https://github.com/EahwW8VfYy/security-by-obscurity

Security By Obscurity

An Examination of Bitcoin Private Key Collisions

AN IMAGE THE SIZE OF A GRAIN OF SAND AT ARM'S LENGTH TAKEN BY NASA'S JAMES WEBB TELESCOPE IN 2022

Google Data Certification Course Capstone Project

Security By Obscurity

A Theoretical & Practical Examination of Bitcoin Private Key Collisions

There are many layers to the security of Bitcoin, and the most foundational layer is the use of public key cryptography[1]. Public key cryptography is a complex subject, but one important intersection between it and Bitcoin is the construction of the Bitcoin private key and how that construction secures Bitcoin wallets.

A Bitcoin private key is simply a very large number (typically 128-bit or 256-bit). And, simply choosing one very large number is the private key that secures your bitcoin. There are a few different ways to represent this number, but the most common form in practice uses the human-friendly BIP-39 collection of seed words to create a mnemonic (12 or 24 words respectively). When first learning about Bitcoin, it seems impossible that just picking a number could possibly secure every Bitcoin transaction — over $700 billion in value as of this writing. None of these numbers are hidden per se because they are a numerical set. Anyone, anywhere can choose from this defined set of numbers — any 128-bit or 256-bit number as their private key. How could the simple act of choosing a number possibly secure anything? Can't we just guess someone else's key and spend their bitcoin? Couldn't two people accidentally generate or guess the same key?

The reason this works is that in picking a Bitcoin private key, we choose from an incomprehensibly massive number of potential keys — 2^{256} if using 24 word seed phrases and 2^{128} if using 12 word phrases. Typically this is done by wallet software but it can be done manually as well with dice[2] or with paper[3]. In order to examine how a 256-bit number can provide the security of Bitcoin, let's examine the size of this number. A 256-bit number in scientific notation is about 1×10^{77} — a one with 77 zeros[4]. This is an incredibly large number, but how bit is it?

[1] https://en.wikipedia.org/wiki/Public-key_cryptography

[2] DIY Bitcoin Private Key Project by armantheparman: https://armantheparman.com/dicev2/

[3] The list of BIP-39 English words used to encode 256-bit Bitcoin private keys: https://github.com/bitcoin/bips/blob/master/bip-0039/english.txt

[4] Alternate ways 2^{256} can be expressed:

115,792,089,237,316,195,423,
570,985,008,687,907,853,269,
984,665,640,564,039,457,584,
007,913,129,639,936

115 quattuorvigintillion

approximately 10^{77}

from WolframAlpha

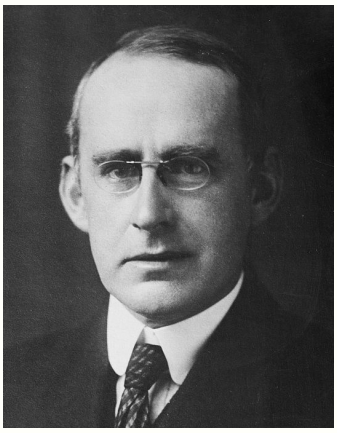
How Big is the Number 10^{77} ?

A. ANTONOPOLIS ILLUSTRATES THE MASSIVE NUMBER OF BITCOIN PRIVATE KEYS

One of the best illustrations of the size of this number in the context of Bitcoin is from Andreas Antonopolis on YouTube (above)[5]. As Antonopolis explains, 10^{77} is bigger than the number of grains of sand on earth and number of water molecules in the ocean. It's bigger than all the sand and water molecules on all rocky and watery planets in the galaxy. It's more than all the molecules in the universe! It's roughly the number of ATOMS in the known universe of all matter!

[5] Down the Rabbit Hole: Can Someone Guess My Private Key? https://www.youtube.com/watch?v=2eZ5DP2P5As

In the 1930s astrophysicist Arthur Stanley Eddington[6] calculated that there are about 10^{79} protons in the observable universe. The Eddington number has been updated more recently to 10^{80}. If we consider there are 1 to 100 protons per atom, our number of 10^{77} is about right as the number of atoms in the observable universe. Of course, when we say 'observable' that means we have no idea how big the universe actually is — but, this gives us a starting point to try to conceptualize this enormous scale.



