar el número total de *bytes* que se quieren mostrar, en este caso con 288 *bytes* es suficiente.

```
~/.bitcoin/blocks/$ hexdump -C -s 93725126 -n 288 blk00116.dat
059621c6 f9 be b4 d9 bd 53 02 00 02 00 00 00 17 97 5b 97 |.....S........[.|
059621d6 c1 8e d1 f7 e2 55 ad f2 97 59 9b 55 33 0e da b8 |.....U...Y.U3...|
059621e6 78 03 c8 17 01 00 00 00 00 00 00 00 8a 97 29 5a |x.............)Z|
059621f6 27 47 b4 f1 a0 b3 94 8d f3 99 03 44 c0 e1 9f a6 |'G.........D....|
05962206 b2 b9 2b 3a 19 c8 e6 ba dc 14 17 87 35 8b 05 53 |..+:........5..S|
05962216 53 5f 01 19 48 75 08 33 63 01 00 00 00 01 00 00 |S_..Hu.3c.......|
05962226 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
05962236 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff |................|
05962246 ff ff 60 03 63 60 04 06 2f 50 32 53 48 2f 04 35 |..`.c`../P2SH/.5|
05962256 8b 05 53 08 44 04 f2 53 00 00 17 e4 46 52 2c fa |..S.D..S....FR,.|
05962266 be 6d 6d 69 06 88 fb 88 6c 0d f0 c8 7c bc 7e a4 |.mmi....l...|. .|
05962276 f7 f1 b5 c0 05 0b d0 ac 37 51 cf c9 97 d9 d6 97 |........7Q......|
05962286 13 28 de 04 00 00 00 00 00 00 00 48 61 70 70 79 |.(.........Happy|
05962296 20 4e 59 21 20 59 6f 75 72 73 20 47 48 61 73 68 | NY! Yours GHash|
059622a6 2e 49 4f 00 00 00 00 01 cb 81 31 95 00 00 00 00 |.IO.......1.....|
059622b6 19 76 a9 14 80 ad 90 d4 03 58 1f a3 bf 46 08 6a |.v.......X...F.j|
059622c6 91 b2 d9 d4 12 5d b6 c1 88 ac 00 00 00 00 01 00 |.....]..........|
059622d6 00 00 01 7d 67 7c de 17 3f 8c bf 43 31 27 a8 5e |...}g|..?..C1'.^|
059622e6 ...
```

En la columna izquierda se encuentra el *offset* correspondiente a cada línea. En la parte central los datos del bloque en base 16 o hexadecimal y en la columna de la derecha se encuentra cada uno de los 16 datos hexadecimales de la columna central representados en caracteres ASCII[4]. Fijémonos en los datos hexadecimales de la columna central. Para facilitar su lectura se han resaltado en diferentes colores los distintos segmentos de datos. En color rojo aparecen los 4 *bytes* que representan el dato **magicID**, a continuación en color verde los 4 siguientes *bytes* el dato **size** que representa el tamaño final de este bloque. El segmento que sigue en color turquesa son los 80 *bytes* que representan el **header**. En color azul un único *byte* que representa el número de transacciones

[3] https://en.bitcoin.it/wiki/Running_Bitcoin#Command-line_arguments
[4] https://www.ieee.li/computer/ascii.htm

registradas en el bloque y finalmente en color morado se han resaltado todos los datos correspondientes a las transacciones del bloque. Los datos no terminan en el *offset* `059622e6`, se ha truncado el bloque en ese punto para une mejor visualización en este documento, pues la cantidad de datos del bloque mostrada en este formato podría abarcar demasiado espacio, pues desde este punto hasta el final del bloque todos los datos restantes son los datos de todas y cada una de las transacciones de este, y son muchas.

Para profundizar más en detalle sobre la estructura de datos de un bloque de la cadena de bloques de Bitcoin es recomendable consultar el apartado 2 del documento *Criptografía aplicada: Cálculo del hash SHA-256 de un bloque Bitcoin*[5].

3. Estructura de datos de una transacción

En el protocolo Bitcoin todos los bloques de la cadena de bloques contienen la misma estructura de datos para almacenar la información segmentada y ordenada en diferentes partes. Las transacciones son el segmento de datos más importante, es donde se ejecutarán sencillos programas o *scripts* que se encargarán de comprobar los saldos disponibles, firma de claves, comprobar la autenticación de las mismas y procesar el envío de saldo entre pares mediante el uso de criptografía de clave pública o asimétrica, la esencia de Bitcoin. A continuación se muestra un esquema de ejemplo en el que se puede ver la estructura de datos de une transacción en el que hay un *input* y dos *outputs*.

