CT255
Introduction to Cybersecurity

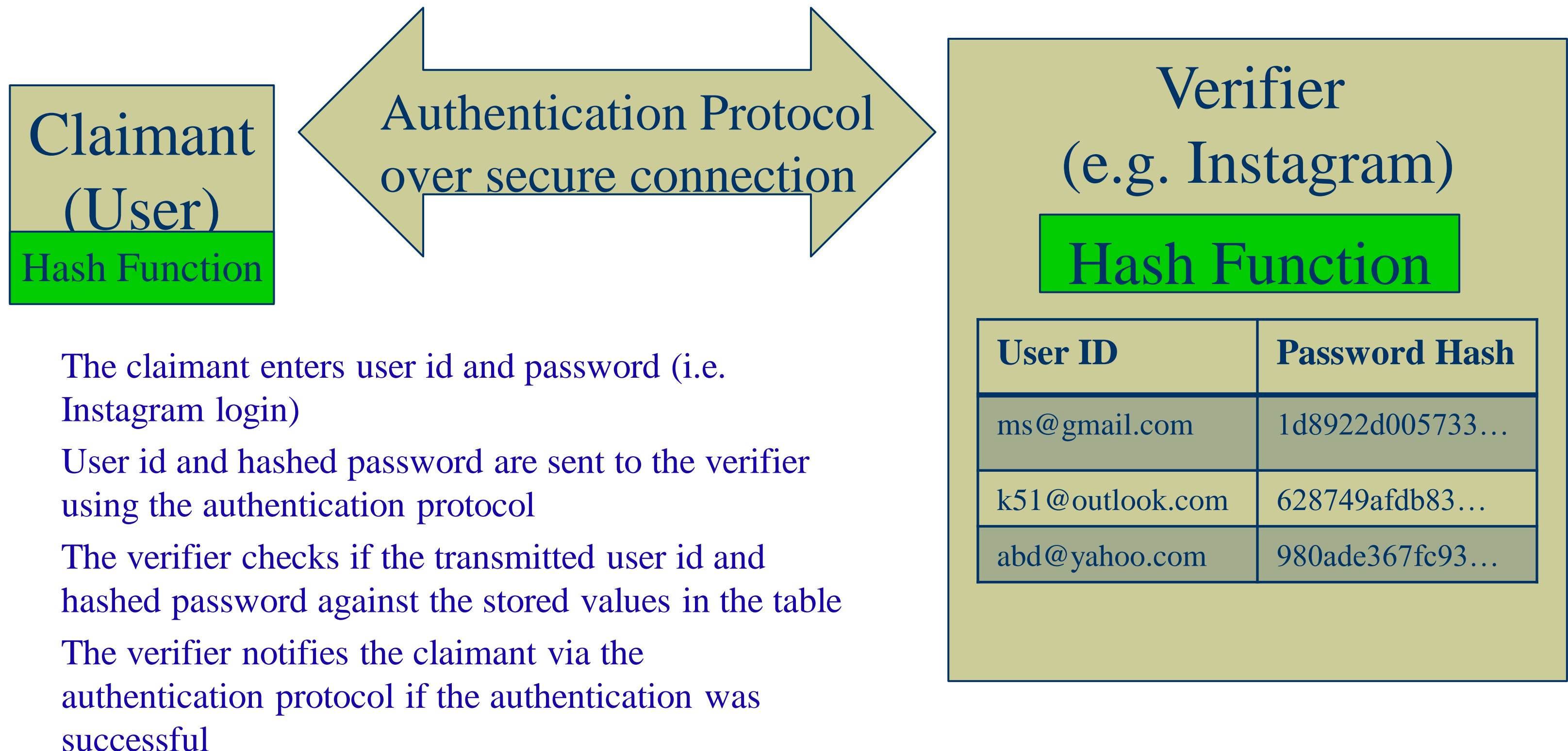
Lecture 5
„Crack me if you can”

Dr. Michael Schukat, 2019-2022

Lecture Overview

- Practical approaches to recover hashed passwords, i.e. hash cracking, and their characteristics / limitations
  - Dictionary attacks
  - Look-up table-based attacks
  - Hash chains
  - Rainbow tables
- Defense mechanisms against hash cracking