[6] Arthur Stanley Eddington

Security by Obscurity

© WLDAVIES/GETTY

So, a 256-bit number means that basically every atom in the 'observable' universe could have a private Bitcoin key. And, the sheer magnitude of this number is what insures collisions in Bitcoin private keys are all but impossible. Yes, there is a 'chance' for a collision, but it's so vanishingly small that it can truly be considered impossible.

But, What About Very Large Numbers in Computing?

These illustrations help us try to understand the scale of large numbers. However, these are physical illustrations that may not be sufficient considering today's computing power. After all, we are discussing Bitcoin, a digital currency, in the 21st century. Of course if two people from earth try to pick the same atom in the universe, or even the same grain of sand in one beach or just from the same bucket, we know intuitively this is all but impossible — especially when considering the physical and time constraints of

this task. But, what if a super computer tries to pick your number? Computers do not have the same constraints of time and space, do they?

Modern Computing Power

Today, we have incredibly fast supercomputers that can churn through gigantic numbers of operations incredibly fast. As of 2023, the fastest computer from the TOP500 list[7] is the HP Frontier[8] at 1.102 exaFlops (10^{18} FLOPS, floating operations per second).

And, this power ranking only includes non-distributed computers. Any powerful computer can be duplicated and, given the resources, any number of them could be added to a network of distributed computing power to produce this power in parallel.

How Secure are Bitcoin Private Keys Given the Power of Modern Computing?

Let's assume we have a supercomputer like the HP Frontier and we'll call this our Exaflop Computer (EC). Let's also assume for the sake of simplicity that one floating point operation equals one Bitcoin private key guess, so that our EC can make 10^{18} guesses per second. Since one FLOP is defined as a single mathematical operation (like addition), then this estimate should theoretically be about right as we can start at 0 and just add 1 until we get to 2^{256}. Easy, right?

If we run our EC 24 hours a day, 365 days a year for a total of $3x10^7$ seconds, and our EC does 10^{18} guesses per second, we can generate $3x10^{25}$ numbers per year[9]. How long would it take to generate every Bitcoin private key? Remember, we are running through all 256-bit — 2^{256} numbers.

If we divide 2^{256} by $3x10^{25}$ numbers per year, we get an astonishing $4x10^{51}$ years! This is approximately $3x10^{41}$ times the age of the universe (estimated at 14 billion years). So, we'd have to run our EC $3x10^{41}$ times the age of the universe.

How About a Distributed Network of ECs?

As we discussed above, our EC is just one non-distributed computer. But, we can create a network of these computers — as many as we need. How many ECs would we need for our network if we want to go through all the Bitcoin private keys in one year[10]?

Since we determined that it would take $4x10^{51}$ years for one EC, we'd need that many ECs to get all the keys in one year — $4x10^{51}$ ECs working in parallel for one year. What would we need to construct such a network?

As of 2023, our single EC costs $600 M, takes up 7,300 ft^2, and consumes 21 MW of power[8]. It's safe to say, that a parallel $4x10^{51}$ distributed network of ECs would cost far more money, land, and energy than is remotely available as of today. The square footage of the earth[11] is $5x10^{15}$ ft^2, so we would need $3x10^{39}$ earths to cover with our distributed EC computer. Similarly, the total power of our sun[12] is about $4x10^{20}$ MW, so we would need to harness $2x10^{32}$ suns to run our crazy contraption.

[7] The TOP500 project ranks and details the 500 most powerful non-distributed computer systems in the world.

[8] HP Frontier supercomputer: https://en.wikipedia.org/wiki/Frontier_(supercomputer)

[9] SINGLE EC PERFORMANCE

Number of EC Guesses Per Year
$3X10^{25}$

Years to Generate Every Key
$4X10^{51}$

Number of Ages of the Universe to Generate All Keys
$3X10^{41}$

[10] DISTRIBUTED EC SPECIFICATIONS REQUIRED TO GENERATE ALL KEYS IN ONE YEAR

Cost to Build
$2X10^{60}$

Number of Earths Covered in ECs Needed
$6X10^{39}$

Number of Suns Needed for Power
$2X10^{32}$

[11] The square footage of earth: https://www.wolframalpha.com/input?i=square+footage+of+the+earth

[12] Total power output of the sun: https://www.wolframalpha.com/input?i=total+power+output+of+the+Sun+in+MW

What About Quantum Computing?

