hashcat 8.0.0 - Release Notes

The hashcat development team is excited to announce the release of hashcat 8.0.0.

This is the largest update in the history of hashcat. No earlier release changed as many files (4,091), and no earlier release collected as many changelog entries (748). It brings 11 major features, 4 new attack-modes, 146 new hash-modes and 306 bug fixes, with contributions from 63 people, 43 of whom are first-time contributors.

Among the 11 major features in this release, 3 stand out:

- **PCFG Attacks.** Probabilistic attacks that try the most likely passwords first. Hashcat can seek anywhere in their keyspace, so they finally work in distributed setups
- **Hash Recipes.** Describe a custom hash on the command line and crack it at GPU speed, without waiting for someone to write a hash-mode for it
- **FPGA Support.** For the first time, hashcat runs on FPGAs, with initial support for bcrypt on 7 types of boards

The numbers below compare this release with v7.1.2. The first table counts what changed in hashcat itself, and the second what happened around it on GitHub.

hashcat v8.0.0	Since v7.1.2
Major features	11
Refactorings	5
New attack-modes	4
New hash-modes	146
New hash primitives	26
Fixed bugs	306
Performance improvements	83
Changelog entries	748
Lines added	439,469
Lines deleted	229,313

On GitHub	Since v7.1.2
New contributors	43
Returning contributors	20
Issues closed	214
Issues opened	141
Pull requests merged	364

> If you are upgrading an existing setup, please read section 5, "Before You Upgrade", first.

1. **New Attacks**
    - 1.1 PCFG, Attack-Mode 4
    - 1.2 Account-Aware Association, Attack-Mode 9
    - 1.3 Table Attacks, Attack-Mode 5
    - 1.4 Wordlists in Masks, Attack-Mode 12
    - 1.5 Rules in Every Attack-Mode
    - 1.6 Feed Plugins, Attack-Mode 8
    - 1.7 Lookup: Where an Attack Reaches a Password
2. **Hashes Without a Hash-Mode**
    - 2.1 Hash Recipes, Hash-Mode 4000
    - 2.2 Python Bridge
    - 2.3 Rust Bridge
3. **FPGA Support**
4. **Large-Scale and Distributed Work**
    - 4.1 Seekable Wordlists
    - 4.2 Many Wordlists, One Keyspace
    - 4.3 Lookup in Distributed Attacks
    - 4.4 Brain Server
    - 4.5 Large Hash Lists
    - 4.6 Shared Machines and Storage
5. **Before You Upgrade**
6. **New Algorithms**
7. **Performance**
8. **User Experience and Platforms**
9. **Bug Fixes**
10. **Under the Hood**
11. **Plans for the Next Updates**
12. **List of Contributors**
13. **Final Words**

A Word on AI

Many of the changes in this release were written with the help of AI coding agents, and we think that is a good thing. AI-generated code has reached a level of quality that makes a real difference, and more and more contributors use it for their pull requests. As maintainers, there is no way around using AI ourselves to review them. Every pull request is now checked by an AI agent of our own, at its highest reasoning setting.

To keep contributions from AI agents easy to review, we added `AGENTS.md` to the repository. It sets out for an agent what a human contributor already knows. This includes how to build and test, and the fact that a kernel which compiles is not yet a kernel that works. It also covers the requirement that every output in a pull request has to come from a real run, and the value of keeping a pull request short. Contributions written with an AI agent are welcome, as long as they follow `AGENTS.md` and `CONTRIBUTING.md`.

1. New Attacks

In this section, we highlight the new strategies hashcat uses to generate password candidates. Hashcat v8.0.0 adds 4 new attack-modes and extends the existing association attack. Under the hood, all of this is built on one new design: the feed plugins. We explain how they work at the end of this section.

1.1. PCFG, Attack-Mode 4

PCFG stands for Probabilistic Context-Free Grammar. A PCFG attack generates password candidates from a model trained on real passwords, and it starts with the most likely ones. The trained model is called a ruleset. This is the term the PCFG tools use, and we keep it to stay compatible with them. A PCFG ruleset has nothing to do with hashcat's rules that you load with `-r`.

PCFG attacks have been out of reach for hashcat for a long time. The problem was that there was no way to seek in a PCFG keyspace. Hashcat needs that ability to split an attack across devices, to restore a session, or to hand out work in a distributed setup. The only options were to pipe the candidates into hashcat via stdin, or to run PCFG as a slow candidate generator.

With hashcat v8.0.0, this changes. Hashcat counts the candidates of a grammar without generating them, and it can jump to any position in the keyspace. As a result, everything around the attack works as you would expect:

- Keyspace, `-s`, `-l`, restore, multiple devices and the brain
- Distributed cracking frameworks and your own scripts

If you don't select a ruleset, hashcat uses the included "default-passwords" ruleset, so this is all you need to start:

```
$ ./hashcat -m 1000 -a 4 hash.txt
```

And it's fast. On one RTX 4090, a PCFG attack runs NTLM at 18.7 GH/s, using the pure kernel. For fast hashes, the attack is fully accelerated on the GPU. The PCFG feed expands base words into candidates right there, so most candidates never have to cross the PCIe bus. The grammar also includes an OMEN part, a Markov model for passwords that no learned shape describes. This part runs on the GPU as well. If you want to leave it out, set `omen=0`.

There is one trade-off you should know about. To make seeking possible, hashcat rounds the probabilities into small steps. This puts the candidates in an order close to the true probability order, but not exactly the same. We measured what that costs against pcfg_cracker, which keeps the exact order. With the same ruleset, the 6,494 MD5 hashes of example0.hash and between 10 million and 1 billion guesses, hashcat found 94% to 97% of what pcfg_cracker found. In return, hashcat produces candidates more than 60 times faster on a CPU alone. On a GPU, it plays in a different league.

Seeking is also fast. Jumping to keyspace position 1 billion and printing the first 5 candidates from there takes less than 2 seconds:

```
$ time ./hashcat -a 4 --stdout -s 1000000000 -l 5
02139305
02139306
02139307
02139308
02139309

real    0m1.883s
```

1.1.1. Masks: A Shape or a Fragment

If you know something about the shape of the password, you can give a PCFG attack a mask. The important difference from a mask in a brute-force attack is that this mask does not filter finished candidates. Instead, it is applied inside the grammar, so shapes that cannot fit drop out before anything is generated. These shapes don't take up any keyspace. The candidates that remain keep the probabilities the grammar gave them, so they come out in the same order as without the mask, only with the others left out.

Look what happens when you give it a mask that allows only candidates ending in "cat" to pass through:

```
$ ./hashcat -a 4 --stdout -l 10 mask=?a?a?a?acat
wildcat
coolcat
bearcat
Wildcat
allycat
bluecat
babycat
nyancat
hellcat
Coolcat
```

The likeliest ones come first, such as "wildcat" and "coolcat", even though no wordlist is involved. Every word comes from the grammar itself.

A mask can also hold a fragment in the middle. Here, "red" sits at positions 3 to 5. Note how the candidates come from many different shapes, and that "red" has nothing to do with colors here:

```
$ ./hashcat -a 4 --stdout -l 10 mask=?a?ared?a?a
paredes
heredia
firedog
boredom
jared12
jared11
Paredes
firedup
zgredek
coreduo
```

The same mask in a brute-force attack has 4.6 times as many candidates, and they are not generated in order of likelihood:

```
$ ./hashcat -a 3 --total-candidates ?a?ared?a?a
81450625

$ ./hashcat -a 4 --total-candidates mask=?a?ared?a?a
17876251
```

The mask uses the same syntax as in attack-mode 3, including the custom charsets `-1` to `-8`.

Some of our beta testers say that `-a 4` is the better `-a 3` now.

1.1.2. Hints: Attacking One Person

A trained grammar describes 2 aspects of how a password is formed: what shape the password has, and which word fills that shape. Sometimes you already know the words. Say you are cracking one person's password, and old breaches gave you 33 passwords this person used before. In a real password history, about half of the passwords usually cluster around a few habits. Here, the person likes random strings. One of them is used on its own, and also as the base for every site, followed by "!!" and the name of the site. Their own name gets a decoration such as "123!@#", a "!" or a "#1", and shows up written backwards once. The names of people close to them, such as their sister Luna, appear only as they are, capitalized, or with a date appended. The name of their home town, Vienna, follows the same pattern. The rest are outliers in both directions: easy ones like "123456", picked under pressure, keyboard walks like "1qaz2wsx", a long run of digits, a social security number, phone numbers, and dates and PIN codes such as "0412" or "04121989". What you don't know is the password they use today.

This is what the included "hints" ruleset is for. It keeps the learned shapes of real passwords, but leaves the words out. You supply the words, and the grammar puts them into the shapes that real passwords have. Write what you know into a file, one word per line. After each word, you can add a tab and a weight for how likely you think it is. These weights don't need to be exact. Here, they are rough guesses written by hand, with the habits first and the outliers last:

```
$ pr -T -3 -w 84 -e20 passwords.txt
```

```
kqmvtrzw            0.4     Marco1975           0.05    123456              0.01
jason123!@#         0.3     vienna              0.05    qwerty123           0.01
Jason2024!          0.3     Vienna2019          0.05    1qaz2wsx            0.01
Tz9$Lq2&Wm5!        0.25    jason#1             0.05    zxcvbnm             0.01
luna                0.2     nosaj123!@#         0.05    password            0.01
Luna1993            0.2     04121989            0.02    Winter2022!         0.01
luna0612            0.15    0412                0.02    2580                0.01
daniela             0.1     kqmvtrzw!!gmail     0.02    1234567890          0.01
Daniela1989         0.1     kqmvtrzw!!paypal    0.02    987-65-4325         0.01
daniela0809         0.1     kqmvtrzw!!ebay      0.01    555-0137            0.01
marco75             0.05    kqmvtrzw!!netflix   0.01    015209876543        0.01
```

The hints ruleset tries each old password on its own, extended with digits and symbols, with capitalization, and joined with another old password in both orders. It mixes all of these in one stream, ordered by how likely these patterns are in real passwords:

```
$ ./hashcat -a 4 --stdout -l 63 hints hintfile=passwords.txt | pr -T -3 -w 84
```

```
kqmvtrzw                luna123                 kqmvtrzwtz9$lq2&wm5!
jason123!@#             luna1993123             kqmvtrzwluna
jason2024!              kqmvtrzw1               kqmvtrzwluna1993
tz9$lq2&wm5!            Jason123!@#             jason123!@#kqmvtrzw
luna                    Jason2024!              jason2024!kqmvtrzw
luna1993                Tz9$lq2&wm5!            tz9$lq2&wm5!kqmvtrzw
luna0612                Luna                    lunakqmvtrzw
daniela                 Luna1993                luna1993kqmvtrzw
daniela1989             kqmvtrzw12              04121989
daniela0809             kqmvtrzw11              0412
kqmvtrzw123             luna0612123             kqmvtrzw!!gmail
marco75                 daniela123              kqmvtrzw!!paypal
marco1975               daniela1989123          Luna0612
vienna                  daniela0809123          Daniela
vienna2019              jason123!@#1            Daniela1989
jason#1                 jason2024!1             Daniela0809
nosaj123!@#             tz9$lq2&wm5!1           KQMVTRZW
Kqmvtrzw                luna1                   kqmvtrzw13
jason123!@#123          luna19931               kqmvtrzw10
jason2024!123           kqmvtrzwjason123!@#     kqmvtrzw01
tz9$lq2&wm5!123         kqmvtrzwjason2024!
```

The order is the trained one. A word followed by 3 digits comes before 2 words joined together. This is because that is the more common shape in real passwords, rather than because someone wrote a rule for it.

If you also know something about the shape, a mask narrows it down even further. The habit of appending a date makes "Luna2026" a good guess for the password they use today. With the mask, it is candidate 1,484 in the `--stdout` stream. Here are the first 9 candidates generated with the mask:

```
$ ./hashcat -a 4 --stdout -l 9 hints hintfile=passwords.txt mask=?u?l?l?l?d?d?d?d | pr -T
-3 -w 84
```

```
Luna1993                Luna2000                Luna2012
Luna0612                Luna2010                Luna2003
Luna1234                Luna2002                Luna2001
```

A few more things worth knowing about hints:

- For a few words, `hintwords=daniela,jason,1989` on the command line works as well
- Each hint word appears at most once in a candidate, because a password based on a fact usually includes that fact only once. Set `hintrepeat=1` to allow repeats
- A word without a probability is weighted by its position. By default, the first word counts as the most likely one. With `hintrank=flat`, all words have the same weight. With `hintrank=linear`, each word has half the weight of the one before it
- Every candidate contains one of your words. The attack is only as good as what you know
- Hint attacks produce candidates more slowly than the default ruleset, because the hint words have to be placed on the host. For an attack on one person, this rarely matters. If you want the speed back, `-r` still amplifies on the GPU
- Positions where a word would repeat are skipped, but they still count as positions. That is why the status shows a large "Rejected" count, and why `-l 63` printed 62 candidates above

If you train your own grammar, `tools/pcfg_hints.py` turns it into a hint ruleset. The association attack uses the same ruleset too, with the words taken from each hash. See section 1.2 for details.