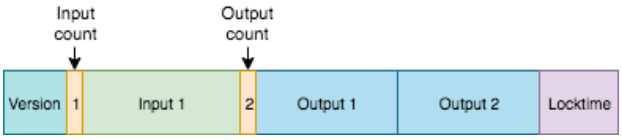

Figura 1: Transacción con un *input* y dos *outputs*.

En la siguiente imagen se muestra en detalle cada una de las partes que componen los segmentos de datos correspondientes a un *input* y a un *output*.

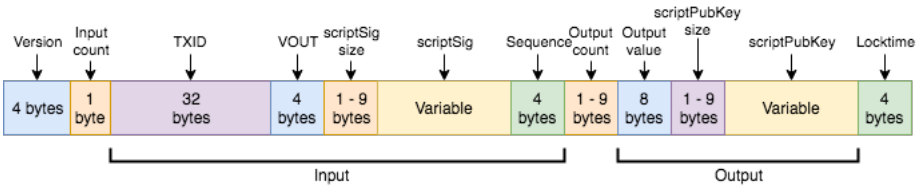

Figura 2: *Input* y *output* en detalle.

[5] https://github.com/JavierDominguezGomez/Bitcoin_cryptography/blob/master/Bitcoin_block_SHA_256_es.pdf

Aunque existen diferentes tipos de transacciones en este documento se explicará en detalle la más común, las transacciones de tipo P2PH *(pay to pubkey hash)*. En cada bloque la primera transacción es la denominada transacción *Coinbase*. Se trata de una transacción reservada para almacenar información relativa a la recompensa final por minar el bloque candidato y toda la información sobre comisiones adicionales en cada una de las transacciones que conformarán finalmente el bloque. Siguiendo con el ejemplo del bloque #286819, podremos encontrar el inicio de la transacción *Coinbase* a partir del *byte* número 93725215, en el *offset* `0x0596221f`, y en este caso abarca los siguientes 181 *bytes* tal y como se muestra a continuación.

```
~/.bitcoin/blocks/$ hexdump -C -s 93725215 -n 181 blk00116.dat
0596221f 01 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 |................|
0596222f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0596223f 00 00 00 00 00 ff ff ff ff 60 03 63 60 04 06 2f |.........`.c`../|
0596224f 50 32 53 48 2f 04 35 8b 05 53 08 44 04 f2 53 00 |P2SH/.5..S.D..S.|
0596225f 00 17 e4 46 52 2c fa be 6d 6d 69 06 88 fb 88 6c |...FR,..mmi....l|
0596226f 0d f0 c8 7c bc 7e a4 f7 f1 b5 c0 05 0b d0 ac 37 |...|. ..........7|
0596227f 51 cf c9 97 d9 d6 97 13 28 de 04 00 00 00 00 00 |Q.......(.......|
0596228f 00 00 48 61 70 70 79 20 4e 59 21 20 59 6f 75 72 |..Happy NY! Your|
0596229f 73 20 47 48 61 73 68 2e 49 4f 00 00 00 00 01 cb |s GHash.IO......|
059622af 81 31 95 00 00 00 00 19 76 a9 14 80 ad 90 d4 03 |.1......v.......|
059622bf 58 1f a3 bf 46 08 6a 91 b2 d9 d4 12 5d b6 c1 88 |X...F.j.....]...|
059622cf ac 00 00 00 00                                 |.....|
```

Otra manera alternativa de visualizar los datos de una transacción es mostrar todos los *bytes* concatenados, sin espacios, sin mostrar los *offset* de cada línea y sin mostrar tampoco la representación de los datos en caracteres ASCII. Se consigue mediante la ejeución de la siguiente línea de comandos:

```
~/.bitcoin/blocks/$ hexdump -C -s 93725215 -n 181 blk00116.dat | cut -c 11-58 |
tr '\n' ' ' | tr -d ' '
```

```
0x01000000010000000000000000000000000000000000000000000000000000
00000000000000000000ffffffff600360040062f503253482f04358b
0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cb
c7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000
004861707079204e592120596f757273204748617368e494f0000000
001cb813195000000001976a91480ad90d403581fa3bf46086a91b2d9
d4125db6c188ac00000000
```

En los siguientes puntos se explica en detalle uno a uno cada segmento de *bytes* de esta transacción *Coinbase*.

