What's inside a Grin Transaction File?

Brandon Arvanaghi Follow
Mar 6 · 7 min read

Unlike other cryptocurrencies, Grin does not use *addresses*. To send or receive Grin, the sender and recipient must send information back-and-forth between one-another.

One way to do this is through a **file**. This file can be shared over a period of seconds, days, years, or anything else.

Let's examine the contents of this Grin transaction file as it gets passed around.

. . .

Before reading this post, I recommend reading "**Grin Transactions Explained, Step-by-Step**" to understand some of these fields.

Exploring Grin Transaction Files

Say you want to send 10.25 Grin to Bob, and you two want to exchange the necessary information for this transaction through a file. You run:

```
grin wallet send -m file -d my_grin_transaction.tx 10.25
```

This generates a file called `my_grin_transaction.tx` which you need to send to Bob so he can provide you his *response* file.

Let's open this `my_grin_transaction.tx` file.


```
{
 "num_participants": 2,
 "id": "56709cfe-8584-4a02-b94e-bb7e79cfae66",
 "tx": {
  "offset": [62, 48, 100, 178, ...],
  "body": {
   "inputs": [{
    "features": "Plain",
    "commit": [8, 95, 99, 251, ...]
   }],
   "outputs": [{
    "features": "Plain",
    "commit": [8, 216, 52, 10, ...],
    "proof": [13, 107, 82, 188, ...]
   }],
   "kernels": [{
    "features": "HeightLocked",
    "fee": 8000000,
    "lock_height": 22023,
    "excess": [0, 0, 0, 0, ...],
    "excess_sig": [0, 0, 0, 0, ...]
   }]
  }
 },
 "amount": 10250000000,
 "fee": 8000000,
 "height": 22023,
 "lock_height": 22023,
```

```
  "participant_data": [{
    "id": 0,
    "public_blind_excess": [2, 66, ...],
    "public_nonce": [2, 102, ...],
    "part_sig": null,
    "message": null,
    "message_sig": null
  }]
 }
```

In order, we have:

- **num_participants**: The number of people involved in this transaction. Alice to Bob means 2 participants. Alice to Bob and Charlie means 3.
- **id**: A unique identifier for this transaction. This allows us to keep track of the transaction when we receive a file back from Bob.

Next, we have a **tx** object, short for *transaction*. This consists of:

1. A *kernel offset* (named **offset** here).
2. A *body* object, containing the **inputs** to be spent in this transaction transaction, the **outputs** for this transaction (currently, just our change), and the transaction **kernel**.

Let's dive into these.

- **offset**: The kernel offset. This is a random value we choose to subtract from our *blinding factor* to make it more difficult for anyone to track this transaction on the blockchain.

Next in this *tx* object is the **inputs** object. This object contains information about the *output we are spending* in this transaction.

```
"inputs": [{
    "features": "Plain",
    "commit": [8, 95, 99, 251, ...]
 }],
```

- **features**: In this example, the *features* of the output we are spending (the *input* to this transaction) is set to "Plain". This *features* variable can be either "Plain" or "Coinbase".

A *Coinbase* output was an output generated from a mining reward. A *Plain* output just means it was not a Coinbase output.

- **commit**: The output we are trying to spend. An output in Grin is a *Pedersen Commitment*. To understand how these work, you can read the Grin wiki.

Still in the *tx* object, we move on to the **outputs** array.

```
"outputs": [{
    "features": "Plain",
    "commit": [8, 216, 52, 10, ...],
    "proof": [13, 107, 82, 188, ...]
 }],
```

You'll notice this **outputs** array already has an output for our transaction. This is our *change output*. If we spend 10.25 Grin, and if we are only using one *input* in this transaction (as is the case here), then that input contains *at least* 10.25 Grin. If this input consists of 20 Grin, for example, then this transaction also requires an output to send 9.75 Grin back to us!

Grin is unique in that Bob needs to be involved in any output that goes to him, and since we haven't actually sent this file to Bob yet, we're not ready to create that part of the transaction. Our change output has nothing to do