1.1.3. Rulesets and Settings

Hashcat ships 2 rulesets, "default-passwords" and "hints". You can train your own with pcfg_cracker, because hashcat reads its ruleset format directly. The PCFG attack page in our wiki also lists 2 more trainers: pcfg-go and hpcfg. If you name more than one ruleset, they all run. With `weights=2:1`, you can shift the share toward the one you trust more:

```
$ ./hashcat -m 0 -a 4 hash.txt default-passwords my-ruleset weights=2:1
```

If you know the password length, set `pwmin` and `pwmax`. The limits apply while the keyspace is counted, so candidates outside the range are never generated. Rules with `-r` and `-g` keep the attack on the GPU engine, and `--debug-mode 6` prints the layout of a PCFG candidate.

See `docs/hashcat-pcfg.md` for every setting.

1.2. Account-Aware Association, Attack-Mode 9

The association attack has always been strong when you know something about each hash, for example the username. The downside was that you had to build the wordlist yourself, with one word per hash. With hashcat v8.0.0, this is no longer necessary, because hashcat now reads the words directly from the hash file.

Usernames are split at separators, at case changes, and where letters meet digits. From passwd lines, hashcat also reads the real name and the last part of the home directory. On top of that, a hash-mode can add hints of its own.

By default, the attack runs through 3 phases, in this order:

Phase	Candidates
1	The account words
2	The account words with the first 1k rules of `rockyou-30000.rule`
3	The PCFG hints ruleset, see section 1.1.2

The PCFG phase brings the strength of the hint attack to every account in the list, each with its own words and its own salt. Note that this phase never ends. If you want a bounded attack, name the phases you want. This runs the account words, then the rules, and then stops:

```
$ ./hashcat -m 500 -a 9 hash.txt phases=words,rules
```

WPA is a good example of a hash-mode with hints of its own. A capture has no account names, but mode 22000 supplies the network name, which a human chose. It also supplies the 2 MAC addresses, each written as 12 hex digits:

```
$ ./hashcat -m 22000 -a 9 handshakes.hc22000
```

The old one-word-per-hash form still works unchanged. Please note that association still requires one salt per hash. See `docs/hashcat-association.md` for details.

1.3. Table Attacks, Attack-Mode 5

The table attack is back, and it returns under its original attack-mode number from hashcat-legacy.

For those who are unfamiliar: a table attack replaces a token in a word with another token. A token is one or more bytes, so multibyte characters work as well, emoji included. The replacement can be longer or shorter than the token, or even empty. A table has one rule per line: the token to replace, a tab, and the replacement to put in its place.

A table can describe any group of things that people swap for each other. That can be case, leetspeak, emoji or keyboard layouts, but also sports, brands, bands, colors, car makes, football clubs, cities, first names, months, weekdays, or anything else that can be grouped. Hashcat ships tables for leetspeak, case toggling, emoji, sports and keyboard layouts. A keyboard layout table maps one layout to another. Several tables can be combined, and the combination runs on the device.

A table is a plain text file with one rule per line: the token to replace, a tab, and its replacement. The same token can appear on several lines, each with a different replacement. Hashcat then chooses separately at every position where the token occurs, and tries every combination of these choices. These are the first rules of the included leetspeak table:

tables/leetspeak-common.table

```
a	4
a	@
e	3
g	9
i	1
i	!
l	1
o	0
s	$
s	5
```

That is how the 2 separate lines for "s" turn "password" into "pa$$word", "pa55word", and the mixed "pa$5word" and "pa5$word" as well.

Tables can also swap whole words. Let's say your wordlist contains these 2 words "ilikefootball" and "football123":

```
$ cat words.txt
ilikefootball
football123
```

A table that swaps football for other sports needs one line for each of them:

sports.table

```
football	baseball
football	basketball
football	hockey
football	tennis
football	golf
football	rugby
football	cricket
football	volleyball
football	cycling
football	skiing
```

Sports are only an example. The same works for music groups, cities, book titles, or anything else that the people you attack care about. Hashcat ships a larger `tables/sports.table` that pairs every sport with every other sport.

The table replaces football in each of the 2 words with every one of these sports:

```
$ ./hashcat -a 5 --stdout words.txt sports.table identity=0 template=1 | pr -T -4 -w 91
```

```
ilikebaseball          ilikerugby             baseball123          rugby123
ilikebasketball        ilikecricket           basketball123        cricket123
ilikehockey            ilikevolleyball        hockey123            volleyball123
iliketennis            ilikecycling           tennis123            cycling123
ilikegolf              ilikeskiing            golf123              skiing123
```

Here, `identity=0` leaves out the unchanged word, and `template=1` leaves out words that did not match any token in the table. That way, the wordlist acts as a set of patterns. Without `template=1`, a run over rockyou would spend 96% of its time on words that don't name any sport.

By default, every combination of substitutions is tried. With `single=1`, hashcat tries one substitution at a time instead. You can also use `maxperm` and `cap=N` to put a limit on the work. See `docs/hashcat-table.md` for all settings.

1.4. Wordlists in Masks, Attack-Mode 12

Attack-mode 12 brings wordlists into masks. The following 2 new placeholders are available:

Mask	Candidates
`?w`	The first wordlist, anywhere in a mask
`?q`	A second wordlist
`?w?d?d`	Each word with 2 digits after it
`?w?q`	The 2 wordlists combined

We borrowed `?w` from JtR, where it stands for the word in a hybrid mask. We also added `?q` for a second wordlist, a feature that JtR does not have. The `?w` placeholder is required, and `?q` is optional. Literal text and mask tokens can appear on either side of a word. Mask files work as long as every mask contains `?w`. Increment mode is not available, because shortening the mask could remove the word. See the "Attack-Mode 12 Markers" table in `--help`.

This is more than just a new attack-mode. The combinator attack and both hybrid attacks now run on this engine as well, with the same candidate order and the same keyspace as before.

1.5. Rules in Every Attack-Mode

Until now, rule files given with `-r` worked only in straight and association attacks. With hashcat v8.0.0, they also work in combinator, brute-force, PCFG, table and hybrid attacks, and with wordlists in masks. Mask files, increments, Markov settings and custom charsets work as before.

There is one thing to keep in mind. When rules are enabled, the base candidates come from the host, and the rules run on the device. Hashcat displays a warning when this happens:

```
$ ./hashcat -m 1000 -a 3 -r rules/rockyou-30000.rule hash.txt ?a?a?a?a?a?a
```

```
ATTENTION! The mask is being generated on the host.
No kernel both walks a mask and applies a rule, so a mask attack given rules builds
its candidates on the host and the rules amplify them on the device.
...
Guess.Base.......: Feed (mask)
Guess.Mod........: Rules (rules/rockyou-30000.rule)
Speed.#01........:  8234.2 MH/s (80.41ms) @ Accel:711 Loops:115 Thr:64 Vec:1
```

This is the same split that rules always had in attack-mode 0, where the wordlist comes from the host and the rules run on the device. On the same RTX 4090, attack-mode 0 with rockyou.txt and the same ruleset runs at 6.8 to 6.9 GH/s, so a mask with rules is even a little faster. Larger rulesets win back more of the speed, because they do more work per base candidate. Attacks without rules still generate their candidates on the device. The exception is PCFG: there, the rules run inside the device engine, so a PCFG attack with rules keeps its speed.

The debug output now also names the rule that produced a crack in combinator, brute-force and hybrid attacks and with wordlists in masks, as it always did in straight attacks.

1.5.1. More Rule Features

With `--rules-concat`, you can join several rule files into one sequence. Their rule counts add up instead of being multiplied, as they are when you stack `-r` options.

Rule operands for `$`, `^`, `i`, `v` and `o` now accept non-ASCII characters directly, so you no longer need one `\xHH` escape per byte. And there are now 5 new rule functions available:

Rule	What it does
`h`	Converts the whole password to lowercase hex
`H`	Converts the whole password to uppercase hex
`S`	Shifts the case of each character, as in JtR
`vNX`	Inserts the character X every N characters
`BNX`	Adds the byte value of X at position N

1.6. Feed Plugins, Attack-Mode 8

All of the attacks above share one design, which deserves its own section. In hashcat v8.0.0, candidate generators run as plugins called feeds. A feed is selected by name, and hashcat runs one instance of it per device. Unlike a generator that you pipe into hashcat via stdin, a feed takes part in keyspace calculation, work distribution, restore, status and debug output.

A feed can also hand base words to the device and let the GPU expand them into candidates. That way, the feed becomes an amplifier. This is how PCFG and table attacks reach GPU speed on fast hashes. Wordlists and masks are feeds themselves. Here's a wordlist feed, selected by name, with rules on top:

```
$ ./hashcat -m 1000 -a 8 hash.txt wordlist example.dict -r rules/dive.rule
```

```
...
Guess.Base.......: Feed (example.dict)
Guess.Mod........: Rules (rules/dive.rule)
Candidate.Engine.: Device Generator
```

Hashcat v8.0.0 includes the following feeds:

- Wordlist, stdin, masks, hybrids and association
- PCFG and tables
- Random generation, written in Rust, which is built when Rust is available

The wordlist feed also supplies the dictionaries to straight, combinator and hybrid attacks, as well as to wordlists in masks. It also supplies them to the wordlist form of the association attack.

Because stdin is a feed now, a stdin attack supports `-s`, `-l` and `--restore`, as long as you give it the same stream again. A pipe has no keyspace, so the status used to show only a count. If you know how many lines the stream has, give that count with `--stdin-count`. The status then shows progress and an estimated finish time. And every feed attack can use `--debug-mode` without `-r` or `-g`. The "Finding-Rule" field then describes the feed.

If you want to write your own feed, you can do it in C or in Rust. Documentation and example feeds are included. `docs/hashcat-generic-attack-mode.md` explains how to use feeds, and `docs/hashcat-generic-attack-mode-development-guide.md` explains how to write them. For Rust, see `docs/hashcat-rust-plugin-quickstart.md`. We are curious to see what the community will build with it.

1.7. Lookup: Where an Attack Reaches a Password

Sometimes you want to know whether an attack covers a certain password, and how far into the attack it comes. Maybe you cracked a hash with one attack and wonder whether a cheaper attack would have found it too. Or you know the password of a test account, and you want to be sure your attack covers it before you spend a weekend of GPU time on it. Until now, the only way to find out was to run the attack, or to generate every candidate with `--stdout` and search through the output.

With hashcat v8.0.0, `--lookup` gives you this information directly. You give it the password, and hashcat works out where the attack produces it. Hashcat then prints the result and quits. It needs no hash file, and it opens no device. Here's a mask attack:

```
$ ./hashcat -a 3 --lookup=Summer2026 '?u?l?l?l?l?l?d?d?d?d'

lookup: 'Summer2026'
lookup: mask ?u?l?l?l?l?l?d?d?d?d reaches it
lookup: base word 133771000 of 175760000, 76.1100% into the run
lookup: this run reaches it at -s 133771000, because -a 3 counts -s in base words
lookup: -s 133771000 -l 1 runs the one cell of 17576 candidates that holds it, where it is
number 4162
```

The answer shows how far into the attack the password sits, and it gives you the exact `-s` value that reaches it. With `-s` and `-l 1`, you can run just that small piece of the attack, for example to confirm that a hash really cracks there. For a mask, the result is available instantly, because hashcat can compute the position. A wordlist takes one pass over its words, with a progress bar and an ETA on long scans.

`--lookup` works in straight, combinator, brute-force, PCFG and hybrid attacks, and with wordlists in masks. In hybrid and combinator attacks, it also shows how the password is cut between the wordlist and the mask, or between the 2 wordlists. And when an attack can never produce a password, hashcat displays that result along with an explanation of why:

```
$ ./hashcat -a 6 --lookup=hashcat2026 example.dict ?d?d?d?d

lookup: 'hashcat2026'
lookup: nothing in this run produces it
lookup: no way of cutting it leaves a word this wordlist holds and a mask value beside it
```

With PCFG, the answer goes even further. Hashcat names the structure that produces the password, and it shows how each slot of that structure is filled:

```
$ ./hashcat -a 4 --lookup='Sarah1992!'
```

```
lookup: 'Sarah1992!'
lookup: derived by structure A5Y1O1, at cost 34 of costmax 64
lookup: A5=sarah C5=ULLLL Y1=1992 O1=!
lookup: base word 254973 of 12747516827886, 0.0000% into the run, which is candidate
889622229 of 34935457766817424
lookup: this run reaches it at -s 254973, because the device engine counts -s in base
words
lookup: -s 254973 -l 1 runs the one cell that holds it
```

Here, "Sarah1992!" consists of a word of 5 letters with a capital first letter, followed by a year and a symbol. If a password is out of reach, hashcat explains why. The grammar may not produce it at all. A mask, `omen=0`, a length limit or a repeated hint word may rule it out. Or it may lie beyond `costmax`. In that case, hashcat displays the `costmax` value that would reach it. For PCFG, you can also ask the same question with the ruleset setting `lookup=`.

