Rainbow Tables

Goals:

Rainbow tables are an implementation middle road between ‘simple’ brute force attacks and dictionary attacks. A rainbow table is a series of precomputed chains leading from plaintext to the final reduced hashed, storing only the initial plaintext and the final reduced hash Espinosa, C. (2024). The table stores these as a compromise between storage space usage and computation intensity.

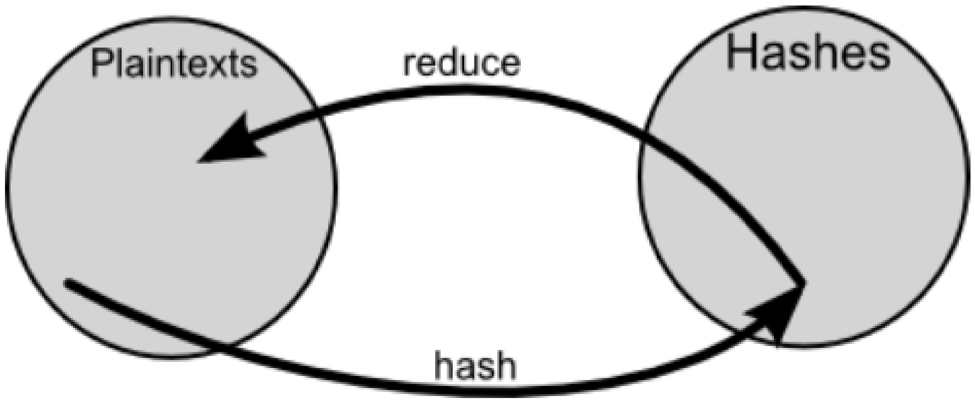
The key features of a rainbow table are, the table itself, the reduction function and the hashing function. The reduction function is a deterministic function that reduces the ciphertext in some way, this can be as complex as a mathematical formula or a simplistic method of selecting the first n numbers (aams-eam 2023). The hashing function is “a mathematical one-way function that takes an input and produces a fixed-size string.” (aams-eam 2023)

The way hashing works, An attacker cannot decrypt a ciphertext by going in reverse instead chooses to utilise chains that come into effect, using this it takes the ciphertext as input and reduces it comparing it to all the other stored reduced ciphertext. If any are equivalent it can then trace it from the beginning of the stored plaintext hashing and reducing until the hashed plaintext is equivalent to the original input and returning the plaintext associated.

Method:

The method of implementation chosen is a class based structure, this allows for two key features, polymorphism of the reduction function and the implementation of multiple rainbow tables with greater ease and simplicity.

The initial step is to create the rainbow table and fill it using an array of starting values, this could be either common passwords or a series of integers (i.e. 0,1,2…), etc. Taking these inputs each is then hashed and reduced repeatedly until the declared chain length is reached.



(Kuliukas, k)

Once the chain length is reached, it stores the original plaintext and the last reduced hashed password. The intermediate steps are not stored, as the intention is to reduce storage requirements, by not storing each step we reduce the amount of data required to be stored to only the necessary data to recreate the traversal of the chain.

This concludes the construction of the table, now it becomes the issue of traversing and searching the table. The first step in traversing is reducing the ciphertext into the new plaintext, then the new plaintext is compared to the reduced hashes (keys) within the table. If there is not a match the new plaintext is rehashed and reduced, and recompared.

If there is a match, we take the plaintext from the chain and begin the cycle of hashing then comparing the hash to the ciphertext, If it doesn't match the plaintext is updated to the reduced hash then is rehashed and recompared. If it does match, the previous plaintext is returned.

The reasoning behind a class structure is it allows for multi tables where each table's domain is reduced by a different reduction function. This is to allow for simplified integration of different reduction functions via polymorphism. Each chain can be implemented using its own reduction function resulting in fewer collisions Joaquin Brandan (Mar 23, 2018), (when two hashes reduce to the same reduction).

Conclusions:

The prime benefits of a rainbow table is the speed in comparison to 'simple' brute force attacks, and its strength against weaker cryptographic hashing algorithms. The key weaknesses of rainbow tables are the large storage requirements, salted passwords, and collisions.