Recap: Putting it all together – Version 2



User ID	Password Hash
ms@gmail.com	1d8922d005733…
k51@outlook.com	628749afdb83…
abd@yahoo.com	980ade367fc93…

1. The claimant enters user id and password (i.e. Instagram login)
2. User id and hashed password are sent to the verifier using the authentication protocol
3. The verifier checks if the transmitted user id and hashed password against the stored values in the table
4. The verifier notifies the claimant via the authentication protocol if the authentication was successful

Hash Cracking

- Reverse-Engineer passwords?
  - One-way function, ergo not possible
  - But hash functions are public

User ID	Password Hash
ms@gmail.com	1d8922d005733…
k51@outlook.com	628749afdb83…
abd@yahoo.com	980ade367fc93…

DEFUSE: A Online Text & File Checksum Calculator

- https://defuse.ca/checksums.htm

Online Text & File Checksum Calculator

This page lets you hash ASCII text or a file with many different hash algorithms. Checksums are commonly used to verify the integrity of data. The most common use is to verify that a file has been downloaded without error. The data you enter here is 100% private, neither the data nor hash values are ever recorded.

Enter some ASCII or UNICODE text...

Remove line endings

Calculate checksums..

File (5MB MAX)

Choose File No file chosen Calculate checksums...

Supported Hash Algorithms

md5 LM NTLM sha1 sha256 sha384 sha512 md5(md5()) MySQL4.1+ ripemd160 whirlpool adler32 crc32 crc32b fnv1a32 fnv1a64 fnv132 fnv164 gost gost-crypto haval128,3 haval128,4 haval128,5 haval160,3 haval160,4 haval160,5 haval192,3 haval192,4 haval192,5 haval224,3 haval224,4 haval224,5 haval256,3 haval256,4 haval256,5 joaat md2 md4 ripemd128 ripemd256 ripemd320 sha224 snefru snefru256 tiger128,3 tiger128,4 tiger160,3 tiger160,4 tiger192,3 tiger192,4

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Dictionary-Based Brute-Force Search

- Dictionary search can be used to systematically identify a match for a given hash value
  - The underlying hash function must be known
- Dictionaries are based on large word, phrase or password collections
- ☺ :
  - Straight forward process
- ☹ :
  - Significant computational effort to find match
  - No guaranteed result

Ollscoil na Gaillimhe
University of Galway

Example

- Assume a hash code and the underlying hash function H are known
- The dictionary for H contains 10^{10} entries (i.e., password candidates)
- A single computer can compute 10^5 hash values per second
- It takes 10^5 seconds (~29 hours) to search the entire dictionary for a match
- Need for performance improvements!

Ollscoil na Gaillimhe
University of Galway

CrackStation's Password Cracking Dictionary

- https://crackstation.net/crackstation-wordlist-password-cracking-dictionary.htm

CrackStation's Password Cracking Dictionary

I am releasing CrackStation's main password cracking dictionary (1,493,677,782 words, 15GB) for download.

What's in the list?

The list contains every wordlist, dictionary, and password database leak that I could find on the internet (and I spent a LOT of time looking). It also contains every word in the Wikipedia databases (pages-articles, retrieved 2010, all languages) as well as lots of books from Project Gutenberg. It also includes the passwords from some low-profile database breaches that were being sold in the underground years ago.

The format of the list is a standard text file sorted in non-case-sensitive alphabetical order. Lines are separated with a newline "\n" character.

You can test the list without downloading it by giving SHA256 hashes to the free hash cracker. Here's a tool for computing hashes easily. Here are the results of cracking LinkedIn's and eHarmony's password hash leaks with the list.

The list is responsible for cracking about 30% of all hashes given to CrackStation's free hash cracker, but that figure should be taken with a grain of salt because some people try hashes of really weak passwords just to test the service, and others try to crack their hashes with other online hash crackers before finding CrackStation. Using the list, we were able to crack 49.98% of one customer's set of 373,000 human password hashes to motivate their move to a better salting scheme.

Download

Note: To download the torrents, you will need a torrent client like Transmission (for Linux and Mac), or uTorrent for Windows.