A few things to keep in mind. `--lookup` cannot be used together with rules, because a rule cannot be inverted. Table, feed and association attacks do not support it yet. And you can give the password as `$HEX[...]`.

2. Hashes Without a Hash-Mode

Sooner or later, everyone runs into a hash that no hash-mode covers. Maybe a web application salts in its own way, or a vendor decided to hash twice. Until now, your options were to write a new hash-mode, including its kernels, or to fall back to a much slower generic plugin. Hashcat v8.0.0 gives you better answers. Hash recipes are compiled into a GPU kernel and are the right choice for most schemes. The Python bridge runs any Python code on the CPU and covers everything else.

2.1. Hash Recipes, Hash-Mode 4000

With hash-mode 4000, you describe the hash on the command line:

```
$ ./hashcat -m 4000 --hash-recipe 'md5(md5(salt) . pass)' hash.txt wordlist.txt
```

That's it. Hashcat compiles the recipe into the kernel at startup. Most recipes run close to the speed of a built-in pure kernel, and some even run faster. On a Radeon Pro W7800, recipes in straight and brute-force attacks run within 1% to 5% of the native kernels. The recipe `hmac_sha256(salt, pass)`, for example, runs at 7.67 GH/s on an RTX 4090, while the native pure kernel of mode 1460 runs at 4.71 GH/s.

Recipes use the syntax of hx, the hash expression language. They cover a lot of ground:

- Hash functions: MD4, MD5, SHA-1, SHA-2, SHA-3, Keccak, RIPEMD-160, RIPEMD-320, BLAKE2, SM3, Whirlpool, Streebog, GOST R 34.11-94, MD6, LM and MySQL323, plus 26 families that no earlier hashcat could compute at all. Section 6.1 lists them. Most families have an HMAC form as well
- PBKDF2, AES and DES, to derive keys and to encrypt, decrypt or unwrap data
- Transforms such as upper, rev and pad, on passwords, salts, digests, loop variables and derived keys
- Salts stored as hex or base64, decoded with fromhex and frombase64
- Digests that are sliced, or re-encoded as hex or base64
- Variables used across several statements, plus loops and operations on computed values. These operations include xor, and and or
- A second salt, a pepper and the user name, read from the hash line
- Results from 8 bytes up, raw or encoded, and hash lines with a fixed prefix such as `{SHA}`

Mode 4000 now covers most of hx. It compiles 903 of the 1005 expressions in the hx catalog.

Here are some examples:

Recipe	What it is
`md5(md5(salt) . md5(pass))`	MyBB and IPB forums, mode 2811
`sha1(salt . md5(pass))`	A web application that hashes twice
`md4(utf16le(pass))`	NTLM, written as a recipe
`hmac_sha256(salt, upper(pass))`	An HMAC with a transformed key
`trunc(sha512(pass), 32)`	A digest cut to 32 hex characters

A recipe is not limited to a password and a salt. It can read more fields from the hash line, and it can work on computed values:

Fields from the hash line	Work on computed values
`sha256(pass . fromhex(salt))`	`md5(xor(sha1_bin(pass), salt))`
`md5(pass . frombase64(salt))`	`hex(rev(md5_bin(pass)))`
`md5(pass . user)`	`h = md5(salt); md5(h . pass)`
`sha256(salt . pass . pepper)`	`sha1(utf16be(pass))`

Recipes are not limited to fast hashes either. A recipe can loop, with the iteration count taken from each hash line, as in `for i = 1 to iter { h = md5_bin(h . pass) }`. It can derive a key with PBKDF2, and it can decrypt or unwrap data with AES. It can end in a comparison, either against a constant or between 2 values computed by the recipe. That way, slow formats such as phpass, PBKDF2 hashes and 1Password agilekeychain run as recipes too, once their hash lines are written in the form mode 4000 reads.

Let's try it with a scheme that hashcat has no hash-mode for, sha1(salt . md5(pass)):

```
$ ./hashcat -m 4000 --hash-recipe 'sha1(salt . md5(pass))' -a 3 hash.txt
'?u?l?l?l?l?l20?d?d!'
```

```
...
Status...........: Cracked
Hash.Mode........: 4000 (Hash Recipe, see --hash-recipe)
Guess.Mask.......: ?u?l?l?l?l?l20?d?d! [11]
Speed.#02........: 10622.5 MH/s (7.32ms) @ Accel:474 Loops:45 Thr:1024 Vec:1
Progress.........: 3875508000/30891577600 (12.55%)
Candidate.Engine.: Device Generator
```

On an RX 7900 XTX, "Autumn2026!" cracks in 1.3 seconds, startup included. The very first run takes about 4 seconds, because it builds the recipe's kernel. One RTX 4090 runs the same recipe at 21 GH/s. Many custom schemes can now use a recipe in place of a dedicated hash-mode.

See `docs/hashcat-hash-recipe.md` for the full recipe language.

If you work with JtR, there's more. JtR writes many of its formats as `$dynamic_N$` hashes. With `--dynamic-x`, hashcat reads the format from the first hash and runs it with the matching recipe, so you need neither `-m` nor `--hash-recipe`. This covers 407 of the 409 dynamic formats built into JtR, and all 58 formats in JtR's `dynamic.conf`. Only phpass and HTTP digest authentication are left out. With `-O`, a format runs in a hash-mode with optimized kernels instead, where one exists. A format that hashcat doesn't recognize, for example from your own `dynamic.conf`, still loads with `-m 4000` and a recipe for what it computes.

2.1.1. A New Hash-Mode Is Now Only a Module

Recipes also changed how we write hash-modes. Before, adding a new hash-mode required a module, up to 4 pure kernels, and the optimized kernels. Now a module can simply name its recipe, and the recipe replaces the pure kernels. This is mode 10, md5($pass.$salt):

```
static const char *HASH_NAME      = "md5($pass.$salt)";
static const char *HASH_RECIPE    = "md5(pass . salt)";
```

We converted the pure kernels of many existing hash-modes:

Recipe conversion	
Fast modes	182
PBKDF2 and iterated modes	47
Kernel files removed	567
Lines removed	111k

The fast modes include MD5, NTLM, DCC, Whirlpool, SHA-3, Keccak, Streebog and the HMAC-MD5, HMAC-SHA1, HMAC-SHA256 and HMAC-SHA512 modes. The PBKDF2 and iterated modes include PBKDF2-HMAC-SHA256 and the formats built on it, such as macOS, GRUB 2, Django and Cisco-IOS 8, as well as phpass, Drupal7, SAP CODVN H, AIX and the SCRAM formats.

From a user's perspective, nothing changes. The hash format, the parser, the self-test and the optimized kernels of each converted hash-mode stay the same. Hash-modes that now run only as a stored recipe moved into a number range of their own, starting at 40000, but their old numbers keep working. See section 5.7 for details. For us, it means a fix to one primitive, such as MD5 or PBKDF2, now reaches every recipe mode at once. If you write a new fast hash-mode from these families, it now needs only a module that parses the hash line and names its recipe. See "Kernel: fast hash type (recipe)" in `docs/hashcat-plugin-development-guide.md`.

2.2. Python Bridge

The Python bridge was introduced with hashcat v7.0.0 as a quick way to add hash-matching logic in Python. In v8.0.0, we made it easier to set up and much faster.

Mode 73000 now starts one ordinary Python worker process per CPU thread and sends it the candidates. This works on Windows and macOS as well as on Linux, and it no longer needs a free-threaded Python build. Hashcat runs the `python3` it finds in PATH, or `python` on Windows. This way, virtual environments and pyenv work without any extra steps. If you want another interpreter, name it with `--bridge-parameter2`. Hashcat also no longer needs the Python headers to build.

Python bridge, 28 CPU threads	Before	Now
SHA-256 Python plugin	16 kH/s	611 kH/s
Candidates per launch	8	1k

These numbers come from an i7-14700K. The Python plugin that ships with hashcat gains 10%.

Writing a plugin got simpler as well:

- A plugin is a normal Python module, with `calc_hash()` and a self-test pair in `ST_HASH` and `ST_PASS`
- The `init()` and `term()` hooks are optional, and `kernel_loop()`, `hcsp` and `hcmp` are gone
- If `calc_hash()` raises an exception, the attack stops. The exception's traceback is also printed, so mistakes no longer go unnoticed
- For debugging, you import your plugin as a normal module, without hashcat

The status shows which interpreter is in use. `make install` now also installs the Python folder, so mode 73000 runs from any working directory after an install.

> Please note that mode 72000 is removed, and a few plugin hooks changed. See section 5.8 and `docs/hashcat-python-plugin-development-guide.md` for the upgrade steps.

2.3. Rust Bridge

The Rust bridges now share the `hashcat-sys` crate, and every crate under `Rust/bridges` is built into `bridges/subs`. Mode 74000 copies only the output digests between host and device, and autotune can size its launches up to 16k candidates. Rust plugins require Cargo 1.85. See `docs/hashcat-rust-plugin-quickstart.md` to get started.

3. FPGA Support

Hashcat has traditionally focused on GPUs and CPUs. With hashcat v8.0.0, this changes. Hashcat now cracks bcrypt on FPGAs, as hash-mode 75000.

When we introduced the Assimilation Bridge in hashcat v7.0.0, FPGAs were the original reason for it. An FPGA plugin was already on our list of plans at the time. The FPGA boards now appear in hashcat as devices, through the bridge. You use the same attacks, the same rules and masks, and the same restore and status as you would on a GPU. Every board gets its own autotune for launch sizes and its own temperature watchdog. Board health is reported just as it is for a GPU. Every board shows its temperature and has its own temperature limit.

The FPGA computes the hashes, while a regular backend device generates the candidates. A CPU OpenCL runtime is enough for that, so a headless host needs no GPU at all. FPGA support is available on Linux.

3.1. What bcrypt Does to a GPU

bcrypt keeps a GPU busy without using most of it, and the power draw shows it. On an RTX 4090, both runs below show 99% utilization, but bcrypt draws only about a quarter of the card's 450 W limit:

RTX 4090	Speed	Power	Peak	Clock	Temperature
MD5	59 GH/s	434 W	441 W	2.7 GHz	71 C
bcrypt cost 10	7.8 kH/s	117 W	118 W	2.8 GHz	37 C

So "99% utilization" only shows you that a kernel is running. It doesn't show you how much of the chip that kernel uses. An FPGA avoids this problem, because the whole chip can be configured as bcrypt cores, each with its own S-boxes.

3.2. What We Built

The bcrypt compute core comes from JtR. It is the ZTEX bcrypt core written by Denis Burykin. We modified the core, and we wrote everything around it. `src/fpga/bcrypt/PROVENANCE.md` lists every change, file by file.

Two hashes per core. On the YPCB, a design with one hash per core runs out of block RAM at 450 cores, while about a third of the logic stays unused. So we changed the core to run 2 independent hashes, interleaved cycle by cycle. While one hash waits for the result of its previous step, the core works on the other one. Every register became a chain of 2 stages, and the hash number was added as the top address bit for every memory. The extra stage puts a second register inside the round loop. This lets the core run at twice the clock of the rest of the design, so each hash keeps the full rate. The S-box memories of a core were half empty before, so the second hash costs no block RAM at all. The published YPCB bitstream runs 760 hashes at once on 380 cores, with the design at 115 MHz and the cores at 230 MHz.

Everything around the cores. The rest of the design was written for hashcat:

- The feeder receives the passwords, expands the bcrypt key on the FPGA, and streams it to a free core. Sending the password instead of the expanded key keeps a record at about 16 bytes instead of 128
- On most boards, a small comparator next to each core drops the candidates that cannot match before they reach the host. With one hash per salt, that is about 98% of them
- Every batch carries a known answer, so a core that computes wrong digests is detected and named
- Each core keeps its S-boxes in block RAM or in logic, so an array can use both resources of a chip
- The board designs: PCIe shells built on LiteX and LitePCIe, a USB shell for the ZTEX 2.13, a design for the Arty Z7's processor system, and a separate Vitis flow for the Alveo U50

The sources to rebuild the bitstreams are in `src/fpga/`, together with a simulation that checks the whole datapath without a board.