3.1. Versión

```
0x01000000010000000000000000000000000000000000000000000000000000
00000000000000000000ffffffff600363604062f503253482f04358b
0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cb
c7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000
004861707079204e592120596f7572732047486173682e494f0000000
001cb813195000000001976a91480ad90d403581fa3bf46086a91b2d9
d4125db6c188ac00000000
```

Se trata del el primer dato que se encuentra en una transacción y representa el número de versión para el registro de las transacciones, se trata de un número entero con una longitud de 4 *bytes* en formato *Little-Endian* que actualmente tiene un valor hexadecimal de `0x01000000` o 1 en base decimal.

```
0x01000000
```

3.2. Input count

```
0x0100000001000000000000000000000000000000000000000000000000000
00000000000000000000ffffffff600363604062f503253482f04358b
0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cb
c7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000
004861707079204e592120596f7572732047486173682e494f0000000
001cb813195000000001976a91480ad90d403581fa3bf46086a91b2d9
d4125db6c188ac00000000
```

Esta variable es un tiene por valor un número entero positivo de longitud variable, desde 1 hasta 9 *bytes*, y representa el número de entradas o *inputs* que tiene la transacción.

```
0x01
```

3.3. Input

```
0x010000000100000000000000000000000000000000000000000000000000
00000000000000000000ffffffff600363604062f503253482f04358b
0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cb
c7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000
004861707079204e592120596f7572732047486173682e494f0000000
001cb813195000000001976a91480ad90d403581fa3bf46086a91b2d9
d4125db6c188ac00000000
```

En una transacción hay un segmento de datos para las entradas o *inputs* y este a su vez se divide en los siguientes cinco segmentos:

- TXID
- VOUT
- scriptSig size
- scriptSig
- Sequence

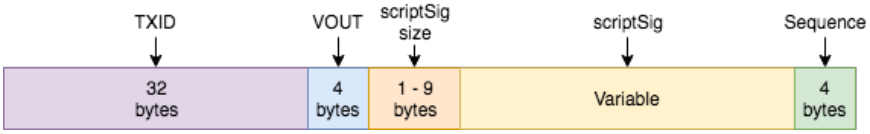

Figura 3: Los 5 segmentos de datos en una entrada o *input*.

3.3.1. TXID

```
0x0100000001000000000000000000000000000000000000000000000000
0000000000000000000ffffffff600363600406 2f503253482f04358b
0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cb
c7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000
004861707079204e592120596f7572732047486173682e494f0000000
001cb813195000000001976a91480ad90d403581fa3bf46086a91b2d9
d4125db6c188ac00000000
```

Se trata del un *hash* de 32 *bytes* que representa la transacción que contiene la salida o el *output* que se va a gastar en esta transacción. Una de las características que diferencia a una transacción de tipo *Coinbase* del resto es que el valor de TXID es un numero hexadecimal con valor cero.

```
0x0000000000000000000000000000000
0000000000000000000000000000000
```

En una transacción corriente este dato se obtiene aplicando la función hash SHA-256 dos veces a los datos de la transacción en formato *Little-Endian*.

3.3.2. VOUT

```
0x0100000001000000000000000000000000000000000000000000000000
0000000000000000000ffffffff600363600406 2f503253482f04358b
0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cb
c7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000
004861707079204e592120596f7572732047486173682e494f0000000
001cb813195000000001976a91480ad90d403581fa3bf46086a91b2d9
d4125db6c188ac00000000
```

En Bitcoin todas las salidas u *outputs* de una transacción se almacenan en un dato de tipo vector. VOUT es un número hexadecimal con una longitud de 4 *bytes* que representa el índice de las salidas u *outputs* de una transacción en dicho vector. Otra de las características de una transacción de tipo *Coinbase* es que el valor para VOUT es siempre `0xffffffff`, el número de 4 *bytes* de más alto valor.

```
0xffffffff
```

3.3.3. scriptSig size

```
0x0100000001000000000000000000000000000000000000000000000000000000
00000000000000000000ffffffff6003636004062f503253482f04358b
0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cb
c7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000
004861707079204e592120596f7572732047486173682e494f0000000
001cb81319500000001976a91480ad90d403581fa3bf46086a91b2d9
d4125db6c188ac00000000
```

Esta variable es un tiene por valor un número hexadecimal de longitud variable, desde 1 hasta 9 *bytes* que representa el tamaño en *bytes* que tendrán los datos almacenados en *scriptSig*.

```
0x60
```

En este caso el tamaño es de `0x60`, que tras convertirlo a base decimal son 96 *bytes*. Para este dato existe límite de tamaño que actualmente es de 1650 *bytes*.