Torrent (Fast)

GZIP-compressed (level 9). 4.2 GiB compressed. 15 GiB uncompressed.

HTTP Mirror (Slow)

Checksums (crackstation.txt.gz)

```
MD5:    4748a72706ff934a17662446862ca4f8
SHA1:   efa3f5ecbfba03df523418a70871ec59757b6d3f
SHA256: a6dc17d27d0a34f57c989741acdd485b8aee45a6e9796daf8c9435370dc61612
```

Lookup Table-Based Attacks

- For a given hash function and dictionary
  - Calculate hash value for all dictionary entries
  - Add both values to a table (i.e. one line per entry)
  - Sort table (e.g. in ascending order of hash values)
    - Also called **lookup table**
- Example table (assuming 44-bit hash values):

Hash value	Password
0x00000000354	gangster
0x00000001003	Bluemoon
…	…

Lookup Table-Based Attacks

- The matching password for a given hash value can be recovered by systematically searching for it in the dictionary
- ☺ :
  - Such a table can be generated offline
  - The search process itself is fast (~$\log_2$(# of entries)) with **binary search**
    - A table containing $1.8\text{x}10^{19}$ entry would require just 64 guesses to find (or not) the correct password for a given hash value
- ☹ :
  - Huge table, with no guaranteed result
  - Different table required for every hash function

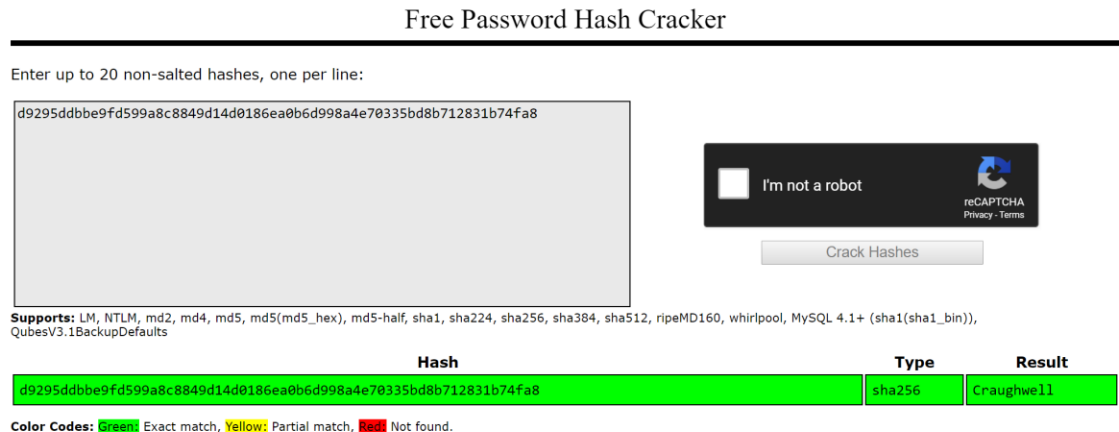
Lookup Table-Based Attacks: Example

- Assume a hash function that generates 16 byte (128 bit) hash values, e.g. MD5
- We calculate a lookup table for all possible 6 character long passwords composed of 64 possible characters A-Z, a-z, 0-9, "." and "/"
- A table would consist of 64^6 (= 68,719,476,736) entries, with every entry consisting of a 6 byte password and a 16 bytes hash
- **Total size of table ~ 1.4 Terabyte**

Ollscoil na Gaillimhe
University of Galway

Crackstation's free Password Hash Cracker

- https://crackstation.net/



In-Class Activity: Password recovery

- 5 minutes only, work alone or in a group
- What to do:
  - Pick a password and calculate its MD5 or SHA1 hash using https://defuse.ca/checksums.htm
  - Copy and paste the hash value into https://crackstation.net/ to see if it is can be recovered
  - Repeat the above and keep a list of all passwords
    - that **can** be cracked
    - that **cannot** be cracked