3.3. Supported Boards

Published bitstreams are available for the following 7 board types:

- ZC706
- YPCB-00338-1P1 (Kintex-7 480T)
- Enigma X1
- Alveo U50
- ZTEX 1.15y
- ZTEX 2.13
- Arty Z7-20

The bitstreams are distributed as binaries, outside the source tree. Download the file for your board from https://hashcat.net/bitstreams/.

The `src/fpga/` directory contains the sources needed to rebuild the bitstreams for 5 of the boards. The Alveo U50 uses a separate Vitis build flow. The 2 files for ZTEX 1.15y are JtR's Spartan-6 bitstreams, redistributed unchanged.

The boards connect over USB, Ethernet or PCIe. The LitePCIe driver ships with hashcat. It builds through DKMS and supports Secure Boot signing. See `docs/hashcat-fpga-setup.md` for the setup of each board.

3.4. Speed

These are the speeds for a single bcrypt hash:

Cost 10	Speed
YPCB, 2 boards	8.9 kH/s
Alveo U50	7.4 kH/s
YPCB, one board	~4.5 kH/s
ZTEX 1.15y	3.8 kH/s
Arty Z7	345 H/s
ZTEX 2.13	125 H/s

Cost 5	Speed
Alveo U50	202 kH/s
Arty Z7	10.5 kH/s
ZTEX 2.13	3.5 kH/s

And this is how they compare to an RTX 4090 on the same bcrypt hashes, with a single hash and attack-mode 3 or a wordlist. Please note that the FPGA and the GPU numbers come from separate runs, with 2 days between those runs:

Cost 10	Speed	Relative
RTX 4090	7.8 kH/s	1.00x
YPCB, 2 boards	8.9 kH/s	1.15x
Alveo U50	7.4 kH/s	0.96x

Cost 5	Speed
RTX 4090	238 kH/s
Alveo U50	202 kH/s

Here's a short example of a PCFG attack on 2 YPCB boards. The target is a bcrypt cost 10 hash of "blowfish", a password with 8 letters:

```
$ ./hashcat -m 75000 -a 4 -d 1,2 hash.txt mask=?l?l?l?l?l?l?l?l
```

3.5. Scaling

FPGA boards need very little bandwidth, so they scale the way GPU mining rigs did. For example, the ASUS H370 Mining Master mainboard has 20 PCIe x1 riser ports. The blue sockets look like USB, but each one carries a single PCIe lane over a USB 3 cable to a riser. Each of these risers holds one card. That's about 20 cards on one mainboard.

4. Large-Scale and Distributed Work

If you run hashcat on many devices or many nodes, or against very large hash lists, this section is for you.

4.1. Seekable Wordlists

In a distributed setup, every device or node gets a range of the keyspace. With a wordlist, that used to mean reading every line before the assigned range, just to find where the range starts. With large wordlists and ranges near the end, that could take a while.

Hashcat now builds a seek database for every wordlist, so a device can start near its assigned range. The first run builds the database, and every seek after that is instant. Here's a seek to line 14 million of rockyou.txt, which has 14.3 million lines:

```
$ time ./hashcat -a 0 --stdout -s 14000000 -l 3 rockyou.txt
05448708072
05448704298
054487-9

real    0m0.137s
```

Hashcat can also seek directly in compressed wordlists, as long as the format allows it. Wordlists, hash lists and rules now accept zstd as well as gzip and xz:

Format	Made with	Seeking
xz, multiple blocks	`xz -T0`	Direct
zstd, multiple frames	`pzstd`	Direct
xz, concatenated streams	`cat a.xz b.xz`	Direct
xz or zstd, single stream	`xz`, `zstd`	From the start
gzip	`gzip`	From the start

4.2. Many Wordlists, One Keyspace

Hashcat has always accepted several wordlists, or a whole directory of them, in one attack. Until now, however, `--skip`, `--limit` and `--keyspace` refused to work with them. The same was true for mask files and `--increment`. In a distributed setup, every wordlist and every mask had to become a job of its own.

With hashcat v8.0.0, they all form one keyspace. This works for several wordlists and for whole directories in straight and hybrid attacks, and the combinator attack now accepts a directory on the left. A mask file, or the lengths of an `--increment` run, form one keyspace in the same way. The `--skip`, `--limit` and `--keyspace` options cover the full queue. This means you can hand out one large job, and every node gets its own slice of it, even when that slice starts in one wordlist and ends in the next.

Here's a directory that holds example.dict, with 128,416 lines, and rockyou.txt. The keyspace covers both files, and a range can cross from one file into the next:

```
$ ./hashcat -a 0 --keyspace wordlists
14472808

$ ./hashcat -a 0 --stdout -s 128414 -l 4 wordlists
zzzzzzzzzz
zzzzzzzzzzz
123456
12345
```

The files of a directory are sorted by name, so every node sees them in the same order. The wordlist keyspace now counts physical lines, so a line hashcat cannot use shows up as a reject. As a result, later keyspace positions don't move.

4.3. Lookup in Distributed Attacks

In a distributed attack, every node runs only its own range of the keyspace. If you want to know which range covers a certain password, or you want to run just that one piece again, `--lookup` provides that information. It works across a whole directory of wordlists, and it names the wordlist that holds the password:

```
$ ./hashcat -a 0 --lookup=sunshine1 wordlists
lookup: 'sunshine1'
lookup: word 129056 of 14472808, in wordlists/rockyou.txt
lookup: 0.8917% into the run
lookup: this run reaches it at -s 129056, because -a 0 counts -s in words
lookup: -s 129056 -l 1 runs the one word
```

If every node runs an equal range, the `-s` value identifies the node whose range covers the password. Hybrid, mask and PCFG attacks work the same way. See section 1.7 for more examples.

4.4. Brain Server

The brain server got a new database. Until now, a brain session was one large sorted array. Every commit from a client had to merge its batch into the whole array, so the larger the database grew, the longer each commit took. And while one commit ran, every other client had to wait.

The server now splits the database into 1024 shards, and each shard has its own lock. The cost of a commit now depends on the size of the batch, and no longer on the size of the database. With 300 million entries, the results look like this:

Brain, 300M entries	Before	Now
Commit	923 ms	12 ms
Lookup	598 ms	45 ms
Longest wait for a client	2.6 s	39 ms
Taking in 300M candidates	1113 s	56 s
Memory above the data	26%	1.3%

The server now reserves one address range for its data. This takes less processing time and memory for large sessions, but it requires unrestricted virtual address space. It does not require physical memory to back the entire reserved range.

Clients now deduplicate ranges and candidates by default. The status shows a breakdown of brain rejects into ranges that are reserved elsewhere and candidates the server has already seen.

4.4.1. Filling the Brain From an Earlier Attack

Say you already ran an attack against an expensive target, such as a bcrypt hash list, but without the brain. Now you want the brain to keep your next attacks from testing those candidates again. Until now, the only way to get them into the brain was to run the same attack again as a brain client, at the same slow speed.

With the new `--brain-feed` option, you pipe the candidates of the earlier attack from `--stdout` straight into the brain. No hash is computed, so this takes seconds instead of days. The brain session is normally computed from the hash list, and the feeder doesn't have one. So start a brain client on the hash list once, and read the session from its status:

```
Session.........: hashcat (Brain Session/Attack:0x65127819/0x88786d89)
```

Then feed the earlier attack into that session:

```
$ ./hashcat --stdout example.dict -r rules/best66.rule | ./hashcat --brain-feed
--brain-password mypassword --brain-session 0x65127819
```

```
Feeding brain session 0x65127819, reading candidates from stdin.

Fed 8475456 candidates from 8475456 lines into brain session 0x65127819, 1703038 were
already known.
```

These 8.5 million candidates went in within 2 seconds. Some of them were already known, mostly duplicates that the rules produce. From now on, every brain client on this hash list skips them, whichever attack produces them. The server keeps running while you feed it, so other clients can keep

cracking.

This pays off for expensive hashes. For fast hashes with fewer than 1024 salts, the brain does not store candidates at all, because testing a candidate is cheaper than checking it against the brain's database. See section 5.2.

> The brain also moves to protocol version 2, with new candidate identities and stretched SHA-256 authentication. Please read section 5.2 before you upgrade a brain setup. See `docs/hashcat-brain.md` for details.

4.5. Large Hash Lists

Large hash lists got a lot of attention in this release. Hashcat now sizes its bitmap filters for both the hash list and the device, and the filter probes mix all 4 words of each digest. On top of that, every CPU core now takes part in counting, parsing, sorting, deduplicating and indexing the hash list. For example, a list of 5 million salted hashes now loads in 1.2 seconds instead of 7.0 seconds.

Load time, 7950X	Before	Now
5M NTLM	3.3 s	0.19 s
10M NTLM	7.1 s	0.36 s
50M NTLM	40 s	1.7 s

NTLM, RTX 4090	Before	Now
1k hashes	93 GH/s	129 GH/s
100k hashes	90 GH/s	116 GH/s
50M hashes	1.1 GH/s	58 GH/s

On an RX 7900 XTX, 50 million NTLM hashes went from 1.0 GH/s to 24.9 GH/s.

There are more improvements for large lists:

- Duplicate hashes share one stored plaintext, which saves time and memory
- Outfiles and potfiles lock and flush once per launch, and unsalted potfile lookups use digest-prefix buckets
- Pairing LM halves is now nearly linear instead of quadratic
- If another run has already cracked the whole hash list, hashcat skips the bitmaps, kernel builds, device buffers and autotune
- Loading a large potfile no longer takes hours for hash-modes with large salt data, such as BitLocker, 7-Zip, KeePass and WinZip
- Hash lists above 4 GB are now addressed correctly

4.6. Shared Machines and Storage

If several runs share one machine, you can now give each process a memory budget, in MiB, with `HASHCAT_DEVICE_MEM_LIMIT` and `HASHCAT_HOST_MEM_LIMIT`.

All rebuildable data now lives under `cache`: the compiled kernels in `cache/kernels`, and the feed data, such as the seek databases, in `cache/feeds/<feed>`. With `--cache-path`, you can move all of it to shared storage.

Here's where this helps. Say you run a pool of 20 compute nodes, and they all read the same wordlists from a network share. Point `--cache-path` on every node to one directory on that share. The first node that needs the seek database of a wordlist builds it, and the other 19 nodes use it right away. A seek database is named after the content of its wordlist, not after its path, so the nodes find it even when they mount the share at different paths. Nodes with the same GPUs and drivers share their compiled kernels in the same way. Every file is written under a temporary name first and then renamed, so a node never reads a file that another node is still writing.

On read-only storage, the feed caches fall back to memory. Devices that share a kernel cache entry compile it only once.

5. Before You Upgrade

> Please read this section before you upgrade an existing setup. Most changes in hashcat v8.0.0 are improvements, but some of them change behavior that you might rely on.

5.1. `--restore` Prints the Stored Command Instead of Running It

A restore file holds the whole command line of its session. Previously, `--restore` read that command line and ran it. To prevent command injection, this is no longer the case (CVE-2026-68766).

With hashcat v8.0.0, `--restore` prints the stored command instead. The command is quoted for your shell, with `--restore-position` added. Read it, and if it is what you expect, run it yourself:

```
$ ./hashcat --session mysession --restore
```

```
...
A restore file holds the whole command line of the session it belongs to, and hashcat does
not run it for you. What this one holds is below.

Read it, then run it yourself if it is what you expect:

  ./hashcat --restore-position --restore-file-path=/tmp/hashcat/mysession.restore -d 2 -m
1000 -a 3 --session mysession --runtime 5 hash.txt '?a?a?a?a?a?a?a?a'

Everything on that line except the two restore options came out of the restore file.
```

If a command cannot be quoted safely for your shell, for example because of non-printable bytes, hashcat prints an escaped argument list instead.

