CMSC 426
Principles of Computer Security

Lecture 15

Password Authentication and Cracking

Last Class We Covered

- “Big” ethics questions and ideas

- Case studies
  - Let’s Encrypt
  - Marcus Hutchins (WannaCry)
  - Hacking back
  - Responsible disclosure
    - Gray hat hacking
  - Apple encryption

Any Questions from Last Time?

Authentication and Hardening

- In the next unit, we will be covering:

- Authentication
  - How do users authenticate themselves to systems?
  - How do attackers take advantage of these authentication methods?

- Hardening
  - How do we configure systems so that they are secure against attackers?

Components of Authentication

- Identification
  - Provide a claimed identity to the system
  - *e.g.*, username, SSN, UMBC ID

- Verification
  - Establish validity of the provided identity
  - *e.g.*, password, PIN, swipe card

Means of Verifying Identity

- Something a user knows
  - Password, PIN, security questions
- Something a user possesses
  - Electronic keycard, smart card
- Something a user is or does
  - Biometrics such as fingerprints, facial recognition

Multifactor Authentication

- Using more than one category of authentication in order to verify a user’s identity

- For example, when you log into your online bank account from a new computer
  - Bank sends a one-time PIN number to your phone
  - Have to know the password <u>and</u> possess the phone to authenticate

Authentication Using Passwords

Why Do We Still Use Passwords?

- They’re kind of terrible?

- Very prone to user error
  - Use of weak passwords
  - Reuse of passwords across multiple accounts
  - Forgetting to change default credentials

- But no one can seem to replace them with anything better yet
  - Still the most widespread verification method

Password Managers

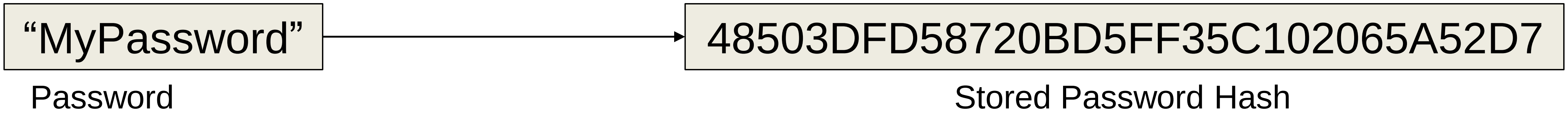
- All of the security of long passwords (with none of the inconvenience)
  - No password reuse across multiple sites
  - Resistant to keyloggers

- Single point of failure
  - Online storage: susceptible to hacking
  - Local storage: susceptible to malware and user stupidity

Password Hashing

- Plaintext passwords should never be stored on disk

- When a user makes an account on a system
  - Their password should be hashed
    - Using a cryptographically secure hashing algorithm
  - The resulting hash digest should be stored on disk



Password Hashing Usage

- When a user attempts to log in, the password they enter is hashed and compared to the one on disk

Stored Password Hash

48503DFD58720BD5FF35C102065A52D7

✓

“MyPassword”

Password Attempt

48503DFD58720BD5FF35C102065A52D7

Password Attempt Hash

Common Password Authentication Features

- Requiring a user to wait between authentication attempts
- Locking a user out if they fail to authenticate multiple times
- These features prevent cybercriminals from simply brute-forcing login attempts on a target system

Distributed Online Password Guessing

- Computers that allow users to log in over a network (such as SSH and RDP) are constantly being scanned by automated password guessers

- Check for computers with default / weak credentials

- Handy for malicious purposes that aren't targeted
  - Botnets, cryptocurrency mining
  - Example: https://www.fireeye.com/blog/threat-research/2015/02/anatomy_of_a_brutef.html

Offline Password Cracking

- If hackers can gain access to the password hashes on a system, they can perform <u>offline</u> password cracking

- No longer limited by restrictions on target computer, such as limited number of guesses or wait time

