All kinds of online services get hacked. This includes services that you might be using. [Scattered Secrets](https://scatteredsecrets.com/) is a password breach notification and prevention service. We continuously collect publicly available hacked databases and try to crack the corresponding passwords. Verified account owners can access their own information and take appropriate action to keep their accounts safe and prevent against account takeovers. At the time of writing, our database includes nearly four billion — yes, that is with a B — plaintext passwords. Users occasionally ask us how we can crack passwords on such a large scale. To answer this, first we need to look at the basics.

Theory

***Hashes***In the old days, passwords were stored in plain text. It didn’t take long to figure out that this was not a good idea: a breach of the system resulted in hackers having instant access to all accounts. To prevent this, one-way hashing was introduced. A variable length password is transformed in a fixed length code (‘hash’) and stored. When a user authenticates, the entered password is hashed too. If the stored and calculated hash match, the entered password is correct. If the system is breached, the passwords of the system are still safe: password hashes cannot be decoded or reversed to retrieve the plain passwords.

***Hash cracking***Although the password hash cannot be decoded or reversed, an attacker can still try to recover the plain password. If the hash algorithm is [secure](https://en.wikipedia.org/wiki/Cryptographic_hash_function#Attacks_on_cryptographic_hash_algorithms), the only way to retrieve the plain password is mimicking the process that is used by the system itself: enter a candidate password, calculate the candidate hash value and compare this candidate hash value with the target hash value. If they match, the plain password is known. For generating candidate passwords, several approaches can be used. Probably the most used approaches are ‘dictionary’, ‘brute force’ and ‘hybrid’ attacks. In a dictionary attack, a list of commonly used password is used (password, welcome, 123456 etc.). In many cases this approach is very effective, based on the fact that most users use commonly used words as a password. If a dictionary attack was unsuccessful, the next step can be a brute force attack. For this type of attack, all possible combinations are tried (e.g. from 00000000 to ZZZZZZZZ). This takes longer but allows an attacker to recover other categories of passwords, including randomly generated passwords. A hybrid attack is a combination of a dictionary attack and a brute force attack, allowing an attacker to find variations of commonly used passwords (e.g. 123Password, Welcome2019!).

***Feasibility of attacks***To perform a password cracking attack, a number of ingredients is required. First an attacker needs to have access to the password hashes. For now, we assume that the attacker found a way to achieve this: every single day websites get hacked. Second, the password hashing algorithm needs to be known. In most cases, standard well-known algorithms are used. If not, an attacker can extract the algorithm from the application code of a breached site. Third, an attacker needs a strategy to crack passwords (dictionary, brute force, hybrid etc.). A good strategy takes two things into account: the cracking speed for the given algorithm, and the number of possible passwords that will be checked — also known as the ‘key space’. The key space depends on the character set of the password — for example digits only or mixed-case alphanumeric — and the length of the password. The key space grows exponentially, based on a simple formula: key space = character set ^ length.

Image for post

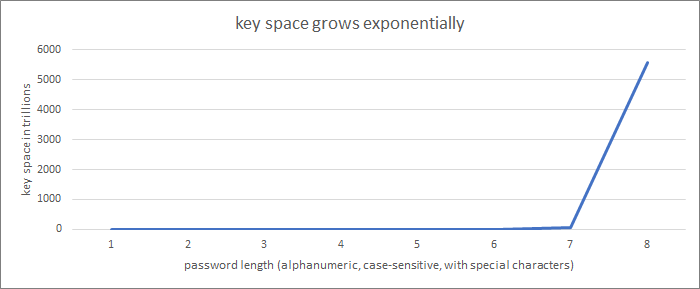


Figure 1: exponential growth (93^length)

Exponential growth means for example that cracking an 8-character password does not take twice as much time as cracking as a 4-character password. Instead it means that an 8-character alphanumeric mixed-case password using special characters (10 digits + 26 lowercase + 26 uppercase + 31 specials = 93) takes 93^8 / 93^4 = almost 75 million times more time to crack. That is 1 second versus 865 days.

The time needed to perform an attack can be calculated using time = key space / cracking speed. Cracking speed — and thus attack time — depends on the speed of a password hashing algorithm. Based on the key space and cracking speed for a specific algorithm, an attacker can determine which approach is feasible (cracking for days, weeks, months) and which is not (years, centuries, forever).