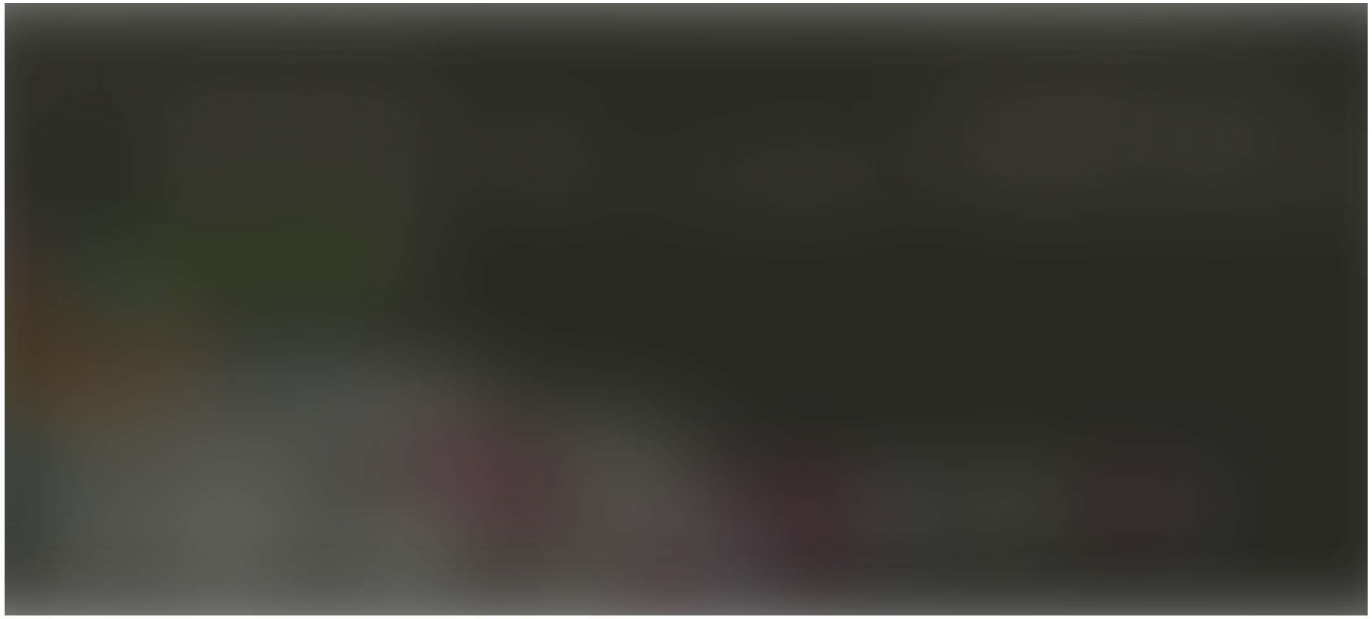
with Bob, though, which is why we are already able to compute our *change output*.

Besides the **features** and **commit**, this output contains a **proof**. This is the range proof for the output, and it proves that the amount of Grin in that output is **non-negative.** Range proofs allow us to cryptographically prove the output is non-negative without revealing what that output actually is.

The next portion of this transaction file is the **kernels** array.

```
"kernels": [{
    "features": "HeightLocked",
    "fee": 8000000,
    "lock_height": 22023,
    "excess": [0, 0, 0, 0, ...],
    "excess_sig": [0, 0, 0, 0, ...]
 }]
```

The **transaction kernel** also starts with the *features* variable. This part may be a little confusing: the *features* variable you see for any output is either "Plain" or "Coinbase", as I explained earlier. But in the transaction kernel, the features variable can either be "Plain", "Coinbase", or "HeightLocked".



The three different kinds of "features" in a transaction kernel: Coinbase, Plain, and HeightLocked

A "Plain" transaction means a *lock_height* of 0. In other words, once this transaction is broadcast to the network, it can be mined immediately. As soon as any Grin transaction is mined, its outputs can be spent. A *HeightLocked* transaction, in contrast, means that the transaction can't be mined until a certain block number.

Once mined, the output will have a *feature* of either "Plain" or "Coinbase", as we saw above, because the fact that it may have been *HeightLocked* before it was mined is no longer relevant.

```
"outputs": [{
    "features": "Plain",
    "commit": [8, 216, 52, 10, ...],
    "proof": [13, 107, 82, 188, ...]
 }],
```

Moving on, the transaction kernel also contains the *fee* and *lock_height*, the *kernel excess* (denoted here as **excess**), and the *signature* of the transaction (denoted here as **excess_sig)**.

We can't yet compute the *excess* and *excess_sig* until we receive the response file back from Bob. This is why these arrays are both filled with 0s.

After the *tx* object, we have these remaining fields:

```
"amount": 10250000000,
"fee": 8000000,
"height": 22023,
"lock_height": 22023,
"participant_data": [{
  "id": 0,
  "public_blind_excess": [2, 66, ...],
  "public_nonce": [2, 102, ...],
  "part_sig": null,
  "message": null,
  "message_sig": null
}]
```

- **amount**: The amount of Grin being transferred, as a multiple of the atomic unit *1 nanoGrin*. A nanoGrin is one billionth of a Grin. Only the sender and recipient will ever see this amount.
- **fee**: The mining fee for the transaction, also as a multiple of 1 n*anoGrin*.
- **height:** The block number at which this transaction file was created.
- **lock_height:** The block number at which the outputs from this transaction become spendable. You'll notice in this case, the lock_height is equal to the *height* variable, meaning this transaction can be mined immediately.