> *"Quantum computing is a new generation of technology that involves a type of computer 158 million times faster than the most sophisticated supercomputer we have in the world today. It is a device so powerful that it could do in four minutes what it would take a traditional supercomputer 10,000 years to accomplish." [13]*

[13] https://www.livescience.com/quantum-computing

Let's assume this estimate of quantum computer power is way too low and a quantum computer (QC) is 1 trillion times faster than our EC. Does this help us in any way? (Un)fortunately, all this does is take 12 zeros off of our extreme numbers. Even if we had a trillion distributed QCs we can only take 24 zeros off these numbers[14]. We'd still need 10^{17} ages of the universe to generate all the keys. We'd still need 10^{15} earths and 200 million suns assuming similar space and energy requirements of the EC.

[14] **DISTRIBUTED 1 TRILLION QC PERFORMANCE**

Number of Distributed QC Guesses Per Year
$3X10^{49}$

Years to Generate Every Key
$4X10^{27}$

Number of Ages of the Universe to Generate All Keys
$3X10^{17}$

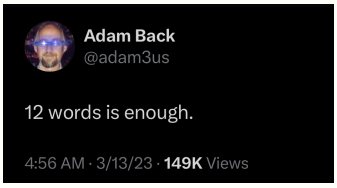
Number of Earths Covered in ECs Needed
$6X10^{15}$

Number of Suns Needed for Power
$2X10^{8}$

For all intents and purposes, the problem of deriving all Bitcoin private keys is impossible due to the scale of the numbers involved.

128-bit Seeds (12 Words) is Enough

Now we know why Bitcoin OGs will say '12 words is enough'[15] — it is! Even with 128-bit numbers, the above exponents are only halved. The numbers are still insanely large. 24 word seed phrases are exponentially more secure, but that security lies on top of security that is already more than sufficient.



[15] https://twitter.com/adam3us/status/1635218357493563392

Other Considerations

There are a couple of other considerations that might speed up or slow down the process of finding and using private keys. First, bitcoin transactions are no longer necessarily signed with single signatures. With the adoption of multi-signature transactions, multiple keys would need to be known and found and matched to a utxo to spend these transactions. Secondly, not all bitcoin keys need to be found. Starting at 0 and incrementing our key by 1 would simply find the first existing key, and that is going to happen much earlier relative to the entire set of keys. It's possible that generating only a small percentage of keys could result in a collision. Still, the numbers are so massive, even if we only needed to find the first 1%, it still doesn't make a meaningful difference. For example, if we only needed to search 1% to find a key, we would divide our Distributed 1 Trillion QC Performance numbers by 100 or remove 2 from the exponents! Such is the confounding magic of these large numbers.

Finally, a Practical Examination

So far, we've only looked at the theoretical impossibility of our computing problem, but what about the practical impossibility?

Despite the clear indication that finding a Bitcoin private key is impossible, I wanted to actually run a program that would attempt it — create a Bitcoin private key, generate hierarchical deterministic wallets, and search for any unspent transaction outputs or UTXOs on the Bitcoin blockchain. Let's get started!

Bitcoin Ingredients

For this recipe, we need the following ingredients:

1. total disregard for theoretical impossibilities
2. an ethics plan
3. Bitcoin private key generator
4. way to derive Bitcoin addresses from that key
5. copy of the Bitcoin blockchain
6. way to search the Bitcoin blockchain
7. record of our results
8. analyze results

We already have a disregard for theoretical impossibilities firmly implanted, but how about an ethics plan? At the end of the day we are searching for unspent transactions (UTXOs) for which we will have presumably found the private key — allowing us to spend the transaction. Before we bring this script to life, we should have a plan in place to deal with the extremely unlikely event that funds could be found and moved. The plan for this exercise is to move 1 satoshi of a found UTXO balance out to a temporary address and then move that amount back to the original address. This would mean no loss of funds to the owner while proving that funds had been moved. If a collision is found, it would be the luckiest discovery known in all of human history and would be reward enough without resorting to thievery.

For the scripting language, I chose Python. I used meherett's python-hdwallet library[16] to generate the Bitcoin private keys and derive addresses.

As for the Bitcoin blockchain, there were a few options. BigQuery has two copies, but they are now out of date and are no longer maintained. You can build your own queryable database of the blockchain with Abe as was done for BigQuery[17] (perhaps I will do this for part II of this project). But, ultimately, setting up and running a node is quite easy, and it can be queried by Python using the Bitcoin RPC API[18] running locally.

Although a local SQL database query would probably be the fastest way to check large numbers of UTXOs — and would also allow for searching all historical addresses in addition (scantxoutset only searches unspent UTXOs) — I found that the RPCs from Python to the local Bitcoin node were quite fast. How fast?