3.3.4. scriptSig

```
0x0100000001000000000000000000000000000000000000000000000000000000
00000000000000000000ffffffff6003636004062f503253482f04358b
0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cb
c7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000
004861707079204e592120596f7572732047486173682e494f0000000
001cb81319500000001976a91480ad90d403581fa3bf46086a91b2d9
d4125db6c188ac00000000
```

Cuando en una transacción se generan salidas u *outputs*, estas salidas se establecen con un *script* de bloqueo, por lo tanto para poder utilizarlas como entradas o *inputs* en las nuevas transacciones antes se debe debe utilizar un *script* de desbloqueo llamado *scriptSig*. Se trata de un dato de de longitud variable, en este caso tiene una longitud de 96 *bytes* pero podría contener más o menos información, dependiendo de las operaciones que se quieran realizar

en el *script* o el mensaje que se quiera incluir. Adicionalmente lo utilizan los mineros para incluir mensajes de texto personalizados empleando caracteres ASCII, normalmente el nombre del minero o del pool de minería que ha resuelto el hash válido para poder minar el bloque. En este caso la transacción *Coinbase* contiene el siguiente valor para el segmento de datos *scriptSig*.

```
0x03636004062f503253482f04358b0553084404f253000017e446522c
fabe6d6d690688fb886c0df0c87cbc7ea4f7f1b5c0050bd0ac3751cfc9
97d9d6971328de040000000000000004861707079204e592120596f7572
732047486173682e494f
```

En una representación de carácteres ASCII se vería de la siguiente forma:

```
`.c`../P2SH/.5..S.D..S....FR,..mmi....l...|.^....
......7Q.......(.........Happy NY! Yours GHash.IO
```

Realmente esta cadena de 96 *bytes* es un *script* o programa con una secuencia de instrucciones ordenadas en lenguaje *Bitcoin Script* o simplemente *script*. En la siguiente muestra se pueden ver los diferentes segmentos en los que se divide la cadena hexadecimal de *scriptSig* en el caso concreto de esta transacción de tipo *Coinbase* de ejemplo.

`0x03`	⇒	**OP_PUSHBYTES_3**
`0x636004`	⇒	`0x636004`
`0x06`	⇒	**OP_PUSHBYTES_6**
`0x2f503253482f`	⇒	`0x2f503253482f`
`0x04`	⇒	**OP_PUSHBYTES_4**
`0x358b0553`	⇒	`0x358b0553`
`0x08`	⇒	**OP_PUSHBYTES_8**
`0x4404f253000017e4`	⇒	`0x4404f253000017e4`
`0x46`	⇒	**OP_PUSHBYTES_70**
`0x522cfabe6d6d690688fb886c`		`0x522cfabe6d6d690688fb886c`
`0df0c87cbc7ea4f7f1b5c0050b`		`0df0c87cbc7ea4f7f1b5c0050b`
`d0ac3751cfc997d9d6971328de`	⇒	`d0ac3751cfc997d9d6971328de`
`040000000000000004861707079`		`040000000000000004861707079`
`204e592120596f757273204748`		`204e592120596f757273204748`
`6173682e494f`		`6173682e494f`

Este *script* introducirá diferentes datos en una pila que inicialmente está vacía. Para evitar que varias transacciones de tipo *Coinbase* pertenecientes a diferentes bloques tengan el mismo valor en TXID se decidió implementar mediante la propuesta BIP34[6] un sistema que añade al inicio del *script* un bloque de 4 *bytes*, dividido en las siguientes dos secciones. La primera es una constante de 1 *byte* de longitud que representa un OP_CODE, en este caso con valor `0x03`, e indica que el siguiente dato que se va a colocar en la pila tiene una longitud

[6] https://github.com/bitcoin/bips/blob/master/bip-0034.mediawiki

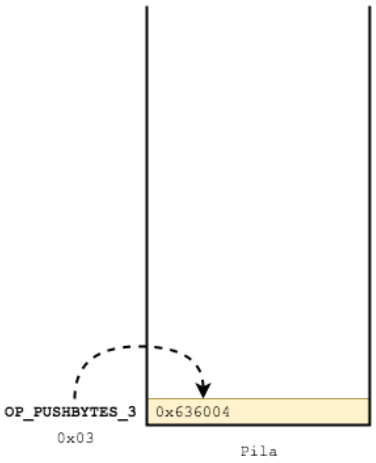

Figura 4: Adición de 3 *bytes* a pila.

