Why You Can't Just Brute-Force a Bitcoin Private Key

Author	josh
Date	2019-12-07 14:12:48

Overview

Unfortunately, sometimes Bitcoin private keys are lost. Users destroy wallets, throw away hard drives, or simply never backup keys in the first place. A seemingly obvious solution to the problem of a lost key might be to try and "guess" all the possible keys until you find the one that unlocks your addresses. One can just spin up their gaming laptop, or maybe some Azure VM's to get this done, right?

Well, no. In fact, there's no practical chance *at all* that you could ever brute-force a Bitcoin private key. The scale of the problem is far larger than we as humans can even appreciate. Let's dive in to the numbers and show why cracking your lost paper wallet simply isn't going to happen.

Bits and Brute-Force: Understanding Key Cracking

A "Bit" on Binary Numbers

In order to understand the scale of the numbers involved in Bitcoin private keys, one must first understand a little bit about the binary number system used by computers. We're used to a base 10 system, where each "place" in a number can have digits 0-9. The number 100, for example, has a 0 in the one's place, a 0 in the 10's place, and a 1 in the 100's place. The number 100 when represented as a power of 10 is 10^2, or ten squared.

When it comes to binary numbers, similar principles apply. But force each "place", we can only have 0 or 1. On or off. The number 18. in binary, for example, is 10010. There's a 0 in the 1's place, 2 in the 2's place, 0 in the 4's place, 9 in the 8's place, and finally, 1 in the 16's place. Each place is a power of 2! 10000 in binary is 16, or 2^4 - two to the fourth power.

The Scale of the Problem

Now this concept helps us understand a bit more about the scale of a particular *keyspace.* Let's say, for example, we have an 8-bit private key for a cryptosystem. 8 bits, and each bit can be either 1 or 0. This means there's a total of 2^8 *possible combinations* of 1's and 0's, giving us 2^8 *= 256 possible private keys.* If we have a 16-bit private key, we have 2^16 = *65536 possible private keys*. You can apply this formula to any size keyspace you'd like.

You may have noticed something from our small sample size here. We only added 8 bits to the keysize going from 8 to 16 bits. However, we went from 256 possible keys to over 65 thousand! That's a lot more possible keys, and therefore a lot more possible guesses we'd have to make to brute force the keyspace. It turns out that adding bits to a keysize makes it *exponentially* harder to brute-force that particular keyspace, and that's why even a seemingly small keysize can make it practically impossible to brute-force those keys.

So, what about Bitcoin? Bitcoin uses *256-bit* private keys. So given our little bit of math here, we can calculate the number of possible combinations:

2^256 = *115792089237316195423570985008687907853269984665640564039457584007913129639936* or *1.157921 x 10^77*

Now that is an *unfathomably* large number of possible keys. For scale, the number of keys available in the space is on par with the number of atoms thought to be in the observable universe. That's a lot of keys to guess.

A More Hands-On Look at the Scale of the Problem

Now one can stare at that large number of possible keys in awe, and still not quite understand how impossible it is to brute force Bitcoin keys. Computers are pretty good at dealing with large numbers, right? Yes, but still not good enough to deal with this many guesses.

I've created a program called *PkTime* written in C. This program will show us how long it takes to brute-force various keysizes on the machine it's run on, or it can be given the number of ops (guesses) per second a machine is capable of. And of course, it's available under a free and open source license!

Pktime will calculate the actual time to brute-force Bitcoin keys when run on the machine without a given ops/second value. It uses Trezor's libraries to generate an actual Bitcoin Keypair, so it more accurately reflects the time to check one keypair in a brute-force attack.