Offline Password Cracking Methods

Brute-Force Attack

- Generate the hash of every possible password and check for matches

- Pros
  - Thorough
  - Can crack short passwords easily

- Cons
  - Exponentially more time consuming as passwords get longer

Dictionary Attacks

- Most users don’t use random strings as passwords
  - Passwords tend to contain real words and predictable patterns

- Create a list of potential passwords, hash all of them, and store password-hash pairs in a dictionary

- How do you create your wordlist?
  - Let’s go on a quick tangent…

Tangent: RockYou

- A company that developed widgets for MySpace
- Suffered a data breach in 2009 due to an unpatched, ten-year-old SQL vulnerability

- Passwords were stored in plaintext!
    - Over 32 million accounts affected
    - Over 14 million unique passwords

- Now commonly used as a password cracking wordlist

Tangent: Common RockYou Passwords

Rank	Count	Password
1	290,792	123456
2	79,076	12345
3	76,789	123456789
4	59,462	password
5	49,952	iloveyou
6	33,291	princess
7	21,725	1234567
8	20,901	rockyou
9	20,553	12345678
10	16,648	abc123

Rank	Count	Password
11	16,227	nicole
12	15,308	daniel
13	15,163	babygirl
14	14,726	monkey
15	14,331	lovely
16	14,103	jessica
17	13,984	654321
18	13,981	michael
19	13,488	ashley
20	13,456	qwerty

Dictionary Attack Viability

- For N passwords:
  - Generating the dictionary is O(N) time
  - The dictionary requires O(N) storage space
  - Looking up a password in the dictionary is O(1)

- Once generated, can be reused across multiple attacks!

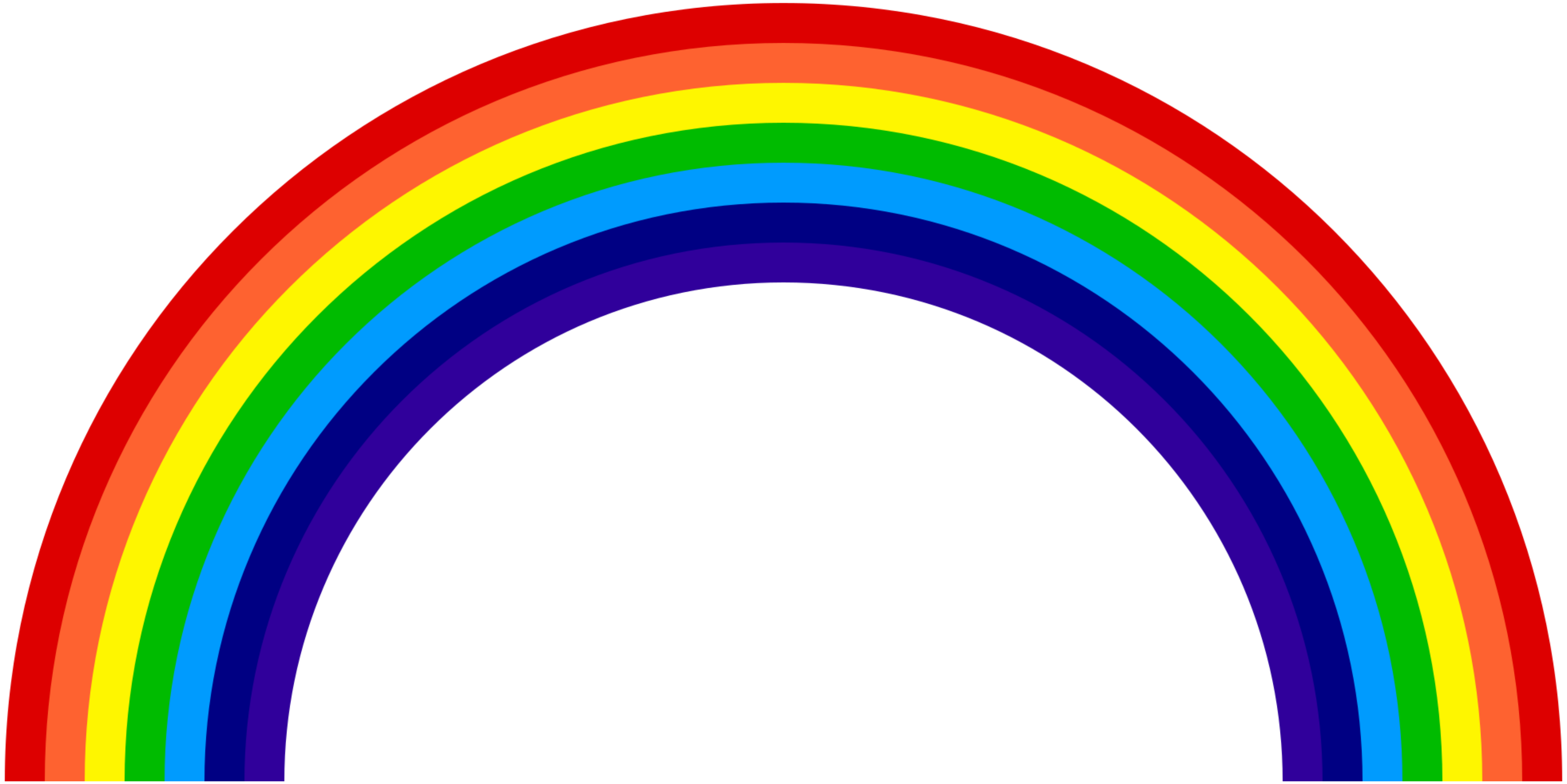
- Not as thorough as brute-forcing - if a target hash isn’t in your dictionary, you’re out of luck