Ollscoil na Gaillimhe
University of Galway

Rainbow Tables

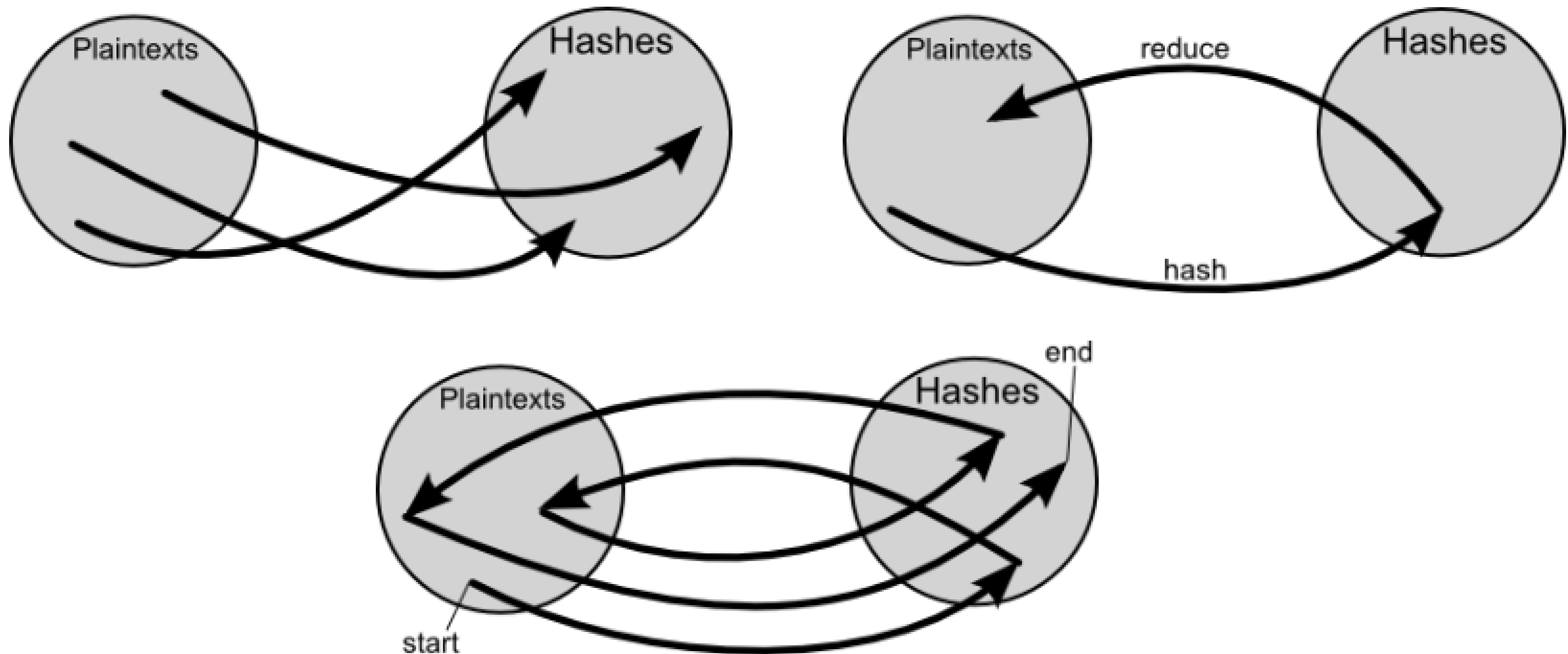
- Uses less computer processing time and more storage than a brute-force attack
- Uses more processing time and less storage than a simple lookup table
- Practical example of a **space–time trade-off**

Pre-Computed Hash Chains

- Calculate long chains of hash values (using a hash function "→" and a reduction function "→", e.g.
  aaaaaa→173bdfede2ee3ab3 → jdjkuo →9fdde3a0027fbb36 →… → k3rtol
  - In the example we only consider passwords that are 6 characters long
  - Each chain starts with a random password and has a fixed length, e.g. 10,000 segments
  - Here "→" converts the 64 bit hash value into an arbitrary 6 byte long string again, i.e. its not an inverted hash function!
- We only store the first and the last value (starting point and end point), i.e. "aaaaaa" and "k3rtol"