Rainbow tables are significantly faster than brute force attacks as they don't cycle through every combination, but instead use pre-computed chains that provide faster lookups by traversing the chain . In small domains or where collisions are common such as in weaker cryptographic hashing algorithms this allows rainbow tables to quickly recompute the original plaintext using the chains (aams-eam 2023).

Rainbow tables walk a middle ground between brute force and dict attacks, where it has reduced computation as it's already been precomputed but also still maintains a large amount of storage as it stores the hash chains. This forces it to use a significant amount of storage space but less than conventional dictionary attacks. (Espinosa, C)

One of the main cryptographic weaknesses to rainbow tables is salt, as it becomes more common to use salts in passwords it becomes increasingly difficult to utilise rainbow tables. (Mirza , S) Salting is a robust method of preventing rainbow table attacks and collisions that cause computational inefficiencies as it becomes efficient to compute all variations.

Collisions are a fundamental consequence of rainbow tables, hashing algorithms aim to have as few collisions as possible, but when a rainbow table reduces it inevitably reduces a hash down to one already existing within the domain. This is a collision, collisions lead to a waste in resources as you end up computing the same chain. (Sheasby, R)

There is a tradeoff between chain length and chain amount, if there's an increased number of shorter chains the lookup will be faster but be more space intensive and vice versa with longer chains requiring less storage but more computation at run time. (aams-eam, 2023)

Key Insights:

The fundamental design of rainbow tables allows attackers a faster less memory intensive way to crack passwords, this is achieved by precomputing final reduction into the hashes into chains associated with the original text. When we know a password is random and unsalted this is where rainbow tables become useful as it allows for both more optimised storage and reduction of run time computation. There are some strong issues and hindrances when implementing a rainbow table, these include salting, and collisions.

Reference list

Espinosa, C. (2024). *Demystifying Rainbow Tables in Cryptography - Blue Goat Cyber*. [online] Blue Goat Cyber. Available at: https://bluegoatcyber.com/blog/demystifying-rainbow-tables-in-cryptography/ [Accessed 10 Oct. 2025].

Mirza , S. (2022). *All You Wanted To Know About Rainbow Table Attacks | Cyphere*. [online] cyphere. Available at: https://thecyphere.com/blog/rainbow-table/. Accessed 12 Oct. 2025].

Sheasby, R. (2021). *Rainbow Tables (probably) aren't what you think — Part 2: Probability, Efficiency, and Chain…*. [online] Medium. Available at: https://rsheasby.medium.com/rainbow-tables-probably-arent-what-you-think-part-2-probability-efficiency-and-chain-9e2fb5e8cdc9.

Joaquin Brandan (Mar 23, 2018). *Information Security Stack Exchange*. [online] Information Security Stack Exchange. Available at: https://security.stackexchange.com/questions/181864/clarification-on-how-rainbow-tables-use-multiple-reduction-functions-to-avoid-co [Accessed 19 Oct. 2025].

Links used during development: *No code was directly sourced instead these were used as an insight into how to structure and develop Rainbow tables*

Luc Gommans (2017). *A toy rainbow table script*. [online] Gist. Available at: https://gist.github.com/lgommans/83cbb74a077742be3b31d33658f65adb [Accessed 16 Oct. 2025].

otus (2017). *Cryptography Stack Exchange*. [online] Cryptography Stack Exchange. Available at:

https://crypto.stackexchange.com/questions/47731/multiple-rainbow-tables-and-their-success-probability [Accessed 12 Oct. 2025].

Kuliukas , K. (n.d.). *How Rainbow Tables work*. [online] kestas.kuliukas.com. Available at: https://kestas.kuliukas.com/RainbowTables/. [Accessed 12 Oct. 2025].

aams-eam (2023). *The Art of Password Cracking: Rainbow Tables*. [online] Pages.dev. Available at:
https://aams-eam.pages.dev/posts/the-art-of-password-cracking_rainbow-tables/
[Accessed 13 Oct. 2025].