Dictionary Attacks: Different Definitions

- “Traditional” dictionary attacks
  - Attempting “dictionary” words when brute-forcing a password

- “Pre-computed” dictionary attacks
  - Pre-compute possible passwords and their hash result
  - Create a dictionary data structure (key:value)
    - Where hashes are keys, and the password is the result
  - In this class, this is what we normally mean when we say “dictionary”

~*~ Rainbow Tables ~*~



Rainbow Tables

- Similar to a dictionary attack
  - Table of pre-computed passwords and corresponding hashes
- Time-memory tradeoff
  - Takes up less space on disk than a dictionary attack
  - Takes more time to perform lookups
- Generate chains of passwords and hashes
  - Only need to store the beginning and end of each chain in the rainbow table

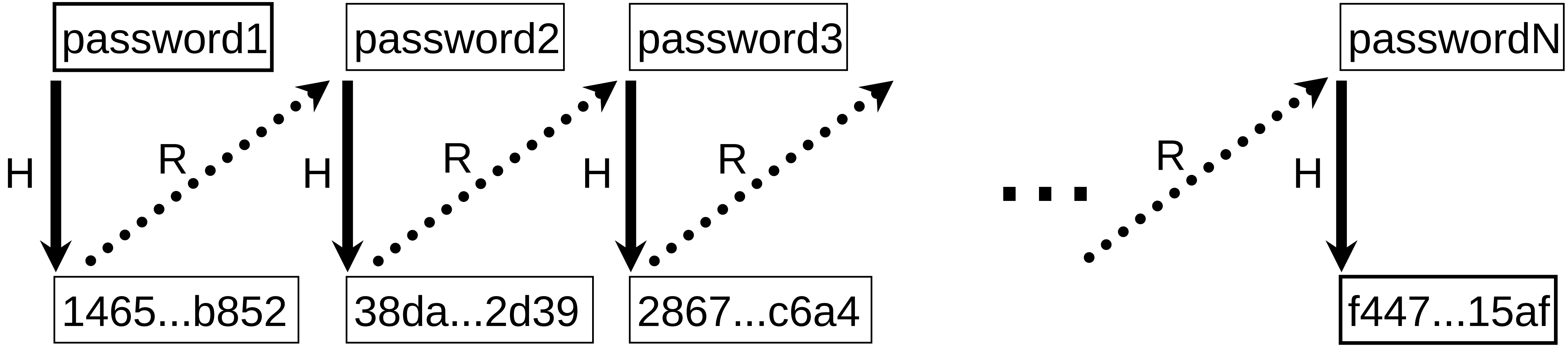
Rainbow Tables

- To crack a password hash, generate its chain and check if hashes are in the rainbow table
  - If so, generate the chain from the beginning password in the chain
  - Plaintext password will be located just before the target hash in the chain

- Two different implementations
  - End the chain when it reaches a certain length
  - End the chain when a password hash meets a certain condition

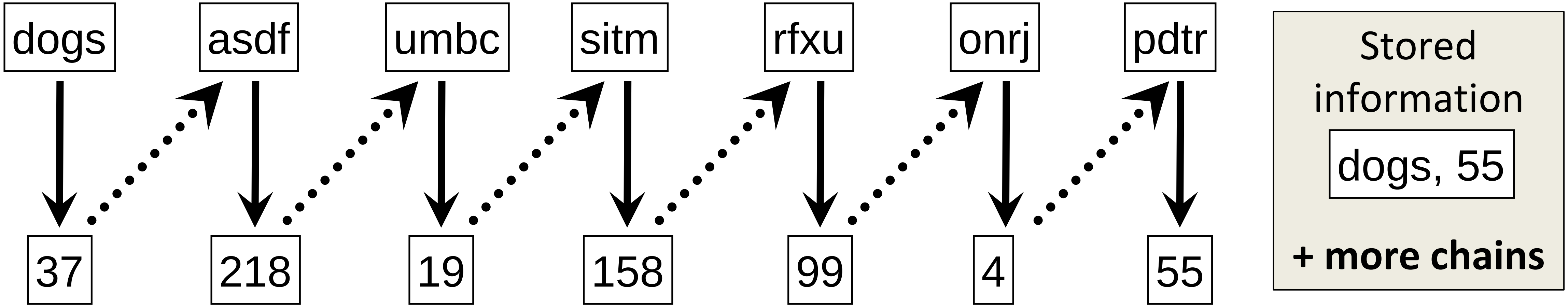
Rainbow Tables

- To generate a chain, need a reduce function
  - One-to-one mapping from a hash to a different password in the list



Rainbow Tables Example: Generation

- Choose a starting password
- Hash and reduce, over and over (and over and over)
- Only store the starting password, and the final hash result
- Repeat previous steps to make many more chains