de 3 *bytes*. Para ello se utiliza el OP_CODE **OP_PUSHBYTES_3**. La segunda sección es un dato de 3 *bytes* en formato *Little-Endian* que representan la altura o el número del bloque al que pertenece la transacción *Coinbase*, en este caso `0x636004`. La transacción *Coinbase* que se ha utilizado en este ejemplo pertenece al bloque número #286819 de la cadena de bloques de *Bitcoin* en la red principal *mainnet*. Si se convierten los 3 *bytes* `0x636004` a formato *Big-Endian* se obtiene el valor hexadecimal `0x046063`, que en base 10 o decimal es el número 286819 y se corresponde con el número de bloque al que pertenece esta transacción de ejemplo. Así pues, mediante la instrucción **OP_PUSHBYTES_3** se añaden los siguientes 3 *bytes* a la pila, tal y como se muestra en la figura 2.

A continuación de los 3 *bytes* que ya se han añadido en el fondo de la pila, se encuentra 1 *byte* que representa otro OP_CODE, en este caso **OP_PUSHBYTES_6** con valor `0x06`. Esta operación va a introducir en la pila los siguientes 6 *bytes* que se encuentran a continuación del *byte* `0x06`, es decir, `0x2f503253482f`, tal y como se muestra en la figura 3.

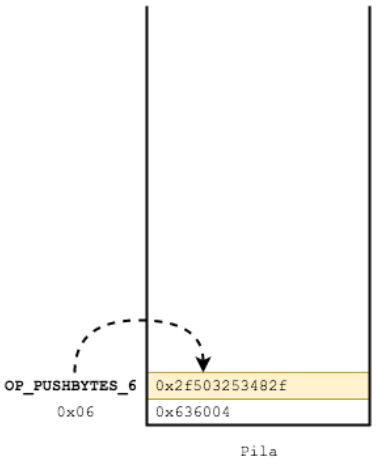

Figura 5: Adición de 6 *bytes* a pila.

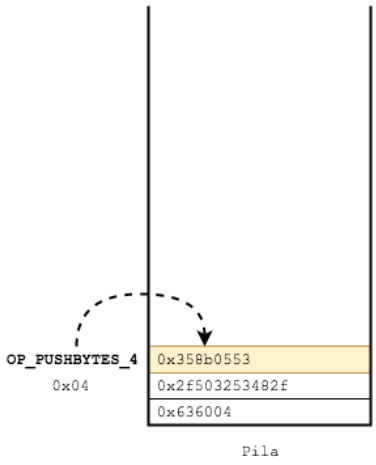

Figura 6: Adición de 4 *bytes* a pila.

El siguiente *byte* tiene valor `0x04`, es el OP_CODE **OP_PUSHBYTES_4**. Este

añade los siguientes 4 *bytes* a la pila, `0x358b0553`, tal y como se muestra en la figura 4. Después viene otro *byte* que tiene como valor `0x08` y representa el OP_CODE **`OP_PUSHBYTES_8`**, de igual manera que los anteriores OP_CODES este añadirá los siguientes 8 *bytes* dentro de la pila, tal y como se muestra en la figura 5.

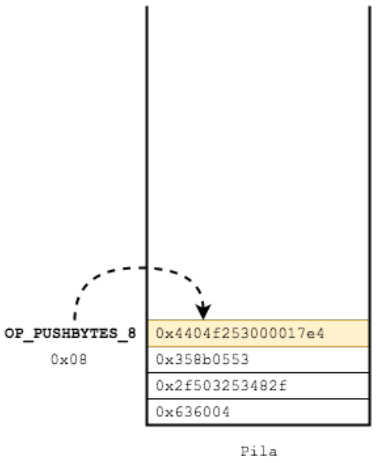

Figura 7: Adición de 8 *bytes* a pila.

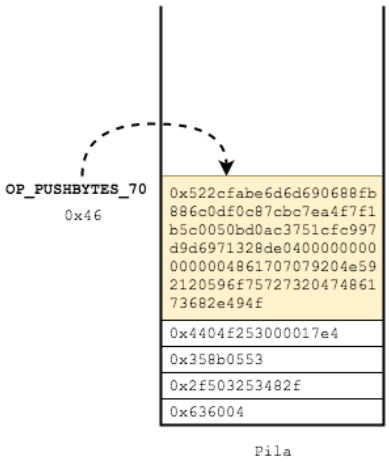

Figura 8: Adición de 70 *bytes* a pila.

