Open-Source Tools for Bitcoin Wallet Password Recovery (2018–2023)

Recovering a lost Bitcoin Core **wallet.dat** password can be challenging, especially for an older wallet file (circa 2010) using strong encryption. Fortunately, several open-source tools released or actively developed in the last five years can aid in this effort. This report highlights tools that leverage **AI/Machine Learning** for intelligent password guessing, as well as advanced **rule-based** and **brute-force** methods. We also note GPU support and optimization strategies for each tool, and discuss how they can be applied to cracking a 2010-era Bitcoin Core wallet password.

BTCRecover – Wallet Password Recovery Suite

BTCRecover is a specialized open-source tool designed for cryptocurrency wallet password and seed recovery. It is well-suited for Bitcoin Core wallet.dat files and similar wallets. BTCRecover is written in Python and geared towards scenarios where you remember parts of the password or have hints about its structure. Key features include:

- **Targeted Search with Heuristics:** Allows using *token lists* (pieces of the password you recall) and wildcards to fill in unknown parts
  github.com
  . For example, if you remember a phrase or pattern from 2010 that you likely used, you can supply those tokens and let BTCRecover try combinations around them. It also supports simulating common **typos or capitalization mistakes** when testing candidates
  github.com
  .
- **Multithreading and GPU Acceleration:** BTCRecover can utilize multiple CPU threads to speed up the search, and it offers experimental OpenCL-based **GPU acceleration** for Bitcoin Core and similar wallet types
  github.com
  . This means you can harness a GPU to accelerate the wallet's key derivation checks, which is crucial for an old wallet with a slow hashing routine.
- **Multi-Platform & Offline Use:** The tool is cross-platform (Windows, Linux, macOS) and completely open-source
  github.com
  . It supports an "offline mode" using **wallet extracts**, where you extract necessary data from the wallet.dat (using provided scripts) and then run the recovery without exposing your actual wallet keys
  github.com
  . This is useful if you want to utilize cloud GPUs securely.

Application: For a 2010 wallet, BTCRecover lets you leverage any memory of how you formed passwords back then. For instance, you could input an old username, favorite phrase, or pattern you often used, and set wildcards for the parts you don't recall. BTCRecover will systematically try variations around those clues. The typo simulation can catch minor mistakes (like an extra character or swapped case) that might be the reason the password isn't working. By running on GPU or across multiple CPU threads, it can significantly cut down the time to test each candidate. In practice, you would run BTCRecover with your wallet.dat (or an extracted key file) and a token file of hints; the tool will output the correct password once found. Its focus on guided recovery makes it a top choice if you have **partial knowledge** of the passphrase.

Hashcat – High-Speed GPU Cracking Tool

Hashcat is a general-purpose password recovery tool often cited as *"the world's fastest and most advanced password recovery utility."* It supports hundreds of hash algorithms and has multiple attack modes

github.com
. In the context of a Bitcoin Core wallet, Hashcat can be used to perform a **brute-force or rule-based attack** on the wallet's passphrase once you extract the relevant hash from the wallet file.

- **GPU Accelerated Cracking:** Hashcat can harness one or multiple GPUs (as well as CPUs or hardware accelerators) to attempt passwords in parallel
  github.com
  . This is critical for wallet.dat files, which use strong key derivivation functions. Hashcat has a built-in hash-mode for Bitcoin Core wallets (hash mode **11300** for wallet.dat passphrases)
  onlinehashcrack.com
  , so it knows how to verify each guess against the wallet's encryption. Using GPUs dramatically increases the trial rate compared to CPU-only tools – though the wallet's deliberate slowdown (tens of thousands of SHA-512 iterations) means even GPUs will test only a few hundred to few thousand passwords per second
  openwall.com

  openwall.com
  .
- **Multiple Attack Modes:** Hashcat supports five main attack modes, including straight dictionary attacks, combination attacks, **mask attacks** (brute-forcing with patterns), hybrid attacks, and rule-based modifications
  github.com
  . For example, you can brute-force with a **mask** if you know the password format (e.g. "8 characters, ends with a year"), or use a dictionary of guesses and apply **rules** to mutate them (adding symbols, changing case, leetspeak, etc.). This flexibility lets you craft an attack that reflects how passwords were created back in 2010. You might start with a

dictionary of common passwords or phrases from that era and add rules to append "2010" or substitute letters with numbers.