dogs asdf umbc sitm rfxu onrj pdtr

37 218 19 158 99 4 55

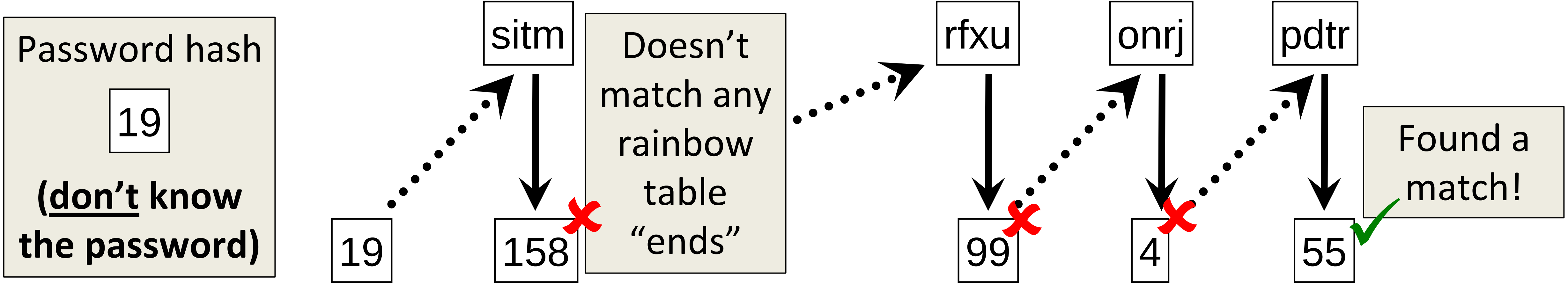
Stored information

dogs, 55

+ more chains

Rainbow Tables Example: Finding

- When attempting to find a user's password in the rainbow table
  - Start running it through the same hash and reduce function chain
  - At each stage, check to see if the hash matches any chain ends
    - If it does, move onto "cracking" the password
  - Search has failed if the hash/reduce steps on the password hash have reached the length of the chain, but no matches were found



Password hash

19

(<u>don't</u> know the password)

19

sitm

158

Doesn't match any rainbow table "ends"

rfxu

99

onrj

4

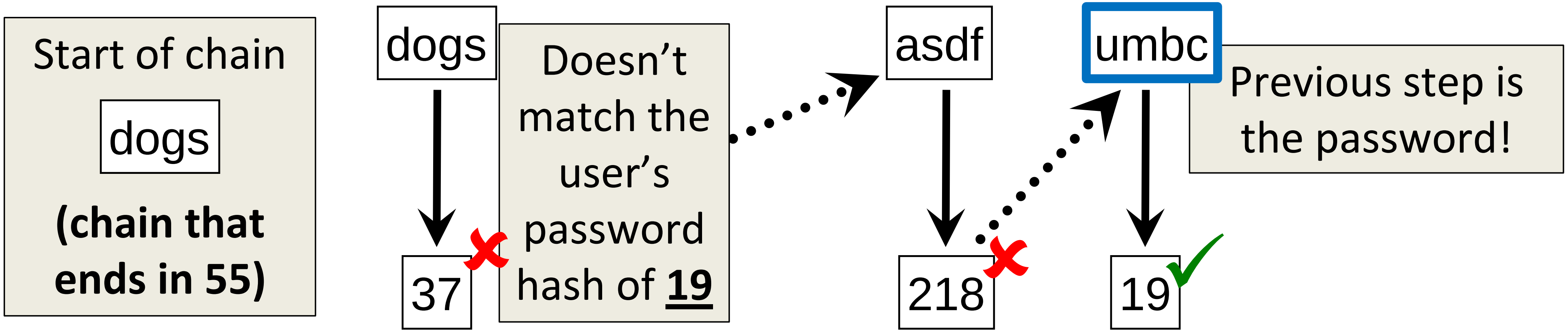
pdtr

55

Found a match!

Rainbow Tables Example: Cracking

- Once a match has been found, we can crack the password
  - Start with the table’s known beginning
  - Perform the hash and reduce chain again
    - Check to see if the hash at each step matches the hash we already know corresponds to the user’s password
    - If it does, the previous step is the password that created it



Start of chain

dogs

(chain that ends in 55)