El siguiente *byte* tiene valor `0x46`, es el OP_CODE **`OP_PUSHBYTES_70`** (el número hexadecimal 46 en base 10 o decimal es 70). Este añade los siguientes 70 *bytes* a la pila, `0x522cfabe6d6d690688fb886c0df0c87cbc7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de0400000000000000048617070792 04e592120596f75727320474861 73682e494f`, tal y como se muestra en la figura 6. Estos 70 *bytes* representan la clave pública generada a partir de una clave privada. Esta clave pública se tendrá que firmar para poder desbloquear la transacción y enviar los fondos. Adicionalmente se utilizará para obtener la dirección Bitcoin a la que se ha de transferir el monto de la transacción.

3.3.5. Sequence

```
0x0100000001000000000000000000000000000000000000000000000000000000
0000000000000000000ffffffff600363600406 2f503253482f04358b
0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cb
c7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000
004861707079204e592120596f75727320474861 73682e494f0000000
001cb81319500000001976a91480ad90d403581fa3bf46086a91b2d9
d4125db6c188ac00000000
```

Se trata de un número de 4 *bytes* de longitud, se utiliza para permitir que las transacciones no confirmadas que tengan un *Locktime* se actualicen antes de darlas por finalizadas o confirmadas. También se utiliza para desactivar el *Locktime* de la transacción. En el caso de una transacción de tipo *Coinbase* se establece este valor a cero.

```
0x00000000
```

Se implementó en el año 2015 mediante la propuesta BIP125[7], hasta entonces no se le daba uso.

3.4. Output

```
0x0100000001000000000000000000000000000000000000000000000000000000
00000000000000000000ffffffff6003636004062f503253482f04358b
0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cb
c7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000
004861707079204e592120596f7572732047486173682e494f0000000
001cb8131950000000001976a91480ad90d403581fa3bf46086a91b2d9
d4125db6c188ac00000000
```

En una transacción hay un segmento de datos para las salidas u *outputs* y este a su vez se divide en los siguientes cinco segmentos:

- Output count
- Output value
- scriptPubKey size
- scriptPubKey
- Locktime

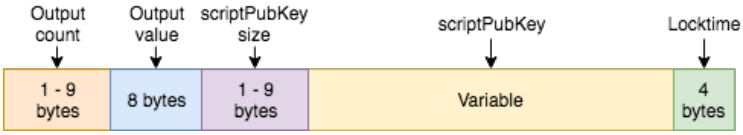

Figura 9: Los 5 segmentos de datos en una salida u *output*.

[7] https://github.com/bitcoin/bips/blob/master/bip-0125.mediawiki

3.4.1. Output count

```
0x0100000001000000000000000000000000000000000000000000000000000000
0000000000000000000ffffffff6003636004062f503253482f04358b
0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cb
c7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000
004861707079204e592120596f7572732047486173682e494f0000000
001cb813195000000001976a91480ad90d403581fa3bf46086a91b2d9
d4125db6c188ac00000000
```

Esta variable es un tiene por valor un número entero positivo de longitud variable, desde 1 hasta 9 *bytes*, y representa el número de salidas u *outputs* que tiene la transacción.

```
0x01
```

3.4.2. Output (Value)

```
0x0100000001000000000000000000000000000000000000000000000000000000
0000000000000000000ffffffff6003636004062f503253482f04358b
0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cb
c7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000
004861707079204e592120596f7572732047486173682e494f0000000
001cb813195000000001976a91480ad90d403581fa3bf46086a91b2d9
d4125db6c188ac00000000
```

Se trata de un número de 8 *bytes* de longitud en formato *Little-Endian* que representa la cantidad de satoshis[8] que podrán ser reclamados en futuras transacciones como *input* para ser gastados de nuevo.

$$0x\overbrace{cb81319500000000}^{Little-Endian} \Rightarrow 0x\overbrace{00000000953181cb}^{Big-Endian} = 2503049675_{10}$$

En este caso 2503049675 *satoshis* son 25.03049675 *bitcoins* o BTC, y como es el *output* de la transacción de tipo es la recompensa que se llevará quien mine este bloque.

