

Understanding Monero Cryptography, Privacy Part 2 -- Stealth Addresses

luigi1111 ⁽⁵⁵⁾ ▾ (/@luigi1111)in #monero (/trending/monero) • 6 years ago (edited)

This is part two of a series of unknown size; it'll be done when it's done. Part one is here (<https://steemit.com/monero/@luigi1111/understanding-monero-cryptography-privacy-introduction>).

Part two focuses on stealth addresses, an essential part of the protocol.

Note: Monero is based on the Cryptonote protocol -- though it has diverged and will continue to diverge -- along with numerous other coins; much of this series applies equally well to the others with some caveats. Monero is easily the largest and most active Cryptonote-based project.

Hello! I am back with part two of my series on Monero cryptography and privacy; in this series, I'm attempting to make the concepts as easy to understand as possible. I want you to have the same "lightbulb" moments I had when I first "got" these concepts (my understanding has evolved over a period of time).

In part two, we will be discussing stealth addresses, though you may learn some new concepts along the way. Stealth addresses are one of the two complementary techniques used to provide sender/receiver privacy in Monero.

So, what are stealth addresses? Well, they look like this:



(just kidding)

In my own words, stealth addressing is a technique whereby a **sender** can take a **recipient's** public address and **transform** it to a one-time address such that:

1. it is **publicly unlinkable** to the original public address;
2. it is **publicly unlinkable** to **any** other one-time address;
3. only the **recipient** can link all their payments together
4. only the **recipient** can derive the secret key associated with the one-time address

Using stealth addressing, a recipient can publish one address and receive unlimited* publicly unlinkable payments.

*[The chance of a collision (two stealth addresses being the same) is cryptographically negligible. Using the Birthday Paradox we can roughly estimate it would take \sqrt{I} , or about 2^{126} , stealth addresses being created before having a 50% chance of a collision. The result would be that the colliding addresses become publicly linkable to each other, but not to any others. There is another, worse problem that would occur in Monero if two stealth addresses were to collide; this will be explained in the ring signature article.

For an analogy on how large 2^{126} is, imagine the world has 10 billion people. Each and every person sends a payment to **every** other person once per **second**. We would reach 2^{126} payments in about 27 billion years.]

Stealth addresses can be implemented by any currency, including Bitcoin, but **by themselves** do not provide significant extra privacy over avoiding address reuse. However, they are very handy for certain uses, e.g., a published donation address, where address reuse is basically unavoidable.

Now you know what stealth addresses **are**, but you probably don't yet "get" them or how they work. *Unless you understood previously, in which case why are you reading this article?*

Before getting into the specifics of stealth addresses in Monero, we need to discuss something:

ECDH

Elliptic Curve Diffie-Hellman is a variant of the original Diffie-Hellman key agreement protocol extended for use with ECC. In simple terms, two parties can independently generate a shared secret over an unsecured connection (implying that no observer can discover the secret by simply watching their communication). Basic ECDH is quite simple to understand with the application of the scalar multiplication technique from my previous article. For simplicity all key pairs will always be referred to as a lowercase/capital letter (private key a , corresponding public key A , etc).

First, we have Alice. Alice has chosen a random private key (scalar) a from our group $[1, l-1]$. Her public key is $A = aG$.

Similarly, Bob chooses random private key b . His public key is $B = bG$.

Knowing we can add points together, Alice could compute point $C = A + B$, but *so could any observer*. C is **shared**, but it isn't **secret**.

Instead, remembering that A and B are curve points, and that we can add a point to itself (scalar multiplication!), Alice computes point $D = aB$. This is just like computing $B = bG$, but with a different "base point". Knowing the result D and the base B is no different with respect to helping an observer learn a than knowing result A and base G !