On an M1 MacBook with 16GB ram, running only Bitcoin core locally, each RPC call for an address took about 10 minutes to execute using scantxoutset. That seemed very slow until I started doubling the number of addresses on each iteration to test the number of addresses I could include in a single query. It turned out that Bitcoin Core RPC could handle quite a few addresses. Requests scaled asymptotically to

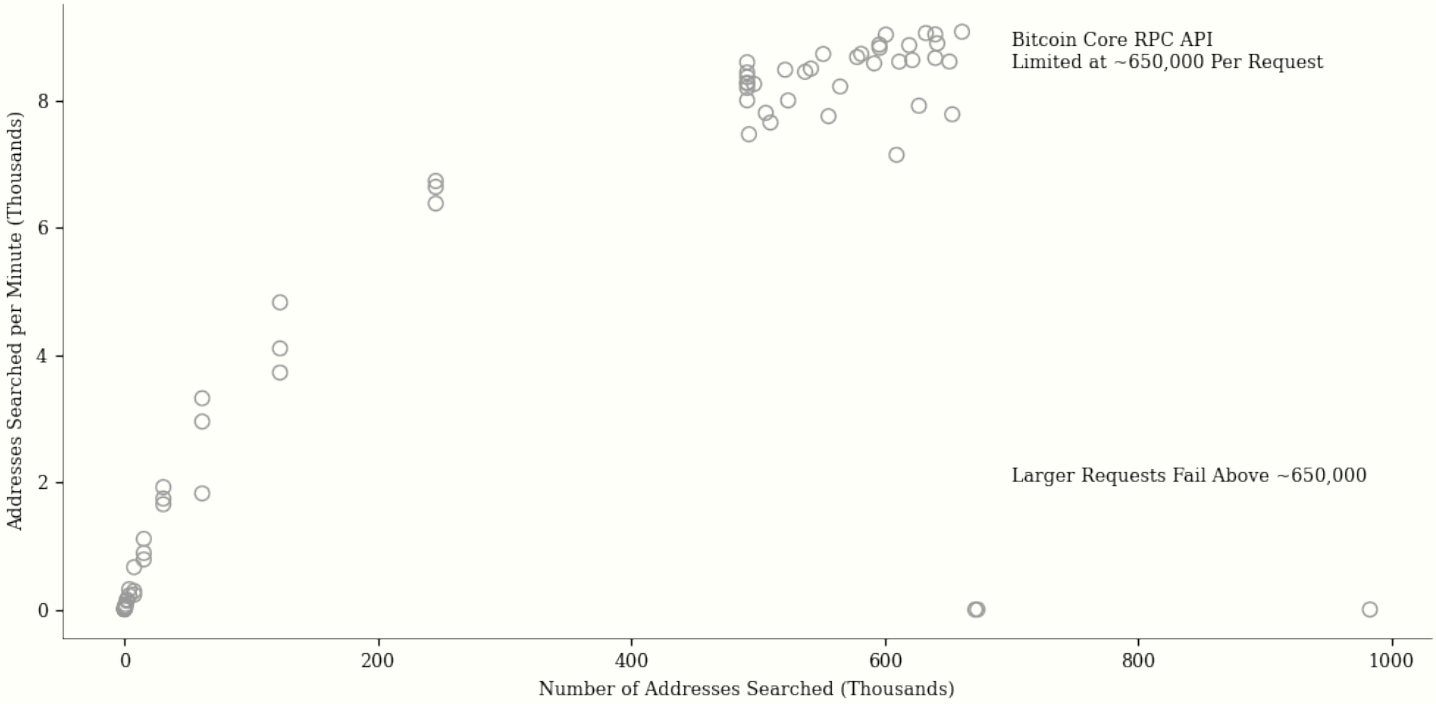
[16] https://github.com/meherett/python-hdwallet

[17] Storing and Querying Bitcoin Blockchain Using SQL Databases https://files.eric.ed.gov/fulltext/EJ1219543.pdf

[18] Bitcoin RPC API Reference https://developer.bitcoin.org/reference/rpc/

about 8,250 addresses per minute or 650,000 addresses per request until returning error "413 Request Entity Too Large".

SPEED OF THE BITCOIN CORE RPC API



I used the upper limit of 650,000 addresses as my RPC size target to maximize addresses per minute. Calls of this size to the bitcoin core API took an average of 69 minutes each.