[8] 1 bitcoin = 100.000.000 satoshis

3.4.3. Output (scriptPubKey size)

```
0x0100000001000000000000000000000000000000000000000000000000000000
0000000000000000000ffffffff6003636004062f503253482f04358b
0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cb
c7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000
004861707079204e592120596f7572732047486173682e494f0000000
001cb8131950000000019 76a91480ad90d403581fa3bf46086a91b2d9
d4125db6c188ac00000000
```

Esta variable es un tiene por valor un número hexadecimal de longitud variable, desde 1 hasta 9 *bytes* que representa el tamaño en *bytes* que tendrán los datos almacenados en *scriptPubKey*.

```
0x19
```

En este caso el tamaño es de `0x19`, que tras convertirlo a base decimal son 25 *bytes*.

3.4.4. Output (scriptPubKey)

```
0x0100000001000000000000000000000000000000000000000000000000000000
0000000000000000000ffffffff6003636004062f503253482f04358b
0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cb
c7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000
004861707079204e592120596f7572732047486173682e494f0000000
001cb8131950000000019 76a91480ad90d403581fa3bf46086a91b2d9
d4125db6c188ac 00000000
```

Es un dato de longitud variable, en este caso de 25 *bytes*, pero podría contener más o menos información. Se trata de un *script* también llamado *locking script* que implementa un mecanismo de bloqueo para un *output* o salida, de modo que esta no pueda ser gastada hasta que no se desbloquee tras cumplirse una serie de condiciones, la mayoría de la veces ser el poseedor de la clave privada que puede desbloquear el *script* en el que se utiliza una clave pública.

```
0x76a91480ad90d403581fa3bf46086a91b2d9d4125db6c188ac
```

Esta cadena de 25 *bytes*, al igual que *scriptSig* es una secuencia de instrucciones ordenadas en lenguaje *Bitcoin Script* o simplemente *script*. En la siguiente muestra se pueden ver los diferentes segmentos en los que se divide la cadena hexadecimal de *scriptPubKey* en el caso concreto de esta transacción de tipo *Coinbase* de ejemplo.

```
0x76                        ⇒  OP_DUP
0xa9                        ⇒  OP_HASH160
0x14                        ⇒  OP_PUSHBYTES_20
0x80ad90d403581fa3bf4       ⇒  0x80ad90d403581fa3bf4
6086a91b2d9d4125db6c1          6086a91b2d9d4125db6c1
0x88                        ⇒  OP_EQUALVERIFY
0xac                        ⇒  OP_CHECKSIG
```

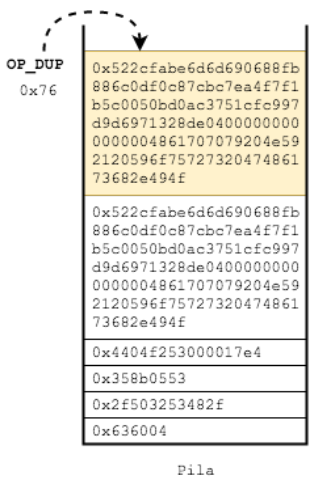

Figura 10: Adición de 8 *bytes* a pila.

Al incio de esta cadena de *bytes* se encuentra el *byte* `0x76` y representa el OP_CODE **OP_DUP**, cuya finalidad es duplicar el elemento ubicado en la cima de la pila. Hay que tener en cuenta que en este momento la pila no se encuentra vacía, tiene los elementos que se introdujeron en la ejecución de *scriptSig*, punto 3.3.4 de este documento, en la Figura 6. En este caso en la cima de la pila se encuentran los 70 *bytes* correspondientes a una clave pública. El siguiente *byte* tiene por valor `0xa9`, que representa al OP_CODE **OP_HASH160**. Este OP_CODE recibe como dato de entrada el dato que se encuentra en este momento en la cima de la pila, es decir, la clave pública que se acaba de duplicar recientemente. A continuación procesa el dato de entrada en dos pasos, en primer lugar aplica a mensaje de entrada la función SHA-256 y después la función RIPEMD-160, obteniendo finalmente un *hash* con una tamaño de 20 *bytes* que sustituirá al dato de entrada de la pila, tal y como se muestra en la Figura 9.

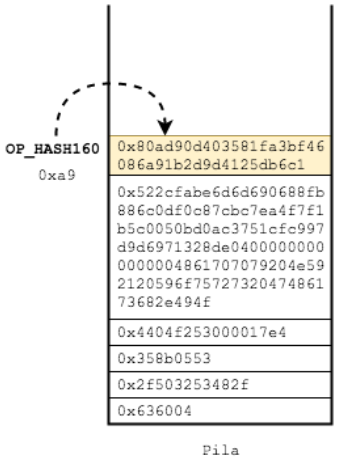

Figura 11: *Hash* de la clave pública.