Next, we have a **participant_data** array.

```
"participant_data": [{
  "id": 0,
  "public_blind_excess": [2, 66, ...],
  "public_nonce": [2, 102, ...],
  "part_sig": null,
  "message": null,
  "message_sig": null
}]
```

This contains the information passed back-and-forth between the participants of the transaction. So far, this contains only *our* data, since we have not yet shared this file with Bob.

- **id:** The participant's ID in this transaction. An ID of 0 corresponds to the *sender* of the transaction, which is us.
- **public_blind_excess**: The commitment to the *sum of our blinding factors*. To understand what this is, you can read Grin Transactions Explained, Step-by-Step.
- **public_nonce**: The commitment to our *nonce*. This is used to build the transaction signature. More information on this can again be found in Grin Transactions Explained, Step-by-Step.
- **part_sig:** Our partial signature is empty, because we're not able to create it until we receive data back from Bob.
- **message:** An extra "message" that can be added to your transaction. If you create this file with a `-g` flag and put any arbitrary string afterwards, that message would appear here.

Note: *when people generally refer to the "message" of a Grin transaction, they typically mean the transaction fee, if any, and the lock_height, if any. This*

extra "message" field is entirely separate: you can add a string to send to Bob, and vice-versa, that never gets published on-chain.

- **message_sig:** The extra "message" field I just mentioned is signed by your *sum of all blinding factors*. That signature is placed here, and Bob can verify that message by using your commitment to your sum of all blinding factors. Bob can add his own extra *message* and *message_sig* to the file once he receives this file. We can verify his message the same way.

We send this file to Bob, and await his *response* file in return.

Bob's Response File

Bob accepts this file in his Grin client with the following command:

```
grin wallet receive -I my_grin_transaction.tx
```

This command generates a **my_grin_transaction.tx.response** file for Bob that he will send back to us.

Let's open this file.


```
{
 "num_participants": 2,
 "id": "56709cfe-8584-4a02-b94e-bb7e79cfae66",
 "tx": {
  "offset": [62, 48, 100, ...],
  "body": {
   "inputs": [{
    "features": "Plain",
    "commit": [8, 95, 99, ...]
   }],
   "outputs": [{
    "features": "Plain",
    "commit": [8, 206, 65, ...],
    "proof": [36, 101, 181, ...]
   }, {
    "features": "Plain",
    "commit": [8, 216, 52, ...],
    "proof": [13, 107, 82, ...]
   }],
   "kernels": [{
    "features": "HeightLocked",
    "fee": 8000000,
    "lock_height": 22023,
    "excess": [0, 0, 0, ...],
    "excess_sig": [0, 0, 0, ...]
   }]
  }
 },
 "amount": 10250000000,
 "fee": 8000000,
 "height": 22023,
 "lock_height": 22023,
 "participant_data": [{
  "id": 0,
```

```
    "public_blind_excess": [2, 66, 3, ...],
    "public_nonce": [2, 102, 201, ...],
    "part_sig": null,
    "message": null,
    "message_sig": null
   }, {
    "id": 1,
    "public_blind_excess": [3, 99, 72, ...],
    "public_nonce": [3, 246, 48, ...],
    "part_sig": [90, 35, 157, ...],
    "message": null,
    "message_sig": null
   }]
  }
```

As we can see, this file is everything we sent to Bob, plus some of his own data he added on top of it.

In the **participant data** array, we see a new entry for a participant with an **id** of 1 (Bob). Bob adds his commitment to his blinding factor, along with his commitment to his nonce, and finally his partial signature.

Finalizing the Transaction

When we receive Bob's response file, we run the following to *finalize* the transaction:

```
grin wallet finalize -i my_grin_transaction.tx.response
```

We don't actually see the missing pieces filled in — instead, we see:

grin wallet finalize command run successfully

We now has all the information we need from Bob to create his output and add it to the **outputs** array. Behind the scenes, our Grin client fills in the missing pieces (like **excess** and **excess_signature**), serializes this data into a byte array, and sends it across the Grin network for validation.

Soon, this Grin transaction will be mined, Bob will receive his 10.25 Grin, and we will receive our change.

. . .

If you enjoyed this post, follow @arvanaghi on Twitter.

. . .

Thanks to DavidBurkett for reviewing.

Bitcoin Blockchain Cryptocurrency Mimblewimble Grin