Ollscoil na Gaillimhe
University of Galway

Hash Functions, Reduction Functions and Chains



Ollscoil na Gaillimhe
University of Galway

Pseudo-Code to create a single Chain

- This example creates a chain with the start value "abcdefg" and a length of 10,000 elements
- Note that the last value of this chain is a hash value (i.e. ciphertext)

```
String plaintext, first, ciphertext;

plaintext = first = "abcdefg";

for ( int i=0; i<10000; i++ ) {
  ciphertext = hash_it (plaintext);
  plaintext = reduce_it (ciphertext);
}

System.out.printf ("%s:%s\n", first, ciphertext);
```


Ollscoil na Gaillimhe
University of Galway

Chain Lookup

Assume we have a table with just 2 chains (with start and end values), i.e.

aaaaaa→173bdfede2ee3ab3 →… → 8995tg →9fdde3a0027fbb36 →… → k3rtol

hfk39f→856385934954950 →… → delphi →759858fde66e8aa8 →… → prp56e

… and a hash value "759858fde66e8aa8" we'd like to crack

We apply consecutively "→" and "→", until we

- hit a known end value (k3rtol or prp56e in the example), or
- have repeated the operation x times (with x being the length of the chain)

If we hit the known end value "prp56e" we repeat the transformation starting with its start value "hfk39f", until we hit "759858fde66e8aa8" again

The password "delphi " that led to this hash is the solution

Ollscoil na Gaillimhe
University of Galway

Chain Lookup Pseudocode

1. Input: Hash value H
2. Reduce H into another plaintext P
3. Look for the plaintext P in the list of final plaintexts (i.e. end values), if it is there, break out of the loop and go to step 6.
4. If it isn't there, calculate the hash H of the plaintext P
5. Goto 2., unless you've done the maximum amount of iterations
6. If P matches one of the final plaintexts, you've got a matching chain; in this case walk through the chain again starting with the corresponding start value, until you find the text that translates into H

Chain Collisions

Consider the following scenario:

aaaaaa→ … → 173bdfede2ee3ab3 → delphi → 759858fde66e8aa8 →… → prp56e
hfk39f→ … → 856385934954950 → delphi → 759858fde66e8aa8 →… → prp56e

2 chains merge, because

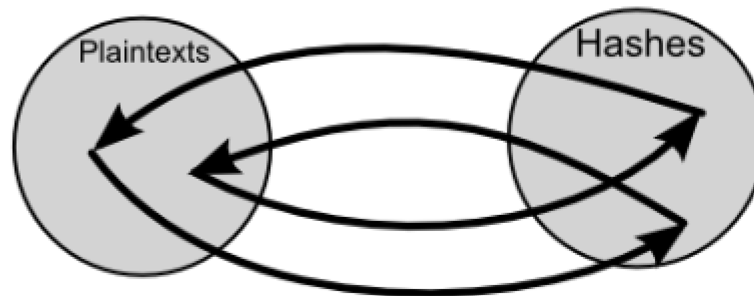
- the reduction function translates two different hashes into the same password, or
- the hash function translates two different passwords into the same hash (which should not happen)

Because of collisions there is no guarantee that your chains will ever cover all possible passwords