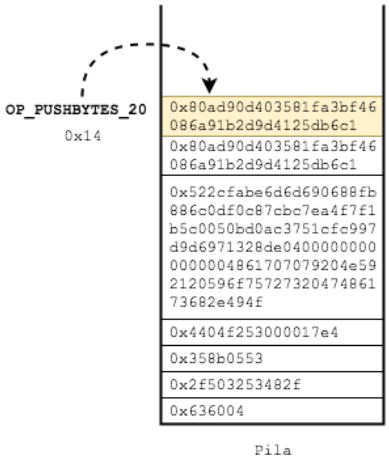

Figura 12: Adición de 70 *bytes* a pila.

A continuación se tiene un *byte* con valor `0x14` que representa el OP_CODE **`OP_PUSHBYTES_20`**. Este introduce en a la pila los siguientes 20 *bytes* se encuentran en el segmento de datos *scriptPubKey*, tal y como se muestra en la Figura 10. El siguiente *byte* tiene valor `0x88` y representa el OP_CODE **`OP_EQUALVERIFY`**. Se encargará de comprobar si los dos elementos que se encuentran en la cima de la pila son exactamente iguales, en cuyo caso marca la transacción como válida y elimina sendos elementos de la pila. Si se diera el caso de que los dos elementos de la cima de la pila no fueran iguales, la transacción se marca como no válida. En cualquier caso siempre se eliminan los dos elementos de la pila.

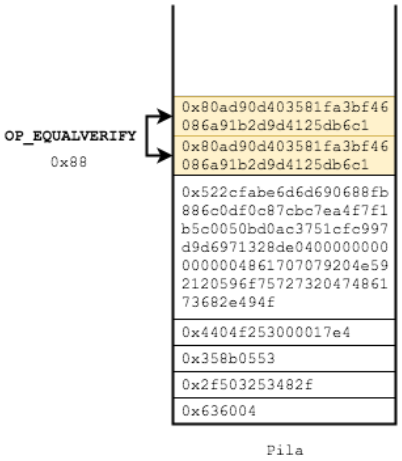

Figura 13: Comprobación de igualdad.

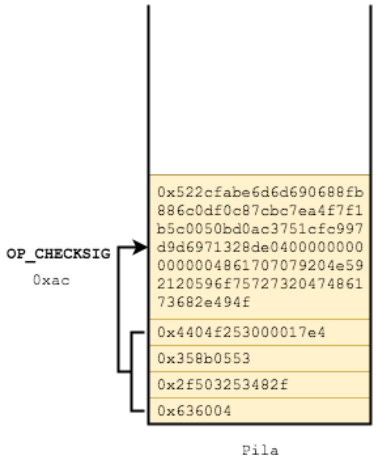

Figura 14: Firma de la clave pub.

Por último se tiene un *byte* con valor `0xac` que representa el OP_CODE **`OP_CHECKSIG`**. Este OP_CODE toma los elementos de la pila correspondientes a *scriptSig* y los usa para firmar la clave pública que en este momento se encuentra en la cima de la pila. Si todo va bien y la firma es válida se vacía procede al pago de la transacción en la dirección Bitcoin especificada y por último se vacía la pila.

3.4.5. Locktime

```
0x0100000001000000000000000000000000000000000000000000000000000000
00000000000000000ffffffff6003636004062f503253482f04358b
0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cb
c7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000
004861707079204e592120596f7572732047486173682e494f0000000
001cb81319500000001976a91480ad90d403581fa3bf46086a91b2d9
d4125db6c188ac00000000
```

Se trata de un número de 4 *bytes* de longitud en formato *Little-Endian* que representa una medida de bloqueo para una transacción, es decir, una condición que mantiene bloqueada la transacción hasta que se cumpla la premisa. El valor de ese consa premisa se establece mediante edicionalque tenga *Locktime*, y ese valor puede hacer referencia bien a una altura de bloque o bien a una fecha en formato *epoch* o *unixtime*[9]. Una vez se cumpla la fecha o la altura del bloque ya se podrá hacer uso de esa transacción. En el caso de una transacción de tipo *Coinbase* no se establece ningún periodo de bloqueo.

[9]https://es.wikipedia.org/wiki/Tiempo_Unix

`0x00000000`

La mayoría de las transacciones no utilizan el tiempo de bloqueo, por lo que suele establecer su valor a `0x00000000`. Por el contrario, se puede utilizar para asegurarse de que una transacción permanece bloqueada por un tiempo específico o hasta que no se llegue a una altura de bloque indicado.