Bob, in turn, can also compute $D' = bA$ (using the apostrophe to be a second "try" at the same thing). Now Alice and Bob have a **shared, secret** point known only to them! $D = D'$

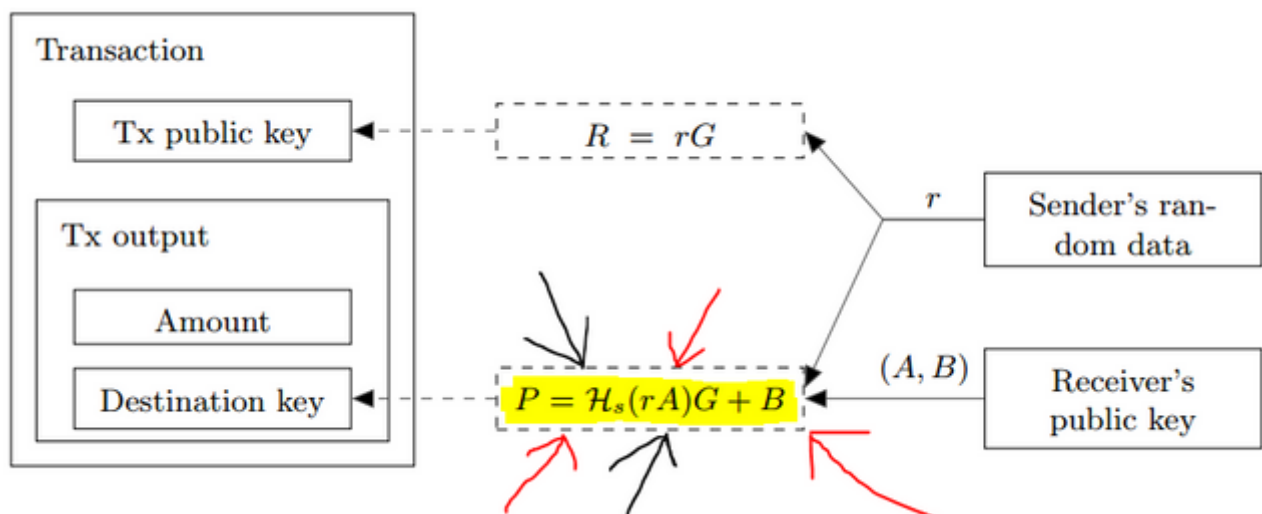
In case it isn't clear why Alice's D equals Bob's D' , here's an example:

1. Use base point G .
2. Use $a = 3$; $A = 3G$.
3. Use $b = 4$; $B = 4G$.
4. $a * b = 12$.
5. "Alice's" $D = aB = 3B = 3*4G = 12G$.
6. "Bob's" $D' = bA = 4A = 4*3G = 12G$!

Note that D has a corresponding scalar d (12 in the above example), that *no one knows*. We only know it in this case because we know both a and b ! *This isn't particularly useful information, but is the kind of thing I find interesting.*

Stealth Addresses

Now that you know about ECDH, let's move on to how **dual-key stealth addresses** actually work! Again from the Cryptonote whitepaper , we get:



As I've helpfully highlighted and pointed out here, we're looking at $P = Hs(rA)G + B$.

Dual-key simple refers to the pairs of spend/view keys, which allows "decoding" (or removing the unlinkability if you will) stealth addresses *without simultaneously allowing them to be spent*.

Now, ignore all that. Let's back up.

Alice wants to send a payment to Bob. Alice's **private spend key** is z , and her **private view key** is y . Her public address is then (Z, Y) in holy whitepaper order. I've helpfully used letters that aren't referenced above, *because Alice's keys aren't used at all*.

Bob's **public address** is (A, B) . Remember from the previous article that the whitepaper helpfully uses A as the public view key and B as the public spend key. Presumably Bob's **private keys** would be (a, b) , but Alice (our current perspective) doesn't know them.

The final piece needed before "building" our first stealth address is r and R . r is a new random scalar chosen by Alice for the express purpose of creating a stealth address for Bob. R is the corresponding curve point for rG . r is not given to anyone and may be destroyed after use, unless Alice wants to later prove to a 3rd party that she paid Bob. R , however, is added to the transaction for everyone to see. A new r should be chosen for every single transaction (reusing r to send to the same recipient would result in a stealth address collision!).

Here is an example of R at chainradar.com :

One-time public key

fd4e0df2868723e0ac1ff326220974e69d9663a150f0956c1624aa41cf4fe595

Now that we have our key definitions out of the way, let's create a stealth address! I think walking through the process is the easiest way to understand.

$$P = Hs(rA)G + B$$

Referenced above we have:

1. P -- the final **stealth address** (one-time output key, the destination where funds will actually be sent);
2. Hs^* -- a **hashing algorithm** that returns a scalar (i.e., the hash output is interpreted as an integer and reduced modulo ℓ);
3. r -- the new random **scalar** Alice chose for this transaction;
4. A -- Bob's **public view key**;
5. G -- the standard Ed25519 base point;
6. B -- Bob's **public spend key**.