***Slowing down attacks***From a defender’s perspective, it is good to use a slow hashing algorithm. This does not hurt user experience — e.g. 1 versus 100 milliseconds for authenticating — and at the same time slows password cracking attacks down — in this example with a factor of 100. A slowdown can be achieved by increasing the key space, decreasing cracking speeds and by introducing hardware specific limitations.

* Key space can be increased by using larger character sets and longer passwords. This is why you need to use long passwords with digits, uppercase, lowercase and special characters on many sites. Key space can also be increased by using a technique called ‘[salting](https://en.wikipedia.org/wiki/Salt_(cryptography))’. Instead of hashing the plain password only, the password and a random ‘salt’ are hashed. For example, the password hash is not created as hash(password)but as hash(password + salt), where salt is a large random value. There is no impact on the key space for a single hash. However, for N hashes, the amount of time required for a given cracking strategy increases with a factor N.
* Cracking speed can be decreased by using slower hashing algorithms. A simple way of achieving this is using N iterations of an algorithm. This slows cracking down with a factor N. This technique is known as ‘[key stretching](https://en.wikipedia.org/wiki/Key_stretching)’.
* Hardware-specific slowdowns typically exhaust fast memory by using relatively large amounts of it. This will result in a significant drop in performance on the targeted platform, since the system needs time to access slower memory. Introduced slowdowns are platform and processor-specific.

Many implementations use a combination of salting and key stretching. Well-known 1970’s and 1990’s examples are [crypt](https://spqr.eecs.umich.edu/courses/cs660sp11/papers/10.1.1.128.1635.pdf)— 25 times DES — and [MD5crypt](http://phk.freebsd.dk/sagas/md5crypt/)— 1,000 times MD5. Both were used in many operating systems at the time. Later examples include [WPA-Personal](http://jorisvr.nl/wpapsk.html) — used in most WLAN routers nowadays — using salting and 4,096 iterations of HMAC-SHA1.

The number of password hashing algorithms that use all three mechanisms to slow down cracking is limited. Probably the first well-known widely used algorithm is [bcrypt](https://www.usenix.org/legacy/events/usenix99/provos/provos_html/node1.html), first presented in 1999 and used in a number of modern operating systems. Bcrypt uses salting, a configurable number of iterations *and*relatively large amounts of memory. More recent developments include [scrypt](http://www.tarsnap.com/scrypt.html)and [Argon2](https://password-hashing.net/).

***Examples***What do the different options mean if we want to crack millions of alphanumeric mixed-case passwords using special characters¹? Let us do the math for trying to crack 10 million hashes using a de-facto standard password cracking device², a top-end gaming graphics card . What result can we achieve using one-month of 24/7 cracking, targeting different types of hashes?

Image for post

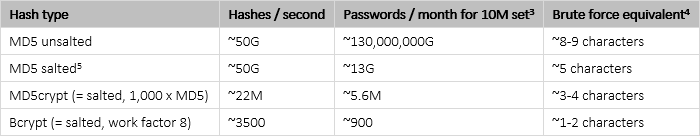


Figure 2: cracking speeds for various types of hashes

*¹ Possibilities per character = 10 digits + 26 lowercase + 26 uppercase + 31 specials = 93.*

*² Based on hashcat benchmarks on a*[*Nvidia RTX 2080 Ti*](https://www.nvidia.com/geforce/graphics-cards/rtx-2080-ti/)*.*

*³ Hashes per second x 86,400 (seconds per day) x 30 (days per month) / number of salts.*

*⁴ log93(passwords per month).*

*⁵ For all salts of all algorithms, we assume that salt entropy is high enough to not have a significant number of duplicate salts.*

The numbers of passwords we can check in a month varies enormously amongst the target hash sets. An 8-character randomly generated password is not by definition secure, and a 5-character dictionary word is not by definition insecure. It is all about the algorithm that is used to store the password.

In this example, one cracking device is used for all tests. This is not an ideal situation. By picking specific cracking platforms for specific hash types, things can be sped up or made more power efficient.

Cracking platforms

There are four different type of platforms for cracking password hashes. Each one has specific characteristics.

***Central Processing Units***A CPU is a generic processor, the component of a computer that performs the basic operations of the system, like running programs and interfacing with peripherals. CPU-based high-performance password crackers are publicly available [since the early 1990’s](http://www.ouah.org/maximumsecurity/docs/v0000011.htm). Using the CPU as the workhorse is an obvious choice: every PC uses one (or more), so every PC can be used as a password cracker. Current CPUs have up to 32 cores, which can be used in parallel. Current open source crackers like [John the Ripper](https://www.openwall.com/john/) include parallelization, advanced processor specific optimizations and support a large number of hash types. Writing code for a CPU is relatively simple. Prices of high-end CPUs start at around a 1,000 Euros.