- **Optimizations and Distributed Cracking:** Hashcat is highly optimized at the kernel level for each algorithm, and it includes features like session checkpointing (to resume interrupted runs) and the **"Hashcat Brain"** to avoid duplicate guesses across distributed systems. It can be scaled out to multiple machines; for instance, using a platform like Hashtopolis or cloud GPU instances, you can distribute the workload. Hashcat's built-in support for distributed cracking coordination
  github.com
  helps ensure each node works on a unique portion of the keyspace. This is useful if the password search space is huge – you can farm out chunks of it to different GPUs.

Application: To use Hashcat on a wallet.dat, you first extract the encrypted hash from the file (using a tool like `bitcoin2john.py` – part of John the Ripper – or an online extractor)

onlinehashcrack.com

onlinehashcrack.com
. Then you run Hashcat with mode 11300 against that hash. For example, you might start with a **dictionary attack**: take a large list of candidate passwords (from leaked databases or wordlists of common passwords) and run Hashcat to test each against the wallet. You can apply **rules** to this dictionary to cover variations (e.g. try capitalizing the first letter, add "123", etc.). If you have an idea that the password was not a common word, you could instead use a **mask attack** to brute-force character by character. For instance, if you think it was 10 characters with numbers and letters, you could set a mask like `?a?a?a?a?a?a?a?a?a?a` (any letter/number symbol for each position) and let it run – though this is only feasible if the length or character set is limited due to the slow hash. Hashcat excels when combined with the right strategy: it provides the raw power (thanks to GPUs and optimized code) to try millions or billions of candidates, but you must feed it **intelligent guesses**. For a 2010 wallet, one might use Hashcat's speed to exhaust likely patterns (like "word + 1-2 digits") and to quickly test all candidates generated by AI or rule-based tools (like those from PassGAN or PCFG, discussed below).

Source: Hashcat is open-source (MIT licensed) and supports GPU acceleration and distributed cracking

github.com
. It includes a dedicated hash mode for Bitcoin Core wallets (using the wallet's salted SHA-512/AES encryption)
onlinehashcrack.com
. Its reputation as *"the world's fastest...password recovery tool"* comes from its highly optimized GPU kernels and support for multiple attack modes
hashcat.net

github.com
.

John the Ripper (Jumbo) – Offline Cracker with Rules

John the Ripper (JtR) is another powerful password cracking tool, well-known in the security community. The "Jumbo" edition is an enhanced open-source version that supports many hash types (including Bitcoin wallet encryption) and various cracking modes. John is often used for its **rule-based** cracking and its ability to integrate with custom scripts.

- **Format Support and CPU Cracking:** John's Jumbo version can crack Bitcoin Core wallet passwords by using the `bitcoin2john.py` script to extract the hash and then running John in "bitcoin" format. It supports hundreds of hash types and runs on many OS, with both CPU and some GPU support
  github.com
  . However, for the Bitcoin wallet hash type, John currently leverages **CPU (OpenMP)** and does *not* have GPU acceleration for this particular format
  openwall.com
  . This means John will test far fewer passwords per second compared to Hashcat on a GPU. For example, on a 4-core CPU, John might only test on the order of ~200 passwords per second for a wallet.dat hash (because the wallet uses ~200k SHA-512 iterations)
  openwall.com
  . This slow speed is due to the wallet's intentionally slow key derivation – a security feature that unfortunately makes brute forcing time-consuming.
- **Rule-Based and Markov Attacks:** John shines in its **flexible attack strategies**. It has a rich rule engine (similar to Hashcat's) and also implements **Markov mode** and **incremental mode** to systematically generate guesses in likely order. You can feed John a wordlist and a set of rules (John comes with default rule sets like "Single" and "Jumbo" rules) to produce variants of each word. For instance, if you suspect the password was based on a word or name, John can try that word with various mutations. John's incremental mode can generate brute-force guesses in order of likelihood (using Markov chains derived from training data), which tries more probable patterns first rather than pure alphabetical order. This can slightly improve efficiency for human-like passwords.
- **Platform and Extensibility:** John is cross-platform and open-source, with a large community. It can run on Windows, Linux, macOS, and even utilize specialized hardware (some formats support GPU, and even FPGA, although not for wallet.dat)
  github.com
  . You can use John in tandem with other tools; for example, using John to generate a candidate list which Hashcat then tests on GPU, or vice versa. John's `bitcoin2john.py` is a commonly used script (shipped with John) to prepare the wallet hash for cracking by John or Hashcat
  onlinehashcrack.com

  onlinehashcrack.com
  .