[*Note: the hashing algorithm of choice for Monero and other Cryptonote coins is keccak-256 . This article's focus is not on hashing algorithms, so if you want to learn more about them please see the linked article.

Wikipedia, however, has a good short list of the properties we want:

1. it is quick to compute the hash value for any given message
2. it is infeasible to generate a message from its hash value except by trying all possible messages
3. a small change to a message should change the hash value so extensively that the new hash value appears **uncorrelated** with the old hash value
4. it is infeasible to find two different messages with the **same** hash value

#1 is obvious. #2 is the one-way property. #3 is very interesting, as "uncorrelated" is quite similar to our magic word, **unlinkable**. #4, collision resistance, is very important because a bad algorithm could **significantly** "improve" the chance of a stealth address collision (versus the 2^{126} discussed above).

I would add to this list:

1. any length input;
2. fixed-length output;
3. output cannot be predicted or chosen in **advance** (preimage-resistance) -- this is related to #2 above.]

Whew, that got long.

Alright, so let's *actually* create a stealth address!

1. Alice does ECDH with her randomly-chosen r and Bob's public view key, A . Let's call this point D . No one other than Alice or Bob can compute D (see the discussion above on ECDH).
2. Alice uses D to generate a new scalar; we'll call it f . $f = Hs(D)$. *Yes I like naming things*. This is the step that actually **causes unlinkability** between Bob's outputs (remember #3 above -- more on this later)!
3. Alice computes $F = fG$.
4. Alice computes $P = F + B$ (Bob's public spend key).
5. P is the stealth destination!

Now let's look at Bob's perspective:

1. Bob receives a transaction; he wants to check if it belongs to him.
2. Bob retrieves R , which Alice has helpfully attached to the transaction.
3. Bob computes D' . Note Bob doesn't (yet) know if D' is equal to D . $D' = aR$.
4. Bob computes $f' = Hs(D')$.
5. Bob computes $F' = f'G$.
6. Bob computes $P' = F' + B$.
7. Bob checks if P' is equal to P , which was included in the transaction as the destination. If yes, Bob realizes he's been paid and does some additional steps (below). If no, Bob ignores the transaction.

Some notes:

1. Computing D and D' requires secret data: either r (Alice) or a (Bob). Thus, external observers are prevented from proceeding past Alice's step 1. Furthermore, because r is randomly chosen, even if the observer suspects Alice is sending to Bob's public address (which the observer knows), due to the ECDLP they still can't link this address to P without knowledge of r or a (or pedantically the later steps' values, namely D and f).

2. You may have noticed that this scheme only gives Alice one output for Bob per r , but with auto-denomination Monero and other Cryptonote coins have many outputs per transaction. To get different stealth addresses for each output, Alice (and Bob) append an "output index" (an output's position in the transaction: 0, 1, 2, etc.) to D before hashing it to create the secret shared scalar f . This is a bit of a clarification to Alice's step 2 on **unlinkability**. That is, while the shared secret D is already unlinkable to observers, appending an output index allows "unlimited" additional unlinkable outputs to be created from one shared secret (see point 3 in the hash section).
3. Back to the **dual-key** concept, Bob (or someone working on his behalf with knowledge of a and B) can "scan" for and detect/link outputs *without* knowledge of b , which is required below to actually spend that output. The whitepaper calls (a, B) the "tracking key".
4. It is possible to do a non-dual-key stealth addressing scheme, but you must make one of two trade-offs. You can either: (1). use the concept in the whitepaper called a **truncated address**, which means the view key pair is publicly known and all incoming transactions can be linked ($a = Hs(B)$); or (2). forego a view key pair entirely, which means scanning requires spending ability ($P = Hs(rB)G + B$).

Bob's Additional Steps

So, Bob has determined some outputs in a transaction belong to him. Now what does he do? I can imagine two things he *wants* to do: check if this output has already been spent, and (later) actually spend the output. To do either of these things, Bob must first compute the secret key associated with that output (the one-time secret key, x).