This is the output of PkTime on my personal machine, an Asus laptop with an intel i5 processor. The program only uses a single core (no multithreading), and you'll see why it makes no difference anyway:

```
josh@Josh-Asus:~/pktime/bin$ ./pktime
Calculating some real and estimated brute force times...please wait
Average optime for one keypair check: 0.00000959
Real time to bruteforce 8 bit keysize: 0.004566 seconds with 256 iterations
Real time to bruteforce 12 bit keysize: 0.035437 seconds with 4096 iterations
Real time to bruteforce 16 bit keysize: 0.457779 seconds with 65536 iterations
Real time to bruteforce 18 bit keysize: 1.902443 seconds with 262144
iterations
Real time to bruteforce 20 bit keysize: 7.585260 seconds with 1048576
iterations
Est. time to bruteforce 24 bit keysize: 2.68233088 minutes with 16777216
iterations
Est. time to bruteforce 28 bit keysize: 42.91729408 minutes with 268435456
iterations
Est. time to bruteforce 32 bit keysize: 11.44461175 hours with 4294967296
iterations
Est. time to bruteforce 36 bit keysize: 7.62974117 days with 68719476736
iterations
```

```
Est. time to bruteforce 40 bit keysize: 122.07585872 days with 1099511627776
iterations
Est. time to bruteforce 44 bit keysize: 5.34760777 years with 1.759219E+13
iterations
Est. time to bruteforce 48 bit keysize: 85.56172438 years with 2.814750E+14
iterations
Est. time to bruteforce 52 bit keysize: 1368.98759015 years with 4.503600E+15
iterations
Est. time to bruteforce 64 bit keysize: 5.607373E+06 years with 1.844674E+19
iterations
Est. time to bruteforce 128 bit keysize: 1.034378E+26 years with 3.402824E+38
iterations
Est. time to bruteforce 192 bit keysize: 1.908090E+45 years with 6.277102E+57
iterations
Est. time to bruteforce 256 bit keysize: 3.519805E+64 years with 1.157921E+77
iterations
```

It's amazing to see just how much longer it takes to brute-force a keyspace by adding just a few bits. The jump from 28 to 36 bits takes the problem from minutes to *days*, and the jump from 40 to just 44 bits makes the difference between days and *years*. This is exponentially scaling.

A look at where we are for a 256-bit Bitcoin key...over *3 x 10^64 years*. Trillions upon trillions upon trillions of years. Even a 50-some bit key is impossible to exhaust in any meaningful amount of time on a modern laptop.

Now you're probably thinking - that's a laptop though...what about the world's best supercomputers, governments, etc. No need to fear for your keys; exponentially scaling still protects them!

Let's re-run our estimation program and give an ops/second for the world's fastest supercomputer. top500.org list's IBM's Summit as the current leader, with a theoretical maximum of 200795000000000000 flops, or floating-point operations per second. Let's *generously* assume we have a supercomputer with this speed, and that it only takes 1 op to check a key (it takes quite a few more).

```
josh@Josh-Asus:~/pktime/bin$ ./pktime 200795000000000000
Average optime for one keypair check: 0.00000000
Est. time to bruteforce 24 bit keysize: 0.00000000 seconds with 16777216
iterations
Est. time to bruteforce 28 bit keysize: 0.00000000 seconds with 268435456
iterations
Est. time to bruteforce 32 bit keysize: 0.00000002 seconds with 4294967296
iterations
Est. time to bruteforce 36 bit keysize: 0.00000034 seconds with 68719476736
iterations
Est. time to bruteforce 40 bit keysize: 0.00000548 seconds with 1099511627776
iterations
Est. time to bruteforce 44 bit keysize: 0.00008761 seconds with 17592186044416
iterations
Est. time to bruteforce 48 bit keysize: 0.00140180 seconds with
281474976710656 iterations
Est. time to bruteforce 52 bit keysize: 0.02242884 seconds with
4503599627370496 iterations
```

```
Est. time to bruteforce 64 bit keysize: 1.53114238 minutes with
18446744073709551616 iterations
Est. time to bruteforce 128 bit keysize: 5.370103E+13 years with 3.402824E+38
iterations
Est. time to bruteforce 192 bit keysize: 9.906091E+32 years with 6.277102E+57
iterations
Est. time to bruteforce 256 bit keysize: 1.827351E+52 years with 1.157921E+77
iterations
```

Our supercomputer here is orders of magnitude faster than a laptop - in fact, it renders 64-bit keys completely insecure. But look how, once again, exponential scaling keeps 256-bit keys very, very safe. It would still take over 1 x 10^52 years to exhaust all the possible Bitcoin keys. Given that the universe is roughly 13 x 10^9 billion years old, I think it is safe to assume that brute-forcing the keys are an *impossible* task.

One last note - even if we could build a theoretical computer that could guess much faster, there are *energy limits* to computation. The amount of energy it would take to do all these operations far exceeds what humanity could harvest.

Brute-Forcing Bitcoin - Not Gonna Happen!

Given the enormity of the 256-bit keyspace, the speed of the world's most amazing computers, and the limits of computation itself - you can clearly assume your Bitcoin keys are safe. As with all things security, there are other attack vectors that can get your coins stolen. But, there's no way someone is spinning up a cloud computer to try and brute-force keys. The probabilities are far too low of finding a key, and the cost is simply too high. If you wanna steal those Bitcoin keys...just crack open a cold one and wait until the end of the universe. Even then, you might run out of time!