***Graphic Processing Units***A GPU is a specialized processor, originally used primarily for 3D game rendering. GPU-based high-performance password crackers became available after Nvidia published its [CUDA](https://developer.nvidia.com/cuda-toolkit)(Compute Unified Device Architecture) framework and APIs in 2007, allowing programmers to use the full potential of GPUs. For most algorithms, a GPU is significantly faster than a CPU. The reason for this is simple: current GPUs have many thousands of cores. Since the process of password cracking is ideal for parallelization — it can be divided in independent subtasks — GPUs and password hash cracking are an excellent match. Current open source crackers like [hashcat](https://hashcat.net/hashcat/)include support for most of the current GPUs and supports a large number of hash types. Writing code for a GPU is relatively simple, although more time consuming than writing code for a CPU. Prices of top-end gaming GPUs start at about 1,200 Euros, high-end professional GPUs cost about a tenfold.

***Field Programmable Gate Arrays***FPGAs are chips that can be programmed for a special application after manufacturing. Unlike a CPU or GPU, FPGAs do not run code. Instead an FPGA *is*the code, build in hardware. FPGAs can be reprogrammed, so functionality can be updated or changed after initial programming. The first FPGA-based high-performance crackers were [built in the late 1990’s](https://link.springer.com/content/pdf/10.1007%2F3-540-48892-8_19.pdf) although they were built for cracking keys, not for cracking password hashes. The [first commercial FPGA-based crackers](https://www.copacobana.org/) were available in the mid 2000’s for for ‘[less than US$ 10,000](https://www.copacobana.org/paper/copacobana_CHES2006.pdf)’. The only known open source password cracker with FPGA-support is John the Ripper, [supporting seven FPGA-enabled algorithms](https://www.openwall.com/lists/announce/2019/05/14/1). Speed for most algorithms is somewhere in between CPUs and GPUs, at significantly lower power usage. Writing code for an FPGA is complex. Prices of descent speed FPGA boards start at around 500 Euros, boards with a high-end FPGA start at around 3,500 Euros.

***Application Specific Integrated Circuits***ASICs are microchips designed for a special application. Like an FPGA, an ASIC *is*the code. Unlike an FPGA, an ASIC cannot be reprogrammed. The first ASIC-based high-performance crackers were [built in the late 1990’s](https://web.archive.org/web/19981203120621/http:/www.eff.org/descracker/), also for cracking keys. Compared to an FPGA, an ASIC is typically significantly faster, uses less power and is cheaper at large volumes. There are no ASIC-based password crackers available on the market. Reason for this are the limited use-cases for an ASIC — a single algorithm per chip design — and very high initial development costs. Initial costs for more or less comparable ASIC projects are [estimated](https://bitcointalk.org/index.php?topic=181192.0) between hundreds of thousands and millions of Euros. This however does not seem to stop governments from [building ASIC-based crackers](https://theintercept.com/2017/05/11/nyu-accidentally-exposed-military-code-breaking-computer-project-to-entire-internet/).

Finding the right tool for the job

With theoretical limitations and available cracking platforms in mind we can pick the right tools for the job. Password hashes can be divided in two categories: hashes with and without hardware-specific slowdowns. Let us call them ‘simple’ and ‘advanced’ hashes.

***Cracking simple hashes***With their enormous number of cores, GPUs are the best choice for cracking standard simple hashes, with or without salts and key stretching. With a single high-end GPU, up to [tens of billions of hashes can be calculated per second](https://gist.github.com/shahril96/168975e2bc8de49492180ae4f1721294). For standard algorithms like MD5 and SHA-1, a GPU outperforms a CPU by a factor 50 to 100.

A CPU might still be a good choice for non-standard simple hashes. Non-standard simple hashes are typically not implemented in GPU-based cracking tools and used by one or a very limited number of companies and organizations only. Fast implementation of the algorithm — developing code for a CPU is simpler in our experience — and relatively slow cracking speed on a CPU-based system typically takes less time than slower development and faster cracking on GPUs.

FPGAs can be very [power efficient](https://www.usenix.org/system/files/conference/woot14/woot14-malvoni.pdf) but are more expensive and have a [hard time](http://publications.jrc.ec.europa.eu/repository/bitstream/JRC91528/lbna26833enn.pdf) to match the raw computing power of GPUs. Unless your budget is virtually unlimited, FPGAs are not interesting for cracking simple hashes.