1. Bob recomputes $f' = Hs(D')$ (as above).
2. Bob derives $x = f' + b$ (b is Bob's private spend key). The "neat" thing is that $P = xG$! Adding scalars (or points) together preserves linearity. $P = xG = (f' + b)G = f'G + B$

3. To check if P is spent, Bob computes its "key image" and queries the blockchain to see if it is marked as spent. Key image $I = xHp(P)$. This will be better explained in the ring signature article. *Fear not!*
4. To spend P , Bob needs to sign a new transaction with x (also detailed in the ring signature article).

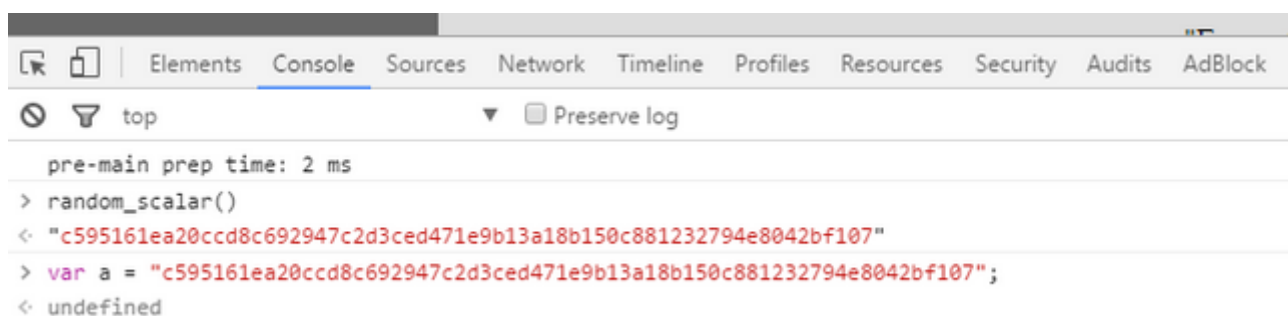
[Skip the next section if you hate fun.]

FUN

Now, let's do a "fun" exercise with real values for illustration. *Great...*

I'll be using functions that are available in Javascript here . Doing so allows the curious and discerning reader (*that's you, presumably*) to easily reproduce the results with only a web browser. If you go to the link above and open your browser's console (right-click the page->Inspect, then Console tab), you can enter or copy/paste all the commands below to "see it in action". All letters and step numbers match those above. The scalars b and r below were randomly generated with `random_scalar()`; (you obviously can't randomly generate your own if you want to reproduce my results!). You can see the contents of a variable by just typing its name and pressing <enter>. `///` is a comment in Javascript.

Here is the Chrome console:



```
pre-main prep time: 2 ms
> random_scalar()
< "c595161ea20ccd8c692947c2d3ced471e9b13a18b150c881232794e8042bf107"
> var a = "c595161ea20ccd8c692947c2d3ced471e9b13a18b150c881232794e8042bf107";
< undefined
```

Preliminary

1. `var b = "c595161ea20ccd8c692947c2d3ced471e9b13a18b150c881232794e8042bf107"; //Bob's private spend key`

```

2. var a = hash_to_scalar(b); //Bob's private view key (deterministic
   derivation); a =
   "fadf3558b700b88936113be1e5342245bd68a6b1deeb496000c4148ad4b61
   f02"

3. var B = sec_key_to_pub(b); //Bob's public spend key, this function
   multiplies base  $G$  by its input;  $B =$ 
   "3bcb82eccc13739b463b386fc1ed991386a046b478bf4864673ca0a229c3ce
   c1"

4. var A = sec_key_to_pub(a); //Bob's public view key;  $A =$ 
   "6bb8297dc3b54407ac78ffa4efa4afbe5f1806e5e41aa56ae98c2fe53032bb4
   b"

5. pubkeys_to_string(B,A); //returns Bob's public address (for curiosity
   only),
   "43tXwm6UNNvSyMdHU4Jfeg4GRgU7KEVAfHo3B5RrXYMjZMRaowr68
   y12HSo14wv2qcYqqpG1U5AHrJtBdFHKPDEA9UxK6Hy"

6. var r =
   "c91ae3053f640fcad393fb6c74ad9f064c25314c8993c5545306154e070b1
   f0f";

7. var R = sec_key_to_pub(r); //R=
   "396fc23bc389046b214087a9522c0fbd673d2f3f00ab9768f35fa52f953fef22
   "