Application: In a wallet password recovery effort, John the Ripper can be used as a supplementary tool. You would first run `bitcoin2john.py` on the wallet.dat to get a hash. If you don't have a powerful GPU available, you could then run John on your CPU with a targeted wordlist or ruleset. For example, you might use John to exploit any personal knowledge of the password structure: set up a custom rule to prepend a particular year or append an exclamation mark to all words, etc., and let John churn through those. If John finds the password, it will output it; if not, its log (or pot file) will at least show you which guesses were tried. Given John's slower speed on this hash, it's often used to **focus on smart guesses** rather than broad brute force. In practice, one might generate a list of candidates using John's rules or incremental mode (for, say, all 8-character patterns John deems likely) and then feed that list to Hashcat for faster testing. Notably, the maintainers themselves advise focusing on probable guesses: *"Bitcoin wallets use a purposefully slow KDF... I recommend that you optimize which attacks you run and in which order – that's more important than speeds. Typically, when recovering lost passwords to Bitcoin wallets, you'd focus the attacks based on whatever information the owner can recall."*

openwall.com
. This guidance applies to using John: leverage any hints to craft specific rule-based attacks instead of naive brute force. In summary, John is a robust tool for generating and testing candidates with complex rules on CPU, and can complement GPU-based tools in a comprehensive recovery strategy.

PCFGCracker – Probabilistic Password Generator (AI-Inspired)

PCFGCracker is an open-source tool that uses a *probabilistic context-free grammar (PCFG)* approach to generate password guesses. This technique, developed in academia and released around 2019–2020, can be seen as an AI/ML-based strategy because it "learns" password patterns from training data. It analyzes a large list of known passwords to build a model of how people construct passwords (common substrings, numbers, symbols, patterns)

kitploit.com
. The result is a generator that can output password guesses in **descending order of probability** – effectively prioritizing the most likely passwords first.

- **Machine Learning of Password Patterns:** PCFGCracker uses machine learning to identify user password habits
  kitploit.com
  . For example, it might learn that a pattern like "word+123" or "Capital word + birthyear" is very common. It breaks passwords into components (letters, digits, symbols sequences) and learns the probabilities of these components and their order. This learned model (a "grammar" of passwords) can then generate new combinations according to those probabilities

kitploit.com
. Unlike a simple brute force, it won't go through every combination blindly; instead, it cleverly enumerates likely patterns (e.g., it might try "Bitcoin2010!" before a random string like "xkD#2pq9" because the former fits common patterns).

- **Targeted Training:** The tool comes with a default model trained on a million passwords (e.g., RockYou leak) but you can train it on custom lists
kitploit.com
. This means you can specialize it if you have insight into the user's context. For instance, if the wallet owner commonly used a particular style of password or if you have a small list of the owner's other passwords (from old accounts), you could train PCFGCracker on those to bias the guesses towards similar patterns. For a 2010 wallet, training on password leaks from the early 2010s might capture the flavor of passwords from that era.
- **Guess Generation and Usage:** PCFGCracker generates guesses in probability order and can output them to a file or pipe them to another program
kitploit.com
. It is slower at generation compared to brute-force (on the order of tens or hundreds of thousands of guesses per second, since it's doing a lot of computation to follow the grammar
kitploit.com
), but the idea is that *far fewer total guesses* may be needed to hit the correct password because it's aiming for the most likely ones first. This is a beneficial trade-off when dealing with a slow hash like a Bitcoin wallet where you cannot test billions of candidates easily. The tool itself is CPU-bound and doesn't use GPU (since generating guesses is not the bottleneck in cracking slow hashes – testing them is). Typically, you would generate a list of the top X guesses and then use Hashcat/John to test them.

Application: In a recovery scenario, PCFGCracker can be used to **generate a smart wordlist** to try on the wallet. For example, you train the PCFG model on a general dataset of known passwords (or any relevant passwords the user used, if available). Then instruct PCFGCracker to generate, say, the first few million likely passwords. These could include patterns like `[name]2010`, `[word]123`, `[word]!`, etc., ranked by likelihood. You would take this list and feed it into Hashcat as a dictionary, possibly combined with additional rules for slight tweaks. Because PCFG guesses are already "optimized" in order, you're testing the high-probability candidates first, which is exactly what you want given the slow attempts. This approach was shown to crack passwords with significantly fewer tries on average than brute force or naive dictionaries

kitploit.com
. In essence, PCFGCracker adds an AI-driven layer of strategy: it **reduces the search space** to plausible human-chosen passwords. For a wallet from 2010, this could be extremely valuable if the passphrase wasn't completely random. Many people in that era might have used a dictionary word or phrase with some numbers; PCFG will catch those patterns. After generating and testing the likely candidates, if the password still isn't found, you can expand the generation

or retrain with different data. PCFGCracker is open-source and was updated in the last few years, making it a state-of-the-art tool for intelligent password guessing
kitploit.com
.