To target the 650,000 address size, I started by generating 21,650 different bitcoin private keys with hdwallet. I then generated the first 5 addresses for each address type (p2pkh, p2sh, p2wpkh, p2wpkh_in_p2sh, p2wsh & p2wsh_in_p2sh) by iterating the derivation path for a total of 30 searchable addresses per key for a total number of 649,500 addresses. I then added a test address to my address array that I knew the balance of, allowing me to confirm the address was found and the search completed, for a grand total of 649,501 addresses. The final script does away with the test address, so that the returned Bitcoin balance can be checked for a non-zero amount.

The Bitcoin Node

There are many great tutorials on running a full Bitcoin node, so I will not go into the details here. I followed the instructions at https://bitcoin.org/en/full-node. Remember to verify file sources, signatures and hashes on any downloads. The initial blockchain sync took about a week for me on 1GB Google Fiber, and without outgoing connections enabled, peers are limited to 10. I will include the bitcoin.conf settings here because there are a couple of important things here in order to query your local node. Here are the contents of my bitcoin.conf:

```
#bitcoin.conf
datadir=/Volumes/Bitcoin # external data storage
server=1 # run bitcoin core as server
txindex=1 # enable indexing of transactions for querying
blockfilterindex=1 # index block heights & to speed up re-scanning
rpcuser=bitcoin # RPC username
rpcpassword=bitcoin # RPC password
```

```
coinstatsindex=1 # index utxos for querying

[main] # RPC bindings
rpcbind=127.0.0.1
rpcallowip=127.0.0.1
```

The txindex and RPC settings are required to be able to query the node via the Bitcoin RPC API. coinstatsindex is required to build an index of UTXOs for scantxoutset. The function used to search for wallets is scantxoutset[19] which takes an array of addresses and returns a JSON object with the UTXO set details.

[19] https://developer.bitcoin.org/reference/rpc/scantxoutset.html

Code

I then ran this script for almost three months. The code repository is on GitHub at https://github.com/EahwW8VfYy/security-by-obscurity . There is a separate script there as well that analyzes the results.csv created, and the results are as follows.

Conclusion: Practical Impossibility Follows Theoretical Impossibility

Because of the results from our thought experiment regarding 256-bit numbers, we expected to churn through millions of keys and hundreds of millions of addresses and come up with nothing. And, our hypothesis is correct! After running for more than 100 days here are the results:

```
---------------------
Summary Stats
---------------------

total number of addresses: 809,806,768
total number of mnemonics: 26,983,409
in total time: 102 days
mean request time, last 200 requests: 75 minutes
```

And, the big reveal.. did we find any wallets? If a key generated a wallet that had a utxo balance, we would have found it, and the balance for that search would be different from the test wallet address:

```
---------------------
Results different from test wallet balance
(indicating a wallet has been found):
---------------------

No wallets have been found :(
```

I also wanted to check for duplicate keys/mnemonics. Is there any way we could have generated a duplicate private key? Comparing all mnemonics resulted in zero duplicates — as expected:

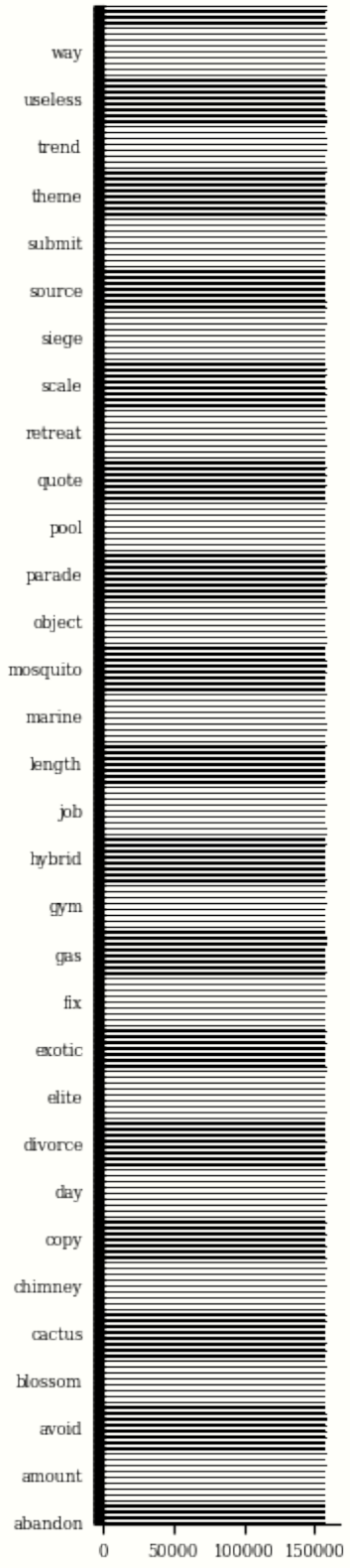
```
---------------------
Any Duplicated Keys?
---------------------

no duplicates were found in 26,983,409 mnemonics
```

