CY 2550 Foundations of Cybersecurity

TLS

Passwords and Authentication

Alina Oprea

Associate Professor, Khoury College

Northeastern University

Outline

- TLS protocol
  - Handshake and record protocols
- Password authentication

Announcements:

- Crypto homework due on Feb. 14
- Holiday: Monday, Feb. 17
- Midterm exam: February 20 in class
- Ethics session
  - February 24 and 27

SSL / TLS Guarantees

- End-to-end secure communications at transport layer in the presence of a network attacker
  - Attacker completely owns the network: controls Wi-Fi, DNS, routers, his own websites, can listen to any packet, modify packets in transit, inject his own packets into the network
- Properties
  - Authentication of server (optionally, client authentication)
  - Confidentiality of communication
  - Integrity against active attacks

TLS Basics

- TLS consists of two protocols
- Handshake protocol
  - Session initiation by client
  - Uses public-key cryptography to establish several shared secret keys between the client and the server
  - Server must have an asymmetric keypair
    - X.509 certificates contain signed public keys rooted in PKI
- Record protocol
  - Uses the secret keys established in the handshake protocol to protect confidentiality and integrity of data exchange between the client and the server

TLS Handshake Protocol

- Runs between a client and a server
  - Client = Web browser
  - Server = website
- Negotiate version of the protocol and the set of cryptographic algorithms to be used
  - Interoperability between different implementations
- Authenticate server
  - Use digital certificates to learn server's public keys and verify the certificate
  - Client authentication is optional
- Use public keys to establish a shared secret

Handshake Protocol Structure

ClientHello

ServerHello

ServerKeyExchange

Agree on crypto algorithms
Server certificate

CertificateVerify

ClientKeyExchange

Send session key using public-key encryption

Derive secret keys for crypto algorithms

Extract session key
Derive secret keys for crypto algorithms

- Common algorithms
- Session key
- Crypto keys

Finished

Finished

Record Protocol Structure

Encryption (m)

- Mac-then-Enc using keys derived from the session key
- MAC uses a counter to prevent replay attacks
- Provides authenticated encryption

Decryption(c)

First decrypt, then check MAC

Encryption (m)

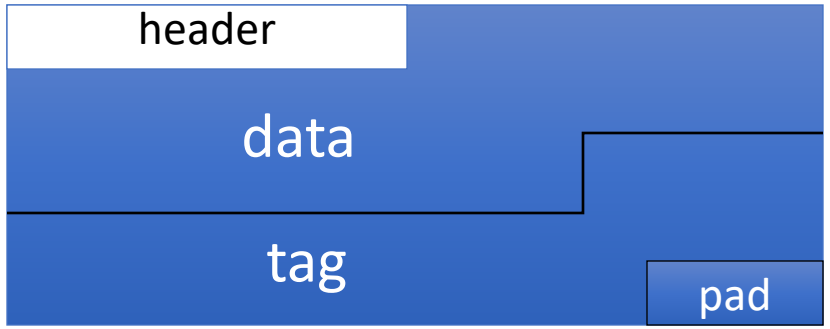
Similar algorithm, but needs different keys (set of keys for each communication direction)

Decryption(c)

First decrypt, then check MAC

TLS record: encryption (CBC AES-128, HMAC-SHA1)

Client: (k_1 , k_2)



Client side **Enc(k_1 , k_2 ,data, ctr) :**

Step 1: tag ⟵ **Tag**(k_2 , [++ctr ll header ll data])

Step 2: pad [header ll data ll tag] to AES block size

Step 3: AES-CBC encrypt with k_1 and new random IV

TLS record: decryption (CBC AES-128, HMAC-SHA1)

Server side **Dec(k_1 , k_2 , c, ctr)** :

Step 1: CBC decrypt c using k_1

Step 2: check pad format: send bad_record_mac if invalid

Step 3: check tag on [++ctr ll header ll data]

send bad_record_mac if invalid

Provides authenticated encryption

(provided no other information is leaked during decryption)

Review Applied Cryptography

- Encryption: Confidentiality
  - Symmetric-key (e.g., CBC-AES, CTR-AES)
  - Public-key (e.g., RSA OAEP)
- Message Authentication Codes (MACs): Integrity
  - E.g., HMAC
- Signature schemes: Integrity and (Weak) Authentication
  - E.g., RSA Full-Domain-Hash
- Hash functions
  - Used for designing MAC and in signature schemes
  - E.g., SHA-2, SHA-3
- PKI: Authentication
  - Distribution of public keys on the Internet