PassGAN – Neural Network Password Guessing (AI)

PassGAN represents a newer class of password-cracking assistance tools that leverage deep learning (neural networks) to generate password guesses. Published around 2018, PassGAN uses a **Generative Adversarial Network (GAN)** to learn the distribution of real passwords and then generate new candidate passwords that resemble those a human might use. This is a truly AI-based approach: rather than relying on predefined patterns or rules, a neural network *learns* from data and can produce creative combinations that may not exist in known leaks but are statistically likely.

- **Deep Learning Approach:** In the PassGAN model, one neural network (the generator) tries to create plausible passwords while another (the discriminator) evaluates if they look like real passwords, and they train together. The result is a generator that can spit out millions of password candidates that statistically mimic real passwords people choose
  arxiv.org

  lamarr-institute.org
  . For example, if many people use song names or keyboard patterns, PassGAN may learn to generate those even if they weren't explicitly in the training list. This can sometimes crack passwords that rule-based methods miss, because humans can pick unpredictable patterns that nevertheless have some latent structure a GAN might latch onto.
- **Effectiveness on Common Passwords:** Studies have shown AI-generated guesses can be very effective. In one 2023 study, an AI password cracker (PassGAN) was able to crack about **51% of common passwords in under a minute**, and up to 65% within an hour (when testing against a large set of hashes)
  timesofindia.indiatimes.com
  . This doesn't directly translate to a single wallet's password (which is one hash, not millions), but it demonstrates that PassGAN generates very plausible guesses that overlap heavily with what real users choose. In practice, PassGAN could quickly produce the kind of password that a 2010 Bitcoin early adopter might have set, if it wasn't completely random.
- **Usage and Integration:** PassGAN's code is open-source (research prototypes are on GitHub) and typically runs in Python with TensorFlow/PyTorch. Using PassGAN involves either using a pretrained model (often trained on leaked password sets like RockYou) or training your own model on relevant data. Once the model is ready, you sample it to generate a list of password guesses (e.g. 1 million guesses). PassGAN by itself doesn't crack the hash; it only generates candidates. So, similar to PCFG, you would take its

output list and feed it into a cracking tool like Hashcat. The generation process can be time-consuming (training can take hours, but using a pretrained model to generate candidates is fairly fast, especially with a GPU). The key point is that PassGAN can output guesses that *aren't in any wordlist* but are likely to be used by people, giving you additional coverage beyond dictionaries.

Application: For recovering a lost wallet.dat password, PassGAN could be employed to **broaden the search with AI creativity**. Suppose you've tried common wordlists and rule-based tweaks without success. The next step could be to let a PassGAN model generate, say, 5 million candidate passwords that look like what a tech-savvy person in 2010 might use. These could include multi-word combinations or unusual character substitutions that simpler rules didn't produce. You'd then load these into Hashcat (possibly splitting into chunks due to the size) and run the GPU cracking on the wallet hash. Because the wallet's KDF is slow, you might prioritize the PassGAN list by probability (the model can often assign likelihood scores) and test the most likely few hundred thousand first. Even though PassGAN guesses are high-quality, it's still important to use them efficiently – perhaps in combination with other methods. One might intermix PassGAN guesses with PCFG guesses to cover both typical patterns and more exotic ones. The benefit of using an AI like PassGAN is to **capture human-like passwords that don't follow strict rules**. Since it's open-source, you can experiment with different training data (for example, training on passwords from 2009–2011 leaks to specialize it for that time frame). While PassGAN requires some machine learning know-how to use, it represents the cutting edge of password guessing approaches.

Source: PassGAN was introduced as a novel approach to replace human-crafted rules with machine learning

github.com
. It has been reported on in cybersecurity news for its impressive ability to crack large sets of passwords by *learning* their distribution
timesofindia.indiatimes.com
. This AI-driven method is a promising complement to traditional brute force in cases like a Bitcoin wallet recovery, where you're essentially guessing a single highly valuable password.