dogs → 37 ✗

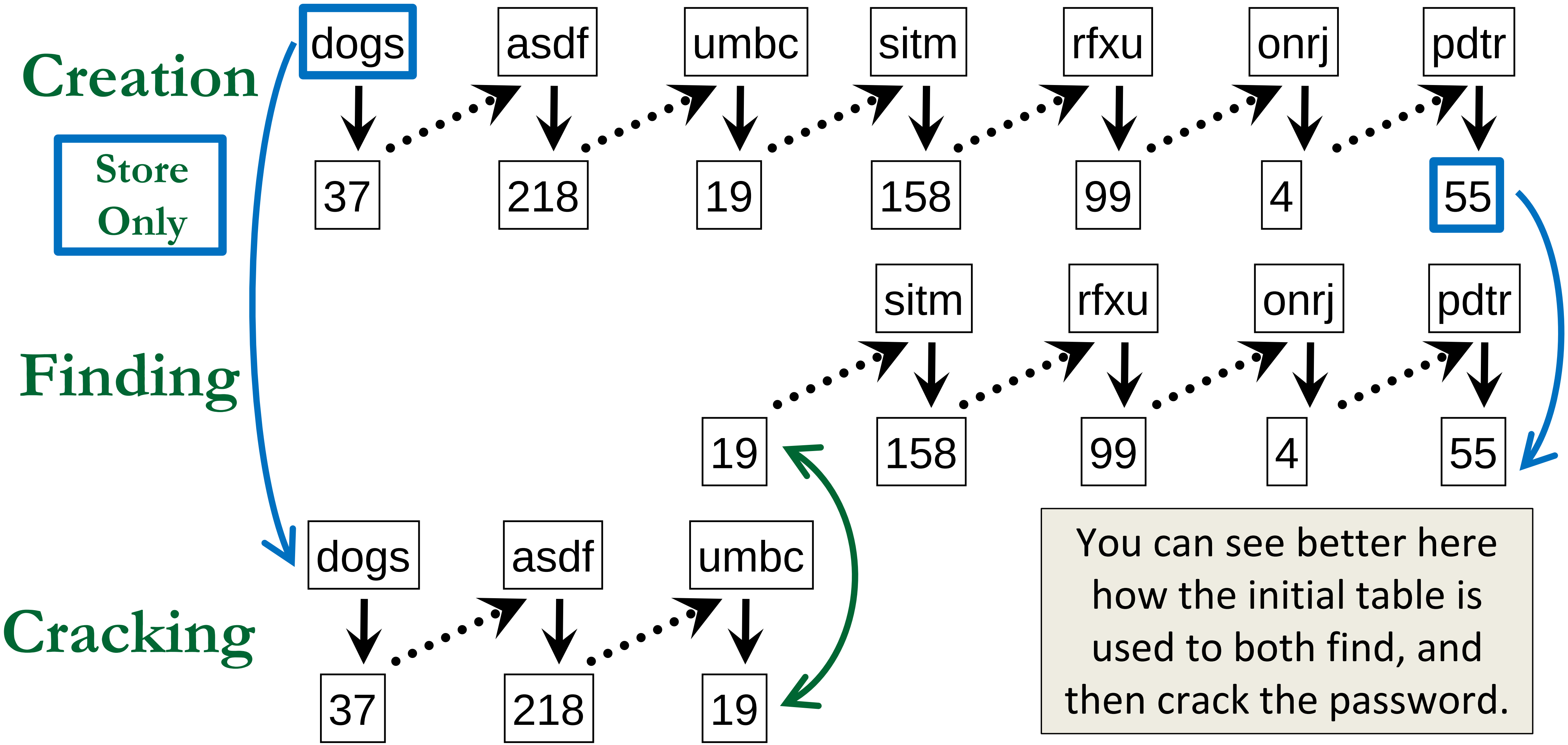
Doesn’t match the user’s password hash of **<u>19</u>**

asdf → 218 ✗

umbc → 19 ✓

Previous step is the password!

Rainbow Tables Example: Overlap



Creation

Store Only

dogs asdf umbc sitm rfxu onrj pdtr

37 218 19 158 99 4 55

Finding

sitm rfxu onrj pdtr

19 158 99 4 55

Cracking

dogs asdf umbc

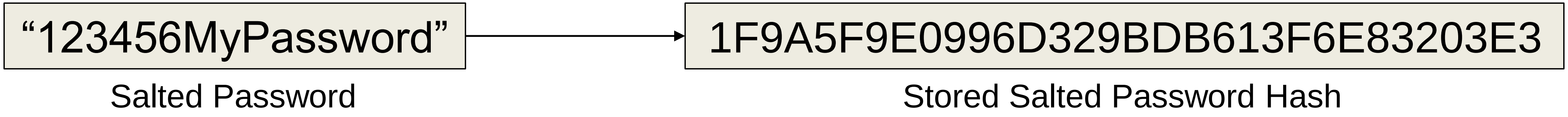
37 218 19

You can see better here how the initial table is used to both find, and then crack the password.