5.2. Brain

- The brain protocol moved to version 2, which changes the candidate identities and uses stretched SHA-256 authentication. Upgrade the clients and the server together. A mismatch reports both versions
- Existing brain databases stay on disk, but they are not loaded, because the session IDs changed. Reusing a `--brain-session` value starts with empty deduplication data
- Clients now default to feature value 3, so they deduplicate both ranges and candidates. In-kernel modes with fewer than 1024 salts skip candidate storage, but keep the range reservation. An explicit setting overrides this
- Range-only clients no longer enable `-S`, so fast hashes keep running on the device. Use `--brain-client-features=2` for this path. The default still enables `-S`
- The attack identity now includes whether `-S` is active, so old coverage records are not loaded. This particular change does not touch the candidate identity databases
- The attack identity now includes only the options of the selected hash-mode. See 5.10 for details. This changes the identity for every hash-mode

5.3. Distributed Work and Monitoring Tools

If you run a distributed cracking framework or your own scripts on top of hashcat, please check these:

- Straight and hybrid attacks now count multiple inputs as one keyspace, and `--skip`, `--limit` and `--keyspace` cover the full queue
- Keyspace positions count base words. Rules and the remaining mask positions expand them later
- Separate dictionary rounds, including the multi-phase association attack, reject `--skip`, `--limit` and `--keyspace`. `--limit` also conflicts with `--loopback`, and both range options conflict with `--restore` across multiple masks
- Exit codes now name the cause, so a wrapper can tell a broken task from a broken environment. A wrong attack command, such as a bad option or a path that does not exist, exits with -2. A named file whose content cannot be used, such as an empty or corrupt hash list, exits with -12. Both used to exit with -1. A bridge or a bridge plugin that cannot load, raises an error or panics exits with -13. Code -2 no longer stands for a GPU watchdog alarm, which hashcat never raised. `docs/exit_status_code.txt` lists every code
- Hashcat reads key presses only from a terminal. Wrappers that sent `s`, `c` or `q` through a pipe can no longer pause, checkpoint or quit a run that way
- Progress now measures the requested range. The `CURKU` and `Restore.Point` fields still refer to the full keyspace. **Monitoring tools must stop adding `--skip` to `PROGRESS`**

5.4. Changed Defaults and Paths

Area	Now
Workload profiles, `-w`	Accepted but ignored. Every run uses the former profile 3
iGPU beside a discrete GPU	Off, unless you select it with `-d`
A device from `-d` fails	The run stops. `--force` lets the other devices continue
Potfiles and sessions	XDG profile directory on installed builds
Kernel cache	`cache/kernels`, feed data in `cache/feeds/<feed>`
Keyboard layouts	`tables/layouts/*.table` instead of `layouts/*.hckmap`
`.hctune`, second column	Names attack kernel 0, 1, 3 or 4, not the attack-mode
Brain clients	Deduplicate ranges and candidates, see 5.2

Old `$HOME/.hashcat` files are not moved automatically. Hashcat displays both new paths and the old one. Custom keyboard layout files in the old format still work. If you have scripts that clear the kernel cache, please update the path. Note that profile 3 may reduce desktop responsiveness.

5.5. Removed and Renamed

Removed or renamed	Now
`.zip` input	Gone for wordlists, hashes and rules. gzip, xz and zstd work
`-c`, `--segment-size`	Removed with the segmented wordlist reader
`--backend-devices-keepfree`	Removed. Hashcat checks free memory or probes allocations
`--hccapx-message-pair`	Now `--eapol-message-pair`. The old name still works
OpenCL 1.0 and 1.1, accelerators	No longer supported. `-D 3` reports the removal
Mode 72000	Removed. Use mode 73000

`HASHCAT_NO_AMD_FREE_MEM=1` skips the AMD free memory query and falls back to the desktop reserve. With `--dynamic-x`, hashcat now reads the format from the first `$dynamic_N$` tag and runs it as a hash recipe in mode 4000, unless you set `-m`. See section 2.1. The PCFG device engine rejects `-j` and `-k`, which still work with host generation.

5.6. Runtime Libraries and Minimum Versions

Hashcat now loads zlib, liblzma and libzstd only when they are needed. If a library is missing, hashcat reports this when you use a feature that needs it. Combinator, brute-force and hybrid attacks, wordlists in masks and `--benchmark` need liblzma to read `hashcat.hcstat2`. Mode 11600 needs liblzma and zlib to verify LZMA and DEFLATE archives. See `docs/hashcat-compression-libraries.md`.

Encoding conversion loads iconv only when conversion is needed. On Windows, hashcat uses the native conversion functions, which do not support EBCDIC, ISO-2022, `//TRANSLIT` or `//IGNORE`.

The minimum versions of the backends changed:

Component	Minimum
CUDA driver and NVRTC	12.0
HIP runtime, Linux and Windows	6.2.0
macOS, for Metal and Apple OpenCL	13.0
OpenCL platform and OpenCL C	1.2
PoCL	5.0, with LLVM 10.0
Intel CPU OpenCL runtime	2020
AMD OpenCL driver	3000
NVIDIA OpenCL driver	500

Below the CUDA or HIP minimum, hashcat disables that backend and falls back to OpenCL. Below the other minimums, hashcat skips the device, unless you override this with `--force`. The minimum versions are listed in `docs/hashcat-requirements.md`.

5.7. Hash-Mode Formats

- The 74 hash-modes that run only as a stored recipe now have numbers from 40009 to 40082, for example 15100 is now 40038. The old numbers still work, and hashcat reports the number you used, so scripts and output parsers don't change. Listings such as `-hh`, `--example-hashes` and `--identify` show the new numbers. `docs/hashcat-hash-recipe.md` lists both numbers for every moved mode. Brain clients that give a moved hash-mode by its old and its new number don't share a brain session. A hash detected without `-m` counts as the new number
- Modes 22000 and 22001 now write the original hash line to the outfile and to `--show`. `--outfile-check-dir` can read these lines for both PMKID and EAPOL. Hash lists for these modes are loaded on one thread again
- Mode 22001 now uses 1240 bytes of salt data per hash, up from 956, to hold the 802.11r fields. This is the same amount as mode 22000 uses. Large hash lists need more host and device memory
- Mode 29700 now takes its keyfile hash as 64 hex characters. Older potfile entries still count as cracked and show as `$HEX[...]`
- Mode 5500 (NetNTLMv1) refuses a line whose last ciphertext block does not fit its challenge. Such a line could never crack
- `--outfile-json` lines now use a spaced style, `{ "hash_hex": ... }`. Parsers won't notice, but byte-exact matching will

5.8. Python Bridge Plugins

- Mode 72000 is removed. Use `-m 73000` for the same hash format and plugin interface
- Plugins no longer need `kernel_loop()`. `init()` and `term()` are optional, and `hcsp` and `hcmp` are gone. Keep `calc_hash()` and `extract_esalts()` as before
- Plugins must define `ST_HASH` and `ST_PASS`. Missing self-test values prevent hashcat from starting

- An exception in `calc_hash()` stops the attack. Plugins must handle candidates they cannot hash, instead of relying on the old "invalid-password" fallback
- `hcshared.dump_hashcat_ctx()`, `hcshared.load_ctx()` and `Python/example.ctx` are removed. Import a plugin as a normal module to debug it
- The `hcshared` salt field getters now read the fixed `salt_t` fields. `get_salt_buf()` still reads the extended salt
- Each worker keeps its own copy of the salts. With 20k salts, 28 workers use about 4.7 GB. If memory is limited, run fewer workers

5.9. Plugin Developers and Package Maintainers

- The plugin interface version is now 800. Custom plugins built for v7.1.2 need porting and a rebuild. `docs/hashcat-plugin-development-guide.md` describes the core library contract and every step of the port
- `SHARED=1` is now the default on Linux and macOS. Android, BSD, CYGWIN and MSYS2 keep `SHARED=0`. The Windows release is shared, but a native Windows build needs an explicit `SHARED=1` for shared plugins
- Modules, bridges and feeds link to `libhashcat.so.8`, `libhashcat.8.dylib` or `hashcat.dll` above their plugin directory. Ship the core library beside the frontend
- To port a module, you now remove `module_dictstat_disable` and set the new hooks such as `module_hash_hints`, `module_hash_recipe` and `module_kern_bits`. You also pass `hashcat_ctx` to the extra-parameter init and term hooks
- Every module now sets every hook in `module_init()`, using `MODULE_DEFAULT` for any hook without an implementation of its own. Hashcat checks all of these hooks when it loads the module. 4 hooks used to be preset by hashcat and are now set by each module as well: `module_usage_notice`, `module_advice_notice`, `module_length_sort` and `module_benchmark_cost`
- Kernel availability now uses `module_kern_bits()` and the `KERN_BIT_*` constants, in place of 16 removed kernel `OPTS_TYPE` flags. There are now 6 auxiliary kernel slots, compared with 5 in the previous version
- `hc_device_param_t` now stores kernels, programs and buffers in arrays. Update old numbered member accesses
- Modules can show their own cost parameters in the benchmark header through the new `module_benchmark_cost()` hook
- Modules register their own command line options with the new `module_register_options()` hook. A module without options of its own sets this hook to `MODULE_DEFAULT`. Option values are read with `module_option_given()`, `module_option_u32()` and `module_option_str()`
- Programs that fill `user_options_t` and call `hashcat_session_init()` now go through the same option checks as the command line. See 5.10 for details
- `module_extra_tuningdb_block()` now returns `char *`. The returned string must be a heap-allocated copy, which hashcat frees, rather than a static string
- Bridges can set the new optional `get_unit_tuning` callback to give a unit fixed tuning, which skips autotune for that unit

- Bridge entry points now receive `hashcat_ctx`. Bridges must set their work-item granularity and all 7 monitoring hooks. Association bridges must use `bridge_salt_pos()` and `salt_per_pw` to choose each salt
- Native builds now target the build CPU. Use `MAINTAINER_MODE=1` for release builds or `MCPU=<core>` for a chosen target
- Master builds default to `PRODUCTION=0`. Release commits, their tags and source archives set `PRODUCTION=1`, like the release packages
- `SOURCE_DATE_EPOCH` makes builds of one commit reproducible
- `USE_SYSTEM_ZLIB`, `USE_SYSTEM_LZMA`, `ENABLE_UNRAR`, `USE_SYSTEM_UNRAR`, `USE_SYSTEM_XXHASH` and `WIN_PYTHON` are no longer supported. Remove them from your build scripts
- See `docs/packages.md` for the shipped files, runtime dependencies and build settings

5.10. Option and Input Checks

Hashcat now checks every combination of options in one place, instead of in checks spread across the code. If you run hashcat from scripts, please check these:

- Contradictory options are rejected, such as `--potfile-disable` together with `--potfile-path`. A combination of 2 actions is rejected, except `-b` with `-B`, `--benchmark-all` or `--benchmark-json`
- Options the attack-mode cannot use are rejected, such as `--remove` in an association attack or `--increment` with wordlists in masks
- Options that have no effect in the chosen action or attack-mode are accepted and ignored. For example, `-S` no longer changes a benchmark, and `--keep-guessing` no longer changes `--show` or `--left`
- An option that would be misleading on its own is rejected without its companion, such as `--debug-file` without `--debug-mode`
- Options used by only some hash-modes, such as the message pair, nonce corrections and PIM range, now belong to those modules. Giving one to a hash-mode that does not use it is rejected. Without `-m`, autodetect considers only the hash-modes that support every module option you gave
- `-n` and `-T` values outside the range of the hash-mode are rejected instead of silently dropped
- A module that fails to load no longer stops `--help`, `-I`, `--hash-info` or autodetect. Only a run of that hash-mode stops
- A run stops early when the hash file looks like a wordlist and the wordlist looks like a hash file. It also stops early if the hash file holds no parsable hash among its first 10,000 lines. `--force` turns these checks off
- A straight attack without a wordlist reads candidates from stdin. On a terminal, this is now rejected, because it usually means the hash file was left out. To type candidates by hand, pipe them in, as in `cat | hashcat ...`

`HASHCAT_OPTION_DEBUG=1` prints the rule behind each decision.