Ollscoil na Gaillimhe
University of Galway

Chain Loops

- Here you find repetitions of hashes in a single chain



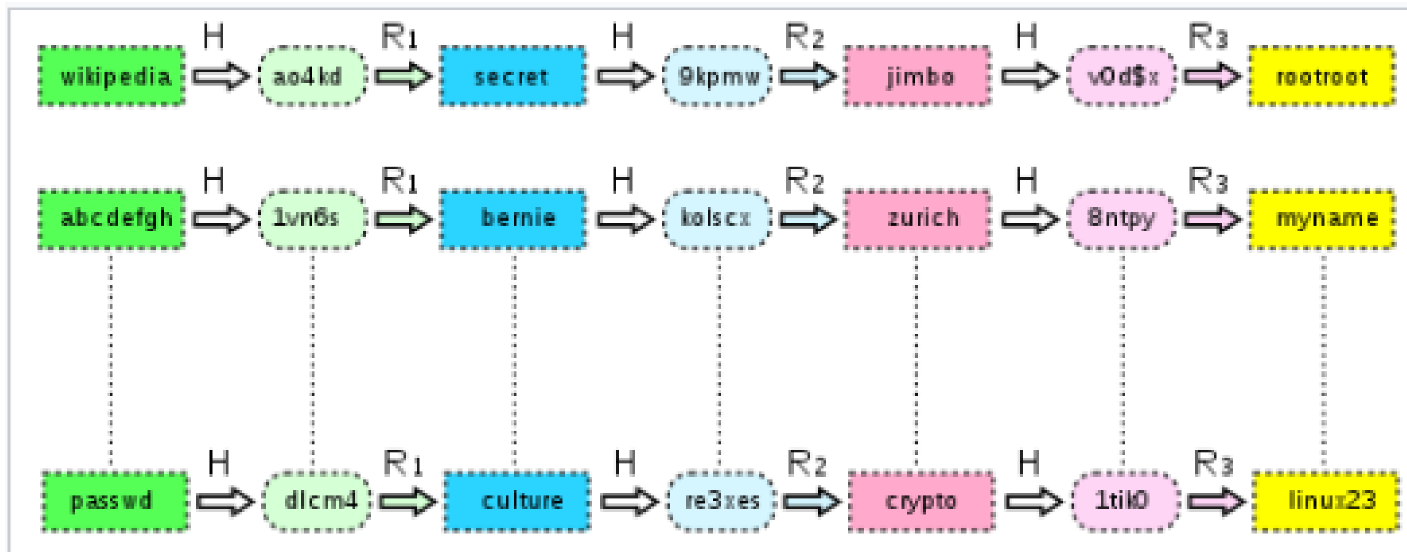
Ollscoil na Gaillimhe
University of Galway

Rainbow Tables

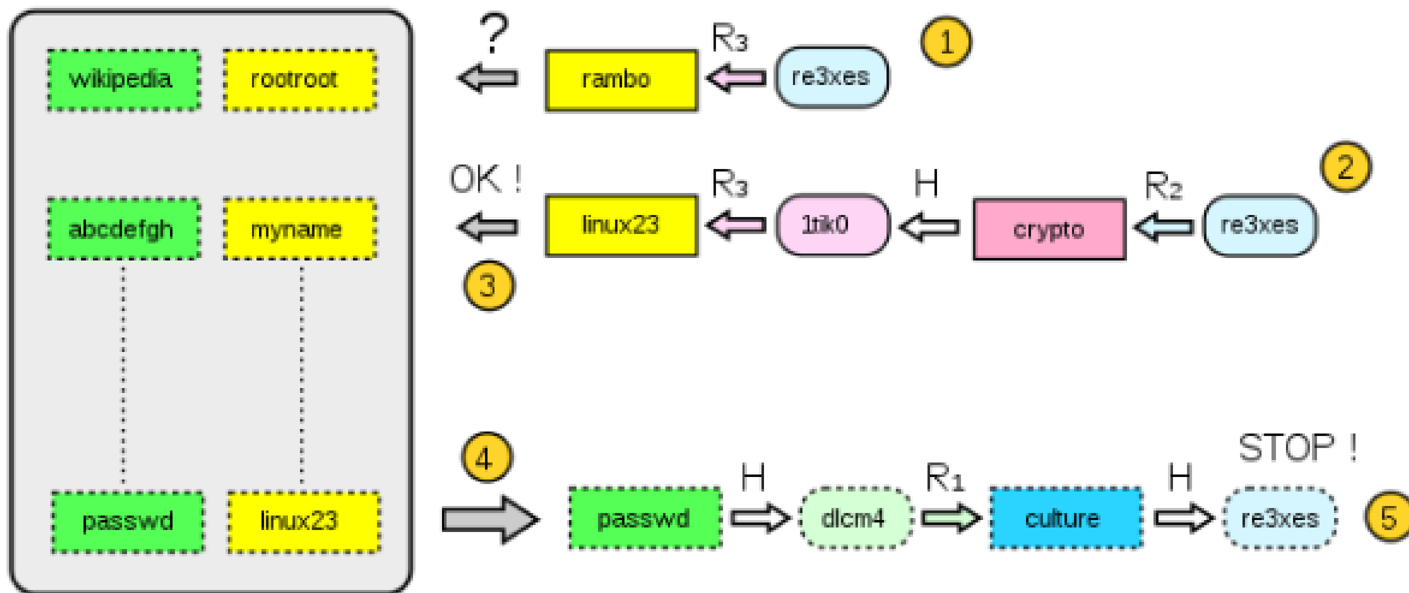
- Rainbow tables effectively solve the problem of collisions with ordinary hash chains by replacing the single reduction function R with a sequence of related reduction functions R_1 through R_k (one reduction function per column)
- In this way, for two chains to collide and merge they must hit the same value on the same iteration, which is rather unlikely