***Cracking advanced hashes***Advanced hashes typically use relatively large amounts of memory. For example, [bcrypt uses over 4 kilobyte](https://en.wikipedia.org/wiki/Bcrypt). To crack advanced hashes efficiently, the algorithm needs to run from very fast memory. Otherwise, the computing cores need to wait for slower memory. If a core is waiting, it is not cracking, resulting in a significant performance drop.

Algorithms of advanced hashes easily fit in the fast level 1 (‘L1’) cache memory of a CPU core, so cracking runs efficiently. For GPUs it’s a different story: the small GPU L1 caches cannot contain the full algorithm. Where the GPU outperforms a CPU on standard hashes with a factor of 50 to 100, for example CPU bcrypt performance is within the same order of magnitude, with similar power usage. Multi-socket many-core CPU systems can be a good choice for cracking advanced hashes. Especially since the multi-CPU based systems take less rack space than multi-GPU based systems and can run with more internal memory — hundreds versus 8 to 16 gigabytes per device — allowing crackers to process larger datasets at once.

FPGAs can be programmed *as*the target algorithm, in hardware. This means that the exact amount of fast memory is available, or that specific functions are implemented as hardware. Furthermore, many implementations of the algorithm can run in parallel in a single FPGA chip. Running bcrypt on the current open source FPGA implementation outperforms a top-end gaming GPU with a factor of about 4 to 5 (one GPU versus one FPGA board with four previous generation FPGAs). Besides being significantly faster, FPGA implementations use just a fraction of the power of a GPU for similar performance: about 10 Watts for the equivalent of about 275 Watts of GPU power. Especially multi-FPGA systems can be an excellent choice for cracking advanced hashes. This is true for cracking power, cracking power per server *and*cracking power per Watt.

Cracking at ScatteredSecrets.com

Let us go back to the original question: how to crack billions of passwords? We have seen that simply buying more generic computing power is not the best solution. Instead, at Scattered Secrets we focus on smart cracking strategies and hardware that excels in a specific job. As a result, we run a wide variety of cracking hardware. For simple standard hashes we mainly use multi-GPU systems. For simple non-standard hashes we mainly use multi-CPU systems. For power and rack space efficiency, we typically combine massive GPU and CPU power in a single system:

Image for post



Figure 3: Scattered Secrets’s cracker category ‘heavy’: four GPUs and four CPUs in a single system

For advanced hashes we use both multi-CPU and multi-FPGA systems:

Image for post

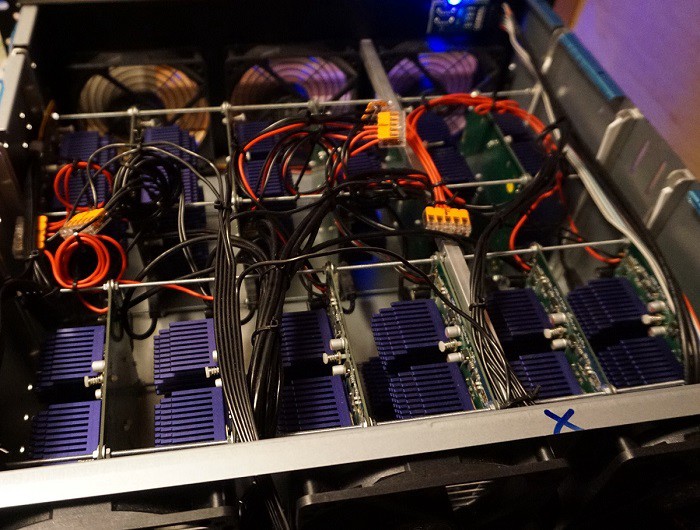


Figure 4: Scattered Secrets’s cracker category ‘FPGA’: under the hood of a forty-eight FPGA system

Cracking strategies vary as well, based on the effective speed for (extremely) large datasets. Brute force, dictionary attacks and advanced rulesets are used for fast hashes. Advanced dictionary attacks and basic rules are used for medium speed salted hashes. Simple dictionary attacks and simple rules are used for slow hashes. Feedback loops make sure that the process is getting more effective and efficient after each iteration: we continuously provide our customers with more and more complex cracked passwords. Scattered Secrets ♥ passwords ;) Don’t forget to [check if your passwords are breached](https://scatteredsecrets.com/)!