6. New Algorithms

Hashcat v8.0.0 adds 146 new hash-modes. 91 of them are raw hash-modes for the hash functions introduced with hash recipes. See 6.1 for details. All of them are listed here by the categories that `--help` groups them in:

Mode	Raw Hash
40084 to 40175	91 hash functions of the hash recipes, see 6.1

Mode	Raw Hash salted and/or iterated
40004	UUID v3 (RFC 4122 name-based, MD5)
40005	UUID v5 (RFC 4122 name-based, SHA-1)

Mode	Raw Hash authenticated
40083	Fernet (key = SHA256($pass))

Mode	Generic KDF
36100	yescrypt
36200	gost-yescrypt
36600	PBKDF2-HMAC-SHA512(utf16le($pass))
40079	phpass(md5($pass))

Mode	User-Defined Hash
4000	Hash Recipe, see --hash-recipe

Mode	Network Protocol
35300	Kerberos 5, etype 23, TGS-REP (NT)
35400	Kerberos 5, etype 23, AS-REP (NT)
37600	SASL DIGEST-MD5
37601	POP3 APOP
38000	Mikrotik RouterOS EC-SRP5 (Curve25519)
38400	Kerberos 5, etype 1/2/3, DB
38500	Kerberos 5, etype 3, TGS-REP
38600	Kerberos 5, etype 3, AS-REP
40001	OSPFv2/RIPv2 MD5 authentication

Mode	Database Server
36601	MSSQL (2025)
40006	MySQL caching_sha2_password challenge-response (fast auth)

Mode	Operating System
3300	MD5(Sun), SunMD5
35600	gost12512crypt [$gost12512hash$] (Unix)
36300	scryptcrypt, scrypt (Unix)
37400	RACF DES Passphrase
37500	RACF KDFAES Passphrase
38200	Cisco-IOS 14 (MD5 (Type 5) + scrypt, Convoluted)
40002	Huawei / H3C / HPE Comware h6$
40080	NTLM(NTLM_bin($pass))
75000	bcrypt $2*$, Blowfish (Unix) [Bridged: FPGA]

Mode	Enterprise Application Software (EAS)
40081	SAP CODVN H (PWDSALTEDHASH) iSSHA-256
40082	SAP CODVN H (PWDSALTEDHASH) iSSHA-384

Mode	Framework
35800	Symfony Legacy SHA256

Mode	Forums, CMS, E-Commerce
35500	Wordpress bcrypt(hmac-sha384($pass))
40007	MODX Evolution (uncrypt)

Mode	Application Database
40000	XCA database password

Mode	Full-Disk Encryption (FDE)
29491	VeraCrypt BLAKE2s-256 + XTS 512 bit
29492	VeraCrypt BLAKE2s-256 + XTS 1024 bit
29493	VeraCrypt BLAKE2s-256 + XTS 1536 bit
38300	Apple DMG (encrcdsa v2, PBKDF2-HMAC-SHA1 + AES-192 key)
38700	FreeBSD GELI
38800	OpenBSD softraid FDE (PBKDF2-HMAC-SHA1)
38810	OpenBSD softraid FDE (bcrypt-pbkdf)

Mode	File-Based Encryption (FBE)
38900	Google fscrypt (Argon2id)

Mode	Password Manager
26150	Mozilla key4.db SHA384
34301	KeePass AESKDF (KDBX v4)
36400	KDE KWallet 4.13+ (PBKDF2-HMAC-SHA512, Blowfish)
36410	KDE KWallet < 4.13 (SHA-1, Blowfish)

Mode	Private Key
17050	GPG (AES-OCB-128 (SHA-1($pass)))
24430	PKCS#12 (.pfx, .p12) (SHA1 + HMAC-SHA1)
24440	PKCS#12 (.pfx, .p12) (SHA256 + HMAC-SHA256)
36800	OpenSSH Private Keys (bcrypt-pbkdf)

Mode	Cryptocurrency Wallet
36000	BIP39 Passphrase Recovery (ASCII, P2SH/P2PKH/P2WPKH)
37700	Cardano Eternl Wallet (PBKDF2-HMAC-SHA512, ChaCha20)

Mode	RFID
36900	iClass Standard (DES + hash0 + iClass Cipher, dual-MAC)
36901	iClass Legacy (partial div_key + iClass Cipher, dual-MAC)
37000	MIFARE Ultralight C 3DES Key Recovery

6.1. New Hash Functions in Recipes

Hash-mode 4000 brings a whole set of hash functions that no earlier hashcat version could compute. You can use every one of them in a recipe, on its own or combined with everything else a recipe can do. Each one also has a raw hash-mode of its own, from 40084 to 40175, which hashes only the password. MD6-256 keeps its existing mode 34600:

Family	Recipe names
MD2	`md2`
SHA-0	`sha0`
RIPEMD	`ripemd`
RIPEMD-128 and RIPEMD-256	`rmd128`, `rmd256`
HAVAL	`hav128` to `hav256_5`
Tiger, Tiger2 and TTH	`tiger`, `tiger2`, `tth`
Snefru	`sne128`, `sne256`
PANAMA	`panama`
HAS-160	`has160`
RadioGatun	`radiogatun32`, `radiogatun64`
Whirlpool-0 and Whirlpool-T	`wrl0`, `wrl1`
MDC-2	`mdc2`

SHA-3 candidate	Recipe names
BLAKE	`blake224` to `blake512`
Blue Midnight Wish	`bmw224` to `bmw512`
CubeHash	`cube224` to `cube512`
ECHO	`echo224` to `echo512`
EDON-R	`edon256`, `edon512`
Fugue	`fugue224` to `fugue512`
Groestl	`groestl224` to `groestl512`
Hamsi	`hamsi224` to `hamsi512`
JH	`jh224` to `jh512`
Luffa	`luffa224` to `luffa512`
Shabal	`shabal224` to `shabal512`
SHAvite-3	`shavite224` to `shavite512`
SIMD	`simd224` to `simd512`
Skein-512	`skein224` to `skein512`

Every family except Fugue, Hamsi, RadioGatun, TTH and MDC-2 also has an HMAC form, such as `hmac_groestl256(salt, pass)`. A plain Groestl-256 hash, for example, is one command away:

```
$ ./hashcat -m 4000 --hash-recipe 'groestl256(pass)' hash.txt wordlist.txt
```

6.2. Changes to Existing Hash-Modes

- Mode 23800 (RAR3) uses a new built-in decoder for both coding paths and all 6 of the standard filters. It counts candidates that reach a construct it does not support. This also removes the C++ compiler requirement
- Mode 11600 (7-Zip) supports ZSTD compression. Hashes using it need libzstd
- Modes 22000 and 22001 accept hcxpcapngtool's 802.11r FT record types 03 and 04, alongside PMKID and EAPOL
- Mode 11000 accepts PrestaShop salts with lengths from 56 to 64 bytes. Its optimized kernels were removed, because they cannot handle these longer salts
- Modes 10000, 10900, 12000, 12100, 20200, 20300 and 20400 accept iteration counts of 8 digits, rather than the previously accepted counts of 6 digits
- Mode 31900 (MetaMask) reads and writes an optional `rounds=` field for vaults that use an iteration count different from the default of 5000 iterations
- GPG (CAST5 (SHA-1($pass))) accepts nonzero IVs and s2ktype 3
- JtR's dynamic formats run as hash recipes, see section 2.1. `--show`, `--left` and `--remove` keep JtR's original hash spelling

- GOST R 34.11-94 (6900), SHA-3 and Keccak (17300 to 18000), MD6-256 (34600) and vBulletin >= v3.8.5 (2711) had optimized kernels only. Without `-O`, they now run hash recipes and accept passwords of up to 256 bytes
- Mode 34300 is now called "KeePass Argon2 (KDBX v4)", to tell it apart from the new mode 34301
- Modes 8500, 14200, 16000, 22001, 27000 and 27100 declare ASCII-only passwords. Other plaintext is displayed as `$HEX[...]`
- The usage notices now cover 124 hash-modes and name the tool that extracts the hash

6.3. Extraction Tools

There are 8 new extraction tools in the `tools/` directory:

Tool	Hash-mode	Extracts from
`rar32hashcat.py`	23800	RAR3 archives
`gpg-ocb-aes2hashcat.py`	17050	GnuPG `private-keys-v1.d`
`routeros2hashcat.py`	38000	RouterOS backups and config stores
`pkcs12_to_hashcat.py`	24430, 24440	`.pfx` and `.p12` files
`geli2hashcat.py`	38700	GELI providers, disk images, backups
`krb5db2hashcat.py`	38400	Kerberos databases
`openbsd_softraid2hashcat.py`	38800, 38810	OpenBSD softraid CRYPTO volumes
`fscrypt2hashcat.py`	38900	fscrypt protector files

Another 3 extraction tools were updated as well. `mozilla2hashcat.py` reads the `key4.db` files of Firefox 146 for mode 26150. `metamask2hashcat.py` reads vaults stored under `persist-root` as well as those under `persist-KeyringController`. And `bitwarden2hashcat.py` reads mode 23400 key hashes and KDF iterations under the active account.

7. Performance

This section is always one of our favorites to write. Fine-tuning hashcat is a lot of fun. In this release, the gains come from 3 different directions: individual hash-mode optimizations, a new autotune, and less memory spent on buffers that were never used. As always, the results depend on the device, the runtime and the workload.

7.1. Faster Cracking

Hash-mode	Hardware	Gain
PKZIP, 17200	RTX 4090	13.8x
PKZIP, 17200	RX 7900 XTX	4.4x
PKZIP, 17200	Arc A770	12.1x
Raw private keys, 30901, `-a 3`	RTX 4090, RX 7900 XTX, Arc A770	32x to 58x
Raw private keys, 30903, `-a 3`	RTX 4090, RX 7900 XTX, Arc A770	5x to 18x
Raw private keys, 30902, 30905, 30906	RTX 4090, `-a 3`	35x to 45x
RACF KDFAES, 14200	Arc A770	10.9x
RACF KDFAES, 14200	RX 7900 XTX	+34%
Radmin3, 29200	RTX 4090, RX 7900 XTX	8.2x to 11x
Electrum Wallet, 21800	Arc A770	6.6x
QNX SHA-512, 19200	Arc A770	4.2x
NetNTLMv1, 14000, `-a 3`	RTX 4090	+17% to +24%
NetNTLMv1, 14000, `-a 3`	Arc A770	+53% to +86%
NetNTLMv1, 14000, `-a 3`	Metal	2.9x to 3.8x
LM, 3000, `-a 3`	Arc A770	8.7x
LM, 3000, `-a 3`	Radeon Pro W5700X, Metal	2.4x to 2.6x
LM, 3000, `-a 3`	RTX 4090	+23%
descrypt, 1500, `-a 3`, 64 salts	RTX 4090, RX 7900 XTX, Arc A770	1.8x to 4x
descrypt, 1500, `-a 3`, one hash	Arc A770	2.5x
8 AES modes, 16600 to 23003	RTX 4090	1.2x to 1.9x
8 AES modes, 16600 to 23003	RX 7900 XTX	1.3x to 2.6x
8 AES modes, 16600 to 23003	Arc A770	1.1x to 1.9x
RAR3-hp, 12500	RTX 4090, RX 7900 XTX, Arc A770	1.2x to 2.4x
SAP CODVN H isSHA512, 40078	RTX 4090	2.1x
Mozilla key3.db, 26000	RX 7900 XTX	1.9x
Mozilla key3.db, 26000	Arc A770	1.7x
PBKDF2-HMAC-SHA512 modes	Arc A770	1.7x to 1.9x
JWT, 16500	Arc A770	2.4x
DPAPI masterkey file v2, 15900	Arc A770	2.3x
Perl Mojolicious cookie, 16501	Arc A770	1.9x

Hash-mode	Hardware	Gain
Perl Mojolicious cookie, 16501	RX 7900 XTX	1.4x
KeePass Argon2, 34300	Arc A770	1.8x
SAP CODVN F/G, 7800, 7801	RX 7900 XTX	1.6x
PDF 1.7 Level 8, 10700	RX 7900 XTX	1.5x
Whirlpool, 6100	Arc A770	1.5x
TrueCrypt boot-mode, 29341	Arc A770	1.5x
MS-AzureSync PBKDF2-HMAC-SHA256, 40037	Arc A770	1.5x
Streebog modes, HIP	RX 7900 XTX	+7% to +57%
VeraCrypt Streebog modes, HIP	RX 7900 XTX	+16% to +85%
md5crypt, 500, optimized	RTX 4090, RX 7900 XTX, Arc A770	+9% to +37%
MD5, 0, optimized combinator	AMD, NVIDIA, Intel	+10% to +92%
RC4 modes 9700, 9800, 10500	RTX 4090	+21% to +33%
SM3, 31100, `-a 3`, optimized	RTX 4090	+19%
scrypt modes 8900, 22700, 27700, 28200	RTX 3070	+18%
scrypt modes 8900, 22700, 27700, 28200	RTX 4090	+11%
Argon2 modes, AMD OpenCL	RX 7900 XTX	+2% to +79%
41 UTF-8 to UTF-16 modes	GPU	+2.3% to +9.0%
MD5, 0, `-a 3`, optimized	Apple M3 Max	+8.7% to +10.2%
NTLM, 1000, `-a 3`, optimized	Apple M3 Max	+9.1% to +9.5%

Most rows compare `tools/benchmark_deep.pl` runs of v7.1.2 and v8.0.0 with one hash. Not every mode got faster. Some run slower than in v7.1.2, most of them on the Arc A770.

Small wordlists with many rules now keep every device busy. Before, the last base words were assigned to the first devices a full wave at a time, and the other devices sat idle. Slow hashes also ran only one rule per launch, so their launches stayed small. With 50,000 words and 150,000 rules, mode 12700 now finishes in 17 seconds on 2 RTX 4090 cards, instead of 60 seconds on one of them. The usual workaround, one hashcat instance per card, took about 31 seconds. On 2 RX 7900 XTX cards, the same run now takes 14 seconds instead of 51 to 57 seconds.