```

Alice

```

1. var D = generate_key_derivation(A, r); //ECDH,  $rA$ ;  $D =$ 
   "a1d198629fadc698b48f33dc2e280301679ab2c75a76974fd185ba66ab8418
   cc"

2. var f = derivation_to_scalar(D, 0); //0 is the output index; the
   standard method combines these last few steps into one, but I split
   them for clarity;  $f =$ 
   "bf1d230a09bfdb0bc7fe04cddf8c1635d0ebaaf159ef85dc408ae608797525
   09"

3. var F = sec_key_to_pub(f); //F=
   "3e4b39c5b5110d6fbdb77fbcd203709c19fef28c982a86bda3f3d35fc0997
   38"

```

4. `var P = ge_add(F, B);` *//*"ge" means group element; this function adds two points together. $P =$
"6cabaac48d3b9043525a703e9e5feb72132f69ea6deca9b4acf9228beb74cd8f"

Bob

1. N/A
2. N/A
3. `var D1 = generate_key_derivation(R, a);` *//* $D1$ = same as above!
4. `var f1 = derivation_to_scalar(D1, 0);` *//* $f1$ = same as above!
5. `var F1 = sec_key_to_pub(f1);` *//* $F1$ = same as above!
6. `var P1 = ge_add(F1, B);` *//* $P1$ = same as above!

Bob's additional steps

1. N/A
2. `var x = sc_add(f1, b);` *//*"sc" means scalar; this function adds two scalars together. $x =$
"97df43cb906896405a8b54ecd4610c92b99de5090b404e5e64b17af17da01601". Now for fun enter `sec_key_to_pub(x);` and compare with P .
3. `var I = generate_key_image_2(P, x);` 9 *//*This combines "hash_to_ec" (H_p , hash to a curve point) and a scalar multiplication of that new point by x . $I =$
"2ba7ee37314d4a1edbeef727f49099c79d55797570cb1206ee2685c94b6550b1" For fun -- spent status
4. N/A

Whew!

[End skip.]



tl;dr (*You get one this time!*), stealth addressing allows senders to create "unlimited" one-time destination addresses on behalf of the recipient (without any interaction). They can only be recovered and spent by the recipient and can't be publicly linked to each other or the standard address from which they were derived.

Until next time!

[#cryptography_ \(/trending/cryptography\).](#)

[#privacy_ \(/trending/privacy\).](#)

[#anonymity_ \(/trending/anonymity\).](#)

⌚ 6 years ago in [#monero_ \(/trending/monero\)](#) by

[luigi1111 \(55\)](#) ▾ [_ \(/@luigi1111\)](#).

⬆️ ⬆️ \$994.20 ▾ | [78 votes](#) ▾

↗️ | [Reply](#) | 💬 [13](#)

[\(/monero/@luigi1111/understanding-monero-cryptography-privacy-part-2-stealth-addresses\).](#)

[f](#) [t](#) [g](#) [in](#) [@](#)

Sort: [Trending](#) ▾

trogdor (64) ▾ (/@trogdor) 6 years ago (/monero/@trogdor/re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160811t220716031z#@trogdor/re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160811t220716031z) [⌵]

This is brilliant. Your writing is amazing, and I think this will push me over the edge to finally buying some Monero.

⬆️ ⬇️ \$0.00 | 2 votes ▾ | Reply

luigi1111 (55) ▾ (/@luigi1111) 6 years ago (/monero/@trogdor/re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160811t220716031z#@luigi1111/re-trogdor-re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160812t141241979z) [⌵]

Thank you.