Salting

Salted Passwords

- Password salting is a defense against dictionary attacks and rainbow tables
- When a user creates an account on a system, a random salt is generated for them
- The salt is prepended to their password before it is hashed



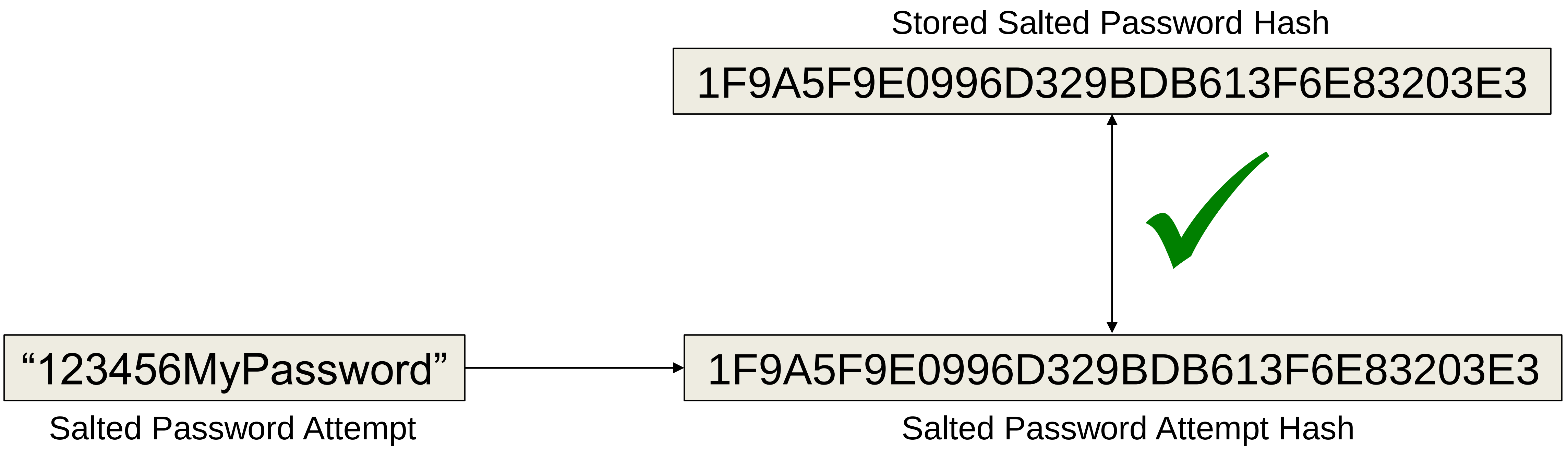
“123456MyPassword” → 1F9A5F9E0996D329BDB613F6E83203E3

Salted Password

Stored Salted Password Hash

Salted Password Usage

- When a user attempts to log in, the salt and password they enter are hashed together and compared to the one on disk



Stored Salted Password Hash

1F9A5F9E0996D329BDB613F6E83203E3

✓

“123456MyPassword”

Salted Password Attempt

1F9A5F9E0996D329BDB613F6E83203E3

Salted Password Attempt Hash

Salted Passwords

- Password salts are stored in plaintext
  - They don’t need to be hidden from an attacker to be effective
  - Why?
- For a *b*-bit salt, the number of possible passwords is increased by a factor of 2^b bits
- Because each user has their own salt, attacks involving precomputed hashes cannot be reused

Image Sources

- Rainbow:
  - https://commons.wikimedia.org/wiki/File:Rainbow-diagram-ROYGBIV.svg