A few more improvements don't fit into one row:

- Sorting candidates by length raises throughput by 50% to 200% in 19 hash-modes that opted in. `--length-sort-disable` turns it off
- Autotune now reaches the full accel of 1024 for slow hashes. It used to stop at 711, which held back the modes that run best at full accel

- Short launches no longer leave the GPU idle while the host reads back the crack count, which cost up to 0.8 ms per launch on CUDA
- A slow hash batch smaller than one launch now spreads over every compute unit, which speeds up small wordlists and the end of every wordlist
- Scrypt TMTO values above 2 raise the throughput of underfilled high-memory GPUs by 5% to 46%
- On an RTX 4090, rule `Z` with counts of 3 or more runs about 40% faster in optimized kernels. On the same card, rule `z` with counts of 2 or more runs up to 2.3 times faster in pure kernels
- Candidate preparation now overlaps the device work, and piped input now arrives in blocks. These changes help attacks that use multiple devices
- Large rule files load faster and use less memory, and rule processing under `--stdout` uses every host core

7.2. Autotune

Autotune used to search for the right launch size by running a kernel at every step. For slow hash-modes in particular, this meant waiting for many launches at sizes the attack would never use. In hashcat v8.0.0, autotune starts from what the device reports:

Autotune	v7.1.2	v8.0.0
Thread count	Searched with timed launches	Occupancy query on CUDA and HIP
Measuring	A launch at every step	A few probes, each sized by the last
Launch size	Fits a target time	Fits the target time in whole waves
Same-model devices	Each one tuned	Tuned once, shared
Repeated rounds	Tuned again	Results reused

From a user's perspective, this means less waiting before the attack starts, and the attack avoids undersized launches. Across equivalent devices and association rounds, autotune takes 32% less time. Reusing the results across wordlist queues, mask files and increments helps even more:

Mask file	Before	Now
30 masks	71 s	6 s
200 masks	243 s	4 s

As a consequence, hashcat no longer reads the workload profile. Every run uses profile 3. See section 5.4 for details. If you want to see what autotune does, set `HASHCAT_AUTOTUNE2_VERBOSE` to print its probe times, the cost model and the chosen launch.

7.3. Memory

- Removing an unused event backlog saves 160 MiB of memory per attack
- Mask attacks that generate their candidates on the device no longer allocate the device and host buffers for host-built candidates, which they never use

- The compressed candidate buffer is sized according to the maximum password length of the hash-mode. With `-m 0 -a 0 -O`, the host allocation falls from 2583 MB to 1305 MB
- Register-spill reserves are sized for the resident work items, which cuts the mode 0 reserve by 99% on an RTX 4090
- Devices with unified memory reserve only the host buffers they need, instead of half their pool
- Argon2 buffers follow the attack size instead of the self-test size
- The Markov charset buffer is sized by the longest mask instead of all 256 positions. A mask with 8 characters now needs 2 MB instead of 67 MB, on the host and again on every device

8. User Experience and Platforms

8.1. Less Waiting

Nobody likes to wait for an attack to start. These are some of the places where hashcat v8.0.0 saves you time:

Before and after	Before	Now
Mode 2600, first Intel CPU run	12 min	44 s
Mode 1410, first Intel CPU run	10 min	3 min
`--keyspace` probe with 4 characters	543 ms	22 ms

Almost all of the time in these 2 Intel CPU runs is spent compiling the kernels. The cracking speed afterwards is unchanged.

Startup, before the first candidate	Time saved
Cold mode 0 start, iGPU skipped	26%
Cold kernel builds, ROCm 7.0.1	42%
Backend info, ROCm 7.0.1	70%

Every run also ends about 0.6 seconds sooner. A `--keyspace` probe across 200 masks runs 9 times faster. And excluded CUDA devices no longer create a context just to query their memory.

8.2. Control and Feedback

You can now move a running attack through its keyspace. Press "r" or "a", the left and right arrow keys, or "<" and ">" to rewind or advance. If you hold a key, it moves faster. You can also press "e" to change the `--runtime` limit while the attack is running. And paused attacks now respond sooner to quit and resume.

The messages got clearer as well:

- Parse errors show the failing text and the expected format, and name hashes that were passed on the command line
- After an error, one more line names where the cause lies, such as the attack command or the task

- The status line no longer waits for the running kernel to build its candidate preview, which delayed it by up to a whole launch, most of all in PCFG attacks
- The `--outfile-check-dir` scan before the attack reports its start and end and the hashes it removed, and gets its own row in `--task-time-breakdown`
- Hashcat notices when the hash file and the wordlist are swapped, names both files, and stops before it loads anything. Autodetect no longer picks a hash-mode because a few words of a wordlist happen to parse as hashes
- Initial wordlist scans show their throughput in MiB/s
- The status labels wordlists as feeds, and shows "Host Generator + PCIe" when candidates are not amplified on the device
- Large device sets get summarized candidate windows, bridge speeds and grouped temperatures, instead of one line per unit on every update
- The diagnostics show NVIDIA throttling reasons and the accepted `-u` ranges. `HASHCAT_MEMORY` shows the device memory budgets, and `-I` shows cache sizes and bridge units
- Tab completion covers the PCFG, table and feed attacks and wordlists in masks, `--lookup`, `--cache-path`, custom charsets 5 to 8, and all debug modes

8.3. New Options

Option	What it does
`--json`	Emits every structured output as JSON, including the benchmark
`--benchmark-json`	Emits only the benchmark as JSON, and turns on `-b`
`-B`, `--benchmark-pure`	Benchmarks the pure kernels. `-b` still benchmarks the optimized ones
`--hash-recipe`	Describes the hash for mode 4000, see section 2.1
`--lookup`	Finds a candidate's position in an attack, see section 1.7
`--rules-concat`	Joins rule files into one sequence
`--hash-copy`	Keeps the input spelling of cracked hashes in outfiles, potfiles and status
`--encrypt-with-pubkey`	Encrypts recovered plaintexts with an RSA key of at least 3344 bits
`--task-time-breakdown`	Shows stage times and peak memory
`--pipeline-stats`	Shows per-device timing for feeding, copies, transfers and loops
`--cache-path`	Moves all caches, see section 4.6
`--brain-feed`	Adds candidates from stdin to a running brain server
`--stdin-count`	Shows progress and a finish time for stdin attacks, see section 1.6
`--length-sort-disable`	Turns off sorting candidates by length

`--encrypt-with-pubkey` deserves a word of its own. It is meant for cases where the recovered passwords must not be stored in plaintext. Hashcat loads OpenSSL 3 only when you use this option. It hides the plaintext in the status, and rejects plaintext output and restore options. See `docs/hashcat-`

`encrypted-plains.md`.

8.4. Benchmark

The default `-b` list now includes yescrypt, gost-yescrypt, scryptcrypt, KeePass KDBX4 and Kerberos 5 etype 23 AS-REP. MyBB replaces vBulletin, and the redundant md5($pass.$salt) modes were removed. The results now name the kernel that was used. The header of each result shows the real cost parameters of Argon2 (m, t, p), scrypt (N, r, p), yescrypt (N, r, p, t) and bcrypt (cost), instead of a single iteration count.

8.5. Platforms

On Apple Silicon Macs with macOS 26, hashcat now uses Metal 4, the newest version of Apple's interface for GPU compute. It caches the compiled pipelines under `cache/kernels`, so later starts skip the compile step. Older Macs and other Metal devices keep the Metal 3 path, and `-I` shows which version each device uses. Builds on Apple Silicon are now native arm64 by default. If you need one binary for both Apple Silicon and Intel Macs, `MACOS_UNIVERSAL_BINARY=1` still builds it.

Hashcat now also builds for Android, where it runs under Termux, and for little-endian LoongArch, PowerPC and MIPS systems. On OpenBSD, it builds with the compiler of the base system, so no extra compiler package is needed.

Hashcat now finds the newest NVRTC, HIPRTC and HIP runtime libraries on the system by itself. On Linux, this means the runtime packages are enough. HIP also picks its interface from the runtime it loads, so the same hashcat binary runs on AMD's ROCm and on Hygon's DTK, the HIP stack for Hygon's accelerators.

On Windows, hashcat now writes UTF-8 to the console, so you no longer need `chcp 65001` to see non-ASCII characters. Paths and arguments keep their Unicode characters everywhere, and paths longer than 260 characters work. The Windows binaries also carry native version information.

8.5.1. AMD GPUs Under Mesa Rusticl

Rusticl is the OpenCL driver of Mesa, the open source graphics stack that most Linux distributions ship. Hashcat now runs on AMD GPUs under rusticl, with the same temperature, fan and clock readings as under AMD's own driver. It is already a real alternative. On an RX 7900 XTX, rusticl runs Streebog 62% faster than AMD's own OpenCL.

Most other hashes are still slower under rusticl, and the main reason is rotations. Hash functions rotate their words by fixed amounts all the time, and Mesa's compiler turns each of these rotations into 2 instructions instead of one. We sent Mesa a fix, merge request 44096, which is waiting for review. With this fix, on the same card, SHA-256 runs 36% faster, SHA-1 32%, NTLM 22% and SHA-512 13%. With the fix applied, rusticl overtakes AMD's own OpenCL on NTLM and MD4. Together with a change on our side that is already in this release, Whirlpool runs at 2206 MH/s, against 1875 MH/s on AMD's own OpenCL.

9. Bug Fixes

With 306 fixed bugs, we can't list them all here. The full list is in `docs/changes.txt`. But some of them deserve your attention.

9.1. Passwords That Older Versions Missed

In the following situations, older versions of hashcat missed passwords:

Hash-modes	What was missed
22000, 22001 (WPA)	PMKIDs from PSK-SHA256 (AKM 6) networks
22000, 22001 (WPA)	EAPOL frames over 256 bytes, such as M2 with vendor RSN data
2500, 2501, 22000, 22001	Nonce error correction on key version 3 handshakes
22 UTF-16 modes, such as NTLM and DCC	Non-ASCII passwords in pure mask attacks on most CPUs
7700, 7701, 7800, 7801 (SAP)	Every crack under a rule that changes the length
11700, 11800, 28000	Every crack under a rule that changes the length
1500, `-a 3`	Lists with several hashes sharing one salt
72000, 73000	Most of a list, when a salt had several hashes
70000, 73000, 74000 in `-a 9`	Every hash but one
14500	Every hash but the first, in lists of 128 or 192 bit keys
23001, 23002, 23003 (SecureZIP)	All but one of the hashes that share an IV prefix
34200, 34201, 34211 (MurmurHash64A)	Shorter candidates in pure combinator and hybrid attacks
200 (MySQL323)	Passwords containing a space or a tab
6900	The empty password
Argon2 modes	Hashes of mixed parallelism, outside AMD and NVIDIA OpenCL
Argon2 modes in `-a 9`	Hashes in lists with mixed memory cost
8300 (DNSSEC)	Domains written with uppercase letters
7800, 7801 (SAP)	SHA-1 messages longer than 119 bytes
SNMPv3 modes	Engine IDs shorter than 13 bytes, or 18 to 32 bytes long
32500 on CPUs, vector width 4	3 of every 4 candidates
12150	Candidates at vector widths above 1, the CPU default
Apple's drivers for Intel GPUs	Every odd work item in `-a 0` pure kernels

If you attacked one of these hash-modes with v7, it may be worth running the attack again.

9.2. Security Fixes

These fixes close security issues in v7.1.2 and earlier versions:

Issue	What was fixed
CVE-2026-68766	Options injected through a restore file, see section 5.1
CVE-2026-68767	An input line that filled the read buffer wrote a byte past it
CVE-2026-68768	Long user names, hashes or plaintexts overflowed the outfile line
Brain server	Any host that reached its port could stop it, without the password

The brain server issue affects every release since v5.0.0.

9.3. Other Notable Fixes

- Hashcat exited with 0 when autotune failed on every device. It now exits with -10
- On Windows, libraries loaded by name were searched in the working directory. Libraries beside `hashcat.exe` still load
- Child processes now close hashcat's file handles and the brain socket, so Python workers no longer inherit the potfile or a live brain connection
- Kernel cache writes now use a temporary file and an atomic rename, so another process can no longer read a partially written kernel
- Kernel cache keys now include the source content, so modified kernels are recompiled
- Combinator and hybrid output no longer truncates plaintexts at 256 characters
- The Metal backend released neither the command buffer nor the encoder of a launch, so memory grew for as long as a run lasted
- Windows paths and arguments lost their Unicode characters, which broke install paths, masks, rules and potfiles

9.4. By Area

64 of the fixes are memory safety fixes, most of them in the hash parsers:

Area	Fixes
Memory safety: hash parsing and encoding	41
Memory safety: candidates, rules and masks	11
Memory safety: services and interface	12
Hash-modes and kernels	67
Backends: OpenCL, tuning and memory	28
Interface and status	28
Backends: bridges and hardware monitoring	26
Attack-modes and candidates	18
Tools and tests	16
Hash lists, potfile and outfile	15
Backends: Metal and resource lifecycle	13
Windows	13
Build and packaging	10
Brain	8

10. Under the Hood

This section is mainly for package maintainers, plugin developers, and anyone who is curious about how hashcat works on the inside.