Example Rainbow Table (Source: Wikipedia)

- Assume a Rainbow table consisting of n chains with 3 reduction functions R_1, R_2 and R_3 per chain and a hash function H
- Each rainbow table entry stores the leftmost and the rightmost password candidate as calculated by the chain (e.g., wikipedia and rootroot)



Example Rainbow Table Processing (Source: Wikipedia)



Ollscoil na Gaillimhe
University of Galway

Example Rainbow Table Processing (Source: Wikipedia)

- The hash we want to crack is re3xes
- Step 1: Calculate R3(re3xes) and determine if the result matches any of the chain end values;
  - if there is a match, then you've identified the hash chain; go through the hash chain from the beginning to find the matching password (as seen before) and exit;
  - else goto Step 2

Example Rainbow Table Processing (Source: Wikipedia)

- Step 2: Calculate R3(H(R2(re3xes))) and determine if the result matches any of the chain end values;
  - if there is a match, then you've identified the hash chain; go through the hash chain from the beginning to find the matching password (as seen before) and exit;
  - else goto Step 3

Ollscoil na Gaillimhe
University of Galway

Example Rainbow Table Processing (Source: Wikipedia)

- Step 3: Calculate R3(H(R2(H(R1(re3xes))))) and determine if the result matches any of the chain end values;
  - if there is a match, then you've identified the hash chain; go through the hash chain from the beginning to find the matching password (as seen before) and exit;
  - else exit with no match found

Ollscoil na Gaillimhe
University of Galway

Perfect and non-perfect Rainbow Tables

- In a **perfect rainbow table** passwords do not appear in more than one chain
- **Non-perfect rainbow tables** do not have this requirement
  - They are easier to compute, but less memory-efficient because of repeating passwords (note that these usually do not cause collisions because of the reduction functions R_K)
- The longer the password, the more complex a rainbow table becomes

Ollscoil na Gaillimhe
University of Galway

Defense against Rainbow Tables

- Idea:
  - Increase the (required) length of a password
  - By doing so there are many more potential passwords to be considered by a rainbow table …
  - … up to a point where such tables are simply no more economical to generate
  - Increasing the password length can be either done by the password owner (i.e. the claimant), or algorithmically

Ollscoil na Gaillimhe
University of Galway

Defense against Rainbow Tables

The following approaches make rainbow tables less economical any

1. Long passwords (enforce users to use long passwords)
2. Passwords salts (algorithmic approach)
   - A unique and random, but known string is appended to each password before its hash is calculated:

User ID	Salt	Password Hash	Password (not part of table)
ms@gmail.com	12367	1d8922d005733…	12367KenSentme!
k51@outlook.com	56f87	628749afdb83…	56f87Fluffybear
abd@yahoo.com	465d0	980ade367fc93…	46d05Limerick

   - A (potentially short) user password is thereby extended

Defense against Rainbow Tables

3. Pepper (algorithmic approach)
   - Like salt, but a unique secret string is added to all passwords before they are hashed
4. Multiple iterations (algorithmic approach)
   - A password p is hashed multiple (> 1000) times before stored in the database, e.g. H(H(H(H(..(H(p))..))))
5. Combination methods (algorithmic approach), e.g.,
   - NewHash(password) = H(H(password) || salt)
   - NewHash(password) = H_1(H_2(password) || salt), with hash functions H_1 and H_2

Ollscoil na Gaillimhe
University of Galway