Good luck! :)

⬆️ ⬇️ \$0.00 | Reply

skyzalimit (31) ▾ (/@skyzalimit) 6 years ago (/monero/@trogdor/re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160811t220716031z#@skyzalimit/re-trogdor-re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160816t051313719z) [⌵]

Im with this guy..also how does one mine Monero or is it mineable

⬆️ ⬇️ \$0.00 | Reply

luigi1111 (55) ▾ (/@luigi1111) 6 years ago (/monero/@trogdor/re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160811t220716031z#@luigi1111/re-skyzalimit-re-trogdor-re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160818t172703921z) [⌵]

or is it mineable

Sure is. Check the Bitcointalk post here a little ways down:
<https://bitcointalk.org/index.php?topic=583449.0>

⬆️ ⬇️ \$0.00 | Reply

trogdor (64) ▾ (/@trogdor) 6 years ago (/monero/@trogdor/re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160811t235741841z#@trogdor/re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160811t235741841z) [⌵]

Just one question, if you don't mind. In your opinion, why is there so much talk about zcash? It seems like all the good things I've heard about zcash already exist with Monero, but the crypto for Monero is already tried and true and peer-reviewed. I guess I just don't understand what zcash is trying to do exactly that would make it better than Monero?

⬆️ ⬇️ \$0.00 | Reply

bitjedi (27) ▾ (/@bitjedi) 6 years ago (/monero/@trogdor/re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160811t235741841z#@bitjedi/re-trogdor-re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160812t054423961z) [⋮]

well, some well known angels and VCs have invested in zcash, so they are running it like a company / brand with marketing and everything. Monero like all other FLOSS projects without premines or capital raises, relies heavily on community involvement and word of mouth marketing. Bitcoin is an example of FLOSS success. Ethereum is an example of corporate success.

⬆️ ⬇️ \$0.00 3 votes ▾ Reply

lapingvino (6) ▾ (/@lapingvino) 6 years ago (/monero/@trogdor/re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160811t235741841z#@lapingvino/re-bitjedi-re-trogdor-re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160812t075329463z) [⋮]

And for that matter, corporate failure...

⬆️ ⬇️ \$0.00 2 votes ▾ Reply

ai-worries-me (25) ▾ (/@ai-worries-me) 6 years ago (/monero/@ai-worries-me/re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160812t144319432z#@ai-worries-me/re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160812t144319432z) [⋮]

Nice article! I have a question about the base point, "G":

First, we have Alice. Alice has chosen a random private key (scalar) a from our group $[1, l-1]$. Her public key is $A = aG$.

Similarly, Bob chooses random private key b . His public key is $B = bG$.

1. How do Bob & Alice agree on a base point G ?
2. If G is publicly known, and Bob & Alice's public keys are publicly known, isn't it easy to derive the private key? e.g. $A = aG$, and A and G are known, so we can easily derive a .

⬆️ ⬇️ \$0.00 Reply

luigi1111 (55) ▾ (/@luigi1111) 6 years ago (/monero/@ai-worries-me/re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160812t144319432z#@luigi1111/re-ai-worries-me-re-luigi1111-understanding-monero-cryptography-privacy-part-2-stealth-addresses-20160817t225150310z) (edited) [⋮]

This is known as the ECDLP (Elliptic Curve Discrete Logarithm) problem/assumption. Security of all elliptic curve crypto systems depend on the intractability of finding such x that $P = xG$ given P and G (at least beyond brute-forcing x 's -- curve points cannot be divided).

Consider in the case of Monero, roughly 50% of potential x 's are between 2^{251} and 2^{252} . Brute-forcing takes an impossible amount of time (and space if you want to store your results for future use); you must add G to itself over and over until you reach P . I believe some algorithms are faster than this simple brute-force, but not so significantly to cause security to be in question.

$$2.79 \times 10^{52} \text{ days}$$

Now to store these for future use you need $2^{252} * 32$ (bytes) of storage space. That's:

If each hard drive was 1mm cubed, we'd need roughly:

2.31 x 10⁴⁷ cubic kilometers of hard drives, which is roughly:

Let me know when you want to get started. ^_^

\$0.00

Reply

$$[-].$$

\$0.00

Reply

[-].

\$0.00

Reply

$$[-].$$


You can view your badges on your Steem Board and compare to others on the Steem Ranking

\$0.00

Reply

crypto-exchanges (27) ▾ (@crypto-exchanges) 9 months ago (/monero/@crypto-exchanges/qxf5tj#@crypto-exchanges/qxf5tj)

[...]

Crypto Exchange In Australia?

This is a good article on the best crypto exchange Australia (<https://steemit.com/cryptocurrency/@crypto-exchanges/best-australia>)!?



\$0.00

Reply