10.1. Build System

Hashcat is now a shared core library. The hash-mode modules, the bridges and the feeds all link to libhashcat, and the core exports a defined plugin API. Host hash and cipher functions remain available to plugins, even when no bundled plugin uses them. Plugins can also receive bridge parameters, describe multi-part units, and use the core's device allocation, compilation and launch functions.

The bundled copies of zlib, LZMA, UnRAR and xxHash are gone. Hashcat now uses the system zlib and LZMA, and its own RAR3 decoder in place of UnRAR. It also replaces xxHash with paw64, our own non-cryptographic 64 bit hash. It passes all 188 tests of SMHasher3, the standard test suite for hash functions of this kind, and it runs 1.6 times faster than xxHash. As a side effect, a C++ compiler is no longer needed.

Release builds use link-time optimization. The shared core also makes the unpacked release about 73% smaller. Builds are also reproducible: with `SOURCE_DATE_EPOCH` set, any 2 builds of the same commit are identical.

Most of the size goes away in the modules. In v7.1.2, every module carried its own copy of the shared code. Now they all link to one core library. There are 728 modules instead of 582, and they take a fraction of the space:

Unpacked size	v7.1.2	v8.0.0
Linux modules	183 MB	14 MB
Windows modules	167 MB	32 MB
Core library, Linux	-	3.9 MB
Core library, Windows	-	3.9 MB

For package maintainers:

- Development installs include the public plugin headers and an unversioned `libhashcat.so`
- `make install` includes the feeds, the rules, the PCFG rulesets and the Python folder
- The `hashcat_binaries` container cross-builds the release, and there are new Ubuntu 26.04 CUDA and AMD images. Patches in `docker/patches` fail the build if they do not apply
- CI builds FreeBSD and Linux ARM64 on every push, and OpenBSD, NetBSD and DragonFlyBSD weekly. `tools/test_package.sh` checks every artifact before upload
- `docs/packages.md` lists the shipped files, the runtime dependencies and the build settings

10.2. Kernel Sources

`inc_common.cl`, which every kernel includes, went from about 39k lines to about 5.4k lines. About 12k hand-typed indices were replaced by a set of 3 proven macros. Kernel selection now lives in one place, which also freed 16 option bits for future flags. Fast kernels reserve constant memory only for the data the attack reads, so mask attacks regain 28 KB and combinator attacks 32 KB of the 64 KB budget.

Every kernel computes exactly what it did before, but it builds a lot faster:

Kernel build, RX 7900 XTX	Before	Now
Empty kernel	3.0 s	0.2 s
Mode 0, `-a 3` pure	7.6 s	0.6 s
Mode 500, optimized	7.5 s	0.4 s
Mode 1700, `-a 3` pure	37 s	3.5 s
OpenCL startup	14 s	8.2 s

The OpenCL hash and HMAC helpers now read only the words within the input length, so plugins no longer need to pad the input to a full block.

10.3. Test Framework

The test framework no longer needs Perl. It is now written entirely in Python, from the test modules to `tools/test.py`, which replaces the old shell scripts. That script runs the test, edge and coverage suites, and it can run them in parallel with `-j`. The test suite needs Python 3.10 or newer, and the packages it uses are pinned in `tools/requirements.txt`. A failing run keeps its hash file and dictionaries in the log folder, so you can reproduce it from the uploaded log.

The generated test passwords used to be pure ASCII. They now include multibyte UTF-8 characters, so the tests also cover passwords with characters outside ASCII. Many real passwords contain these characters.

Test modules	v7.1.2	v8.0.0
Count	462	544
Language	Perl	Python

Coverage	Modes
Real containers	105
Independent oracles	29
`-M` quick selection	24

The edge suites cover every attack-mode, kernel type, target type and vector width of a hash-mode, including the PCFG, feed and association attacks and wordlists in masks. The container tests create real LUKS, TrueCrypt, VeraCrypt, GPG, PKZIP, RAR, 7-Zip, PDF and OpenSSH files and crack them. The `--bridge` option cracks each mode's own test oracle through the Python bridge, which checks the parser against the reference implementation. Multi-hash runs now give 2 separate pairs of hashes a shared salt wherever the mode allows it. This testing found real bugs in shipped hash-modes. See section 9.1 for details.

`tools/test_options.py` tests every option on its own and in pairs. It fails when a run crashes, starts to hang, or accepts a combination it used to refuse.

On top of that, a weekly CI run covers the optimized and the pure kernels, the Python bridge, and the containers.

10.4. Documentation and Licensing

New documents in this release:

Document	What it covers
`SECURITY.md`	How to report a vulnerability
`CONTRIBUTING.md`	Rules and code style for contributions
`AGENTS.md`	Guidance for an AI coding agent
`docs/hashcat-requirements.md`	The minimum versions of every backend
`docs/packages.md`	Shipped files and dependencies
`docs/hashcat-rust-plugin-quickstart.md`	Rust bridges and feeds
`docs/plugin-development-workshop/`	Exercises for writing a hash-mode
`docs/hashcat-pcfg.md`	PCFG attacks
`docs/hashcat-table.md`	Table attacks
`docs/hashcat-association.md`	Association attacks
`docs/hashcat-hash-recipe.md`	Hash recipes
`docs/hashcat-generic-attack-mode.md`	Using feeds
`docs/hashcat-generic-attack-mode-development-guide.md`	Writing feeds
`docs/hashcat-compression-libraries.md`	When zlib, liblzma or libzstd is needed
`docs/hashcat-encrypted-plains.md`	`--encrypt-with-pubkey`
`docs/hashcat-task-time-breakdown.md`	`--task-time-breakdown`

`README.md` was revised as well. Issue reports on GitHub now use forms, with separate fields for the version, the command line, the output and the operating system.

We added the license texts of the bundled Argon2, scrypt-jane, yescrypt and PPMd code, and of the Windows liblzma and libzstd libraries.

10.5. Environment Variables

A few environment variables let you look inside hashcat or switch a new behavior back off. Most users will never need them, but if you run into trouble, they help us help you:

Variable	What it does
`HASHCAT_DEVICE_MEM_LIMIT`	Device memory budget per process, in MiB
`HASHCAT_HOST_MEM_LIMIT`	Host memory budget per process, in MiB
`HASHCAT_MEMORY`	Prints how each device's memory budget was spent
`HASHCAT_AUTOTUNE2_VERBOSE`	Prints the autotune probes, cost model and launch
`HASHCAT_PIPE=N`	Prints `--pipeline-stats` every N launches, not 50
`HASHCAT_PIPE_SYNC`	Prepares candidates in step with the device
`HASHCAT_CLSHARE=0`	Gives OpenCL clones separate contexts and programs
`HASHCAT_SALT_INNER=1`	Tests each amplifier item against all salts first
`HASHCAT_FORCE_NO_INLINE`	Builds kernels with `FORCE_NO_INLINE`
`HASHCAT_NO_AMD_FREE_MEM=1`	Uses the desktop reserve, not AMD's free memory query
`HASHCAT_OPTION_DEBUG=1`	Prints the rule behind each option check, see 5.10
`HASHCAT_MODULE_CACHE_DISABLE=1`	Turns off the cache of module options

11. Plans for the Next Updates

- We will redesign the main loop for salted hash lists in v8.1.0. Today, hashcat runs many candidates against one salt, then moves on to the next salt. That order dates from a time when large hash lists were mostly unsalted, such as raw MD5. Today, large lists are mostly salted, and most salts hold a single hash. If many salts run against the same candidates instead, a salt is likely to crack early and can drop out of the list right away. This means the rest of the run gets cheaper. We will measure both orders on GPUs and make the loop flexible enough to choose between them
- We plan to support cloud FPGAs, such as AWS EC2 F2 instances, so you can rent the boards by the hour instead of buying them
- We will change how module hooks work. Today, each module lists every hook in `module_init()` and sets the ones it does not use to `MODULE_DEFAULT`. So a new hook has to be added to every one of the more than 700 modules, as this release does for `module_register_options()`. With hashcat v8.1.0, adding a hook will no longer require touching every existing module

12. List of Contributors

We would like to thank the following people who contributed to hashcat 8.0.0:

Name	Name	Name	Name	Name
0xVavaldi	4RH1T3CT0R7	ahmed-alnassif	albaropereyra22	AlbertVeli
AmmoniaAvenue	AndrewWestberg	aurabirb	b8vr	bandrel
bbrtj	bernardladenthin	bishwabikash	blurbdust	Br0kenUK
c0rs	carlosrodfern	Chick3nman	Coontzy1	dev-goalz
Deztryoz	DJStompZone	Foxushka	fse-a	fukc-gihtub
golem445	gqxw2tjbxb-prog	holly-o	hops	Igloo-Garage
ImArjunJ	jeffmcjunkin	jengelh	jessevz	jsteube
JustGuardian	kozmer	lql341	Lyapsus	maede97
matrix	mimi89999	MisterMaluco	mohammedjhaish	MRX-72
Naster17	npinto	Oblivionsage	PenguinKeeper7	pwrbob
racerxdl	raulperdomo	RealEnder	roycewilliams	shadowshan125
Shooter3k	skippdot	Spoonman1091	Suyun114	Svenja240602
thatux	toor11	valpackett	VladDBA	whoot
zyronix				

These are GitHub account names. Each profile page is at https://github.com/ followed by the name.

Some of them shaped this release in particular:

- **blurbdust** opened the first pull request for bcrypt on FPGAs through LiteX, and added the YPCB-00338-1P1 board and BPI flash support to openFPGALoader, LiteX and litex-boards, which hashcat's FPGA support builds on
- **Br0kenUK** had the idea behind the hints ruleset, and found the folded AES key schedule tables that speed up 8 hash-modes by up to 93%
- **Chick3nman** added 11 hash-modes, among them GPU yescrypt, RACF, iClass, MIFARE Ultralight C and SAP CODVN H, three new rule functions, and fixes for more than a dozen crashes, overflows and hangs
- **jessevz** added the KWallet hash-modes, let brain clients reserve keyspace without forcing `-S`, made the candidate queues cheaper, and wrote the `--lookup` tests for every attack-mode it answers
- **jsteube** found how to make PCFG seekable and added the PCFG engine, the hash recipe engine, the FPGA ecosystem, the table attack, support for `?w`, the new `--lookup` feature and the feed plugins, and reviewed and merged the pull requests
- **matrix** sent 136 pull requests across the whole release: the hashcat-toolchain build container and many CI and platform builds, Metal 4 support, `--rules-concat`, 4 new hash-modes, speedups and fixes for the PCFG device engine, and a long series of memory safety and leak fixes

- **PenguinKeeper7** added the WordPress bcrypt, phpass(md5) and Symfony Legacy hash-modes and the `--hash-copy` option, and sped up MD5 in straight and combinator attacks
- **Shooter3k** kept a fork whose ideas came into this release, among them rules in mask and hybrid attacks, moving a running attack through its keyspace, the task time breakdown, multibyte rule operands, UTF-8 output on Windows and many long-standing fixes
- **thatux** wrote `tools/test.py` and moved every test module to Python, built the real test containers and the sanitizer and fuzzing builds, and fixed the bugs they found. thatux also added `--json`, `--seekdb-path`, length-sorted candidates and the usage notice of every module

13. Final Words

Hashcat v8.0.0 brings several major features, and honestly, they could have filled several releases. But each of them mattered from a different angle: large-scale distributed work, a future-proof foundation with generator plugins, a refactored core that is easier to package, and a long list of bug fixes. They all came in at the same time, and at a pace that left no room to slow down for a release in between.

I was asked whether one of the new features is the most important. And I can actually name them: PCFG and hash recipes. Both are for the users, not for the developers.

PCFG with a mask on top is better than a plain mask. A mask attack runs faster, but the probability order of PCFG easily makes up for that, by more than I could put a number on.

But PCFG matters far beyond masks. PCFG itself is not new, but in hashcat it runs on the GPU. And because hashcat can seek anywhere in its keyspace, you can now split a PCFG attack across many machines, like any other attack. If you work with hashcat every day, a PCFG attack is most likely what you will start once your wordlist and rule attacks are exhausted. You can leave it running unattended, with the good feeling that it is not a brain-dead brute force.

Hash recipes are so simple and so fast that they may well be the next big thing in module development. We may need to extend the language to cover even more use cases, but what you can already do with it is hard to put into numbers as well. Hashcat no longer supports "only" 700 hash-modes. It supports an unlimited number of them.

I want to sincerely thank all contributors, testers, and supporters who helped bring this release to life. You know who you are. Thank you.