Distribution of BIP-39 Words in Seed Phrases

Since we recorded all the seed phrases, I wondered what the distribution of seed words looked like. If the seed/key generation was truly random, there should be a pretty even distribution of seed word usage. While this is just a reflection of hdwallet's randomization functions, I thought it would still be interesting to explore to get a sense of whether our results are based on a truly random dispersion of private Bitcoin mnemonic words. According to random.org, one of the simplest ways to see if numbers are random is to look at them visually[20]. Here, the frequency of each word is plotted in a horizontal bar graph[21] — 323,800,908 words in total. Every 64th word is used on the y-axis to free up space, but all 2048 BIP-39 words are here, and it looks pretty consistent. But, what about a statistical analysis of this randomness?

[20] https://www.random.org/analysis/

[21] **FREQUENCY OF BIP-39 WORDS IN SEED PHRASES**



Random Number Testing

I used the count output of the BIP-39 words from the seed phrases and tested the randomness of this series using stevenang's randomness_testsuite[22], a python-based NIST Randomness Test Suite. Results from non-zero P-Values seem to confirm a good amount of randomness. Here are the test results:

```
Test Data File:bip39words.csv

Type of Test                                    P-Value                 Conclusion
01. Frequency Test (Monobit)                    0.0                     Non-Random
02. Frequency Test within a Block               3.50613425080008e-303   Non-Random
03. Run Test                                    0.0                     Non-Random
04. Longest Run of Ones in a Block              0.0                     Non-Random
05. Binary Matrix Rank Test                     7.14315879175677e-136   Non-Random
06. Discrete Fourier Transform (Spectral) Test  0.0                     Non-Random
07. Non-Overlapping Template Matching Test      0.36581532439133885     Random
08. Overlapping Template Matching Test          2.0210990582707992e-33  Non-Random
09. Maurer's Universal Statistical test         -1.0                    Non-Random
10. Linear Complexity Test                      0.9801623012013757      Random
11. Serial test:

                                                0.0                     Non-Random
                                                0.0                     Non-Random
12. Approximate Entropy Test                    0.0                     Non-Random
13. Cummulative Sums (Forward) Test             0.0                     Non-Random
14. Cummulative Sums (Reverse) Test             0.0                     Non-Random
15. Random Excursions Test:
        State           Chi Squared             P-Value                 Conclusion
        -4              46.33819241982507       7.750306806822414e-09   Non-Random
        -3              4.8208                  0.4381393147182503      Random
        -2              12.320987654320987      0.03064461946854812     Random
        -1              18.0                    0.0029464045878802923   Non-Random
        +1              2.0                     0.8491450360846096      Random
        +2              0.6666666666666666      0.984747879018509       Random
        +3              0.4                     0.9953295932358704      Random
        +4              0.2857142857142857      0.9979031595107858      Random
16. Random Excursions Variant Test:
        State           COUNTS                  P-Value                 Conclusion
        -9.0            3                       0.9034789137192296      Random
        -8.0            2                       1.0                     Random
        -7.0            4                       0.7815112949987133      Random
        -6.0            4                       0.763024600552995       Random
        -5.0            2                       1.0                     Random
        -4.0            4                       0.7054569861112734      Random
        -3.0            9                       0.11752486809663916     Random
        -2.0            10                      0.020921355337794014    Random
        -1.0            6                       0.04550026389635843     Random
```

Conclusion

A Bitcoin private key is a number so incomprehensibly large that the most practical way to describe the possibility of key collisions is: 'impossible'. The numbers are far too big for a collision to occur if the keys are sufficiently random — even with foreseeable advances in modern computing. Bitcoin has Security by Obscurity — it is secured by the truly staggering obscurity that large numbers provide for in the construction of the Bitcoin private key.

[22] https://github.com/stevenang/randomness_testsuite