Combining Tools and Strategies for a 2010 Wallet

Recovering a wallet password from 2010 likely requires **combining these tools and techniques**. Each tool serves a different role: Hashcat (and John) are the engines that actually verify passwords against the wallet, while tools like BTCRecover, PCFGCracker, and PassGAN help generate smarter guesses or narrow down the search. Given the slow hashing of Bitcoin Core wallets

openwall.com
, a strategic approach is essential:

- **Leverage Known Clues First:** Use BTCRecover to exploit any remembered fragments or patterns. Its ability to handle partial passwords and common typos can quickly find a password if you were “close” but not exact. For example, if you recall the password had the word “Bitcoin” and a number, BTCRecover can try “Bitcoin” + wildcard combinations and catch things like “Bitcoin2010” or “bitcoin@2010” etc. with minimal effort.
- **Generate Intelligent Guesses with AI/ML:** While BTCRecover runs, you can prepare wordlists using PCFGCracker and/or PassGAN. These AI-driven lists will contain candidates ordered by likelihood. For instance, a PCFG list might prioritize simple patterns (“Satoshi123”) whereas PassGAN might throw in more random-looking yet plausible ones (“CorrectHorseBattery” style phrases). By merging these lists (removing duplicates) and sorting by probability or some heuristic, you get a master candidate list that is far more targeted than a brute-force list.
- **GPU-Crack with Hashcat:** Feed the refined candidate list into Hashcat with the wallet hash. With a GPU, Hashcat will churn through the list much faster than CPU tools. If the list is huge, consider using Hashcat’s rules and masks to expand around the highest-probability guesses instead of exhaustively testing low-probability ones. Also, Hashcat’s **–incremental mask** feature or custom masks can systematically brute-force within certain bounds (e.g. try all combinations of 8-character passwords made of [a-z][0-9] if you suspect the password was relatively short). Hashcat will also let you utilize multiple GPUs or even distribute the task across machines if available github.com
  .
- **Iterate and Optimize:** If the first round doesn’t succeed, analyze the results. Hashcat’s “potfile” will log any cracked passwords (for other tested hashes); even though you have one target, the potfile and BTCRecover’s output might show if any partial matches or patterns were hitting (some tools can report how many attempts were close in Hamming distance, etc.). You can refine your approach – maybe your guesses need to include a special character you didn’t consider, or maybe the password length range needs to increase. At this stage, you might deploy John the Ripper to try an **incremental search** with the parameters adjusted (e.g. include a larger symbol set or try a Markov attack that guesses more obscure combinations).
- **Use Distributed/Cloud Resources:** Because time is a factor, consider scaling out. As noted, Hashcat can work with distributed cracking; you can rent GPUs on the cloud (services like vast.ai as documented in BTCRecover’s guide) to run many instances in parallel
  btcrecover.readthedocs.io

  btcrecover.readthedocs.io
  . BTCRecover’s documentation shows an example of using 10 GPUs in cloud instances to speed up a Bitcoin Core wallet search by ~1000x compared to a single CPU
  btcrecover.readthedocs.io

  btcrecover.readthedocs.io
  . Just be cautious with security when handling wallet data – use BTCRecover’s **extract**

mode so that the actual keys aren't exposed, only the data needed for password checking
github.com
.

In conclusion, recovering a 2010-era Bitcoin Core wallet password is difficult but not impossible with these open-source tools. Prioritize **intelligent guessing** (using AI/ML methods and personal memory of the password) over blind brute force, because the wallet's encryption deliberately slows down each attempt

openwall.com
. A combination of **BTCRecover's guided search**, **Hashcat's GPU muscle**
openwall.com
, and **AI-generated candidate lists (PCFGCracker, PassGAN)** can dramatically improve your odds of success. Each tool brings something to the table: BTCRecover for user-friendly recovery and typos, Hashcat/John for raw cracking power and rule-based flexibility, and ML tools for creative guess generation. By applying them in concert, you cover all bases – from common passwords to the more complex or unconventional – in the most efficient way possible. Good luck with the recovery!

Sources:

1. BTCRecover documentation and README – features (open-source, multi-threaded, GPU support, typo handling)
   github.com

   github.com
   .
2. Hashcat official wiki/GitHub – description of capabilities (fastest GPU password cracker, multiple attack modes, distributed cracking)
   github.com

   hashcat.net
   .
3. OnlineHashCrack guide – Bitcoin wallet hash extraction and Hashcat mode 11300 example
   onlinehashcrack.com

   onlinehashcrack.com
   .
4. Openwall John-users mailing list – discussion of Bitcoin wallet cracking speeds (slow KDF, importance of focused attacks, John vs Hashcat GPU)
   openwall.com

openwall.com
.
5. KitPloit – PCFGCracker overview – machine learning of password grammar and generation of guesses in probability order
kitploit.com

kitploit.com
.
6. News article (Times of India, 2023) – effectiveness of AI password cracking (PassGAN able to crack large % of common passwords quickly)
timesofindia.indiatimes.com
.
7. Hashcat GitHub README – confirmation of open-source license (MIT) and multi-OS, hardware support
github.com
.
8. John the Ripper Jumbo GitHub – multi-platform support and hundreds of hash types (incl. Bitcoin)
github.com
.