Review: Applied Cryptography

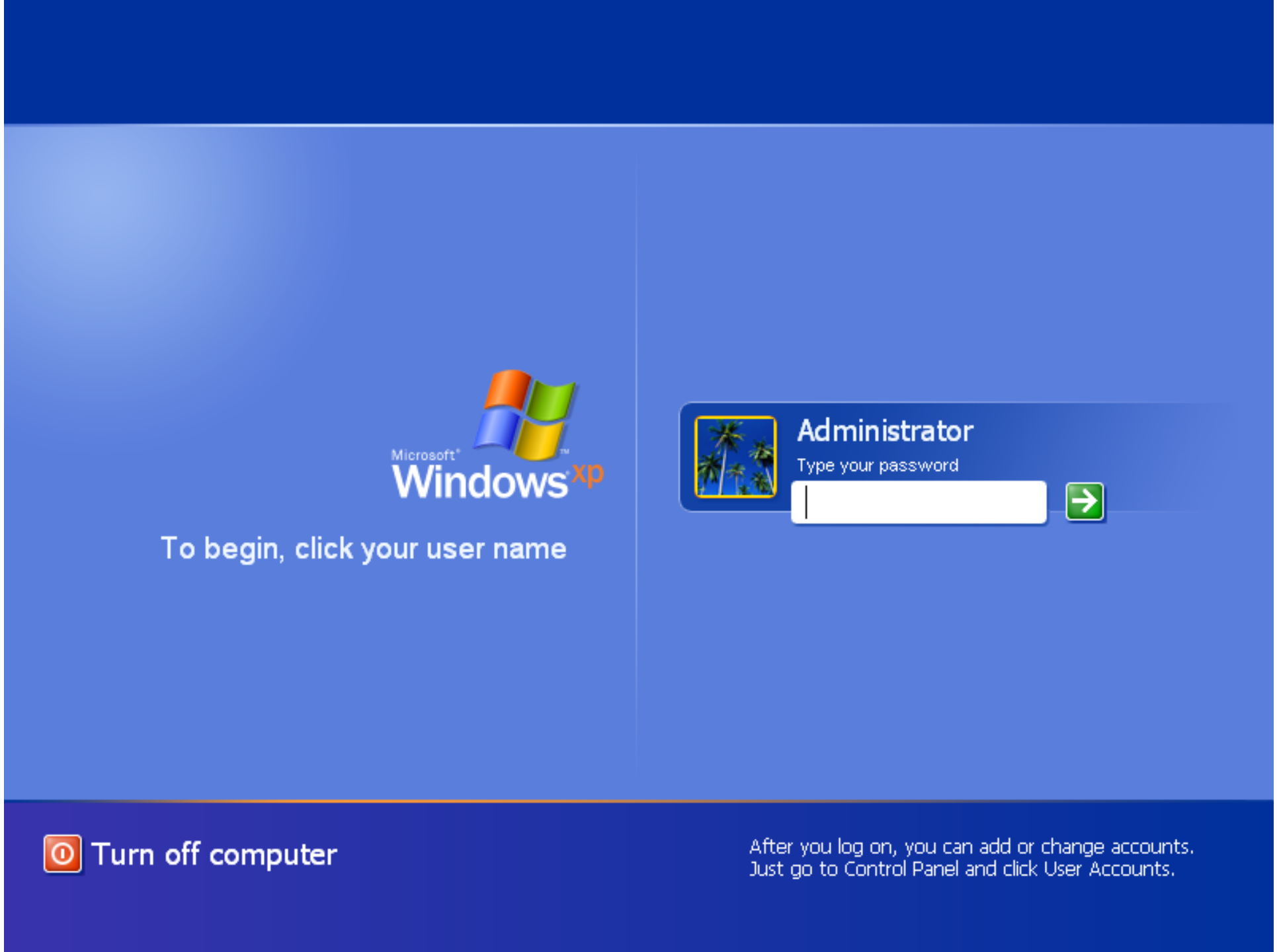
- Crypto is a powerful tool
- Practical crypto relies on computational assumptions, such as factoring
- Use crypto to build a secure system is difficult
  - By composing crypto primitive the result is not always secure!
- Example 1: IV in CBC encryption to prevent chosen plaintext attacks
  - Incorrect implementations may create vulnerabilities to CPA attacks as in TLS v1 (predictable IV)
- Example 2: order of encryption and MAC for both confidentiality and integrity
  - Incorrect ordering could invalidate both properties
- Practical issues
  - Combining public key and private key encryption

Passwords and Authentication

Password storage

Password cracking

Password recovery


Microsoft
Windows xp
To begin, click your user name
Administrator
Type your password
Turn off computer
After you log on, you can add or change accounts.
Just go to Control Panel and click User Accounts.

Authentication

- Authentication is the process of verifying an actor's identity
- Critical for security of systems
  - Permissions, capabilities, and access control are all contingent upon knowing the identity of the actor
- Typically parameterized as a username and a secret
  - The secret attempts to limit unauthorized access
- Desirable properties of secrets include being *unforgeable* and *revocable*

Types of Secrets

- Actors provide their secret to log-in to a system
- Three classes of secrets:
  1. Something you know
     - Example: a password or PIN
  2. Something you have
     - Examples: a smart card, smart phone, or hardware token
  3. Something you are
     - Examples: fingerprint, voice scan, iris scan

Password Storage

Hashing and Salting

Key Stretching and Work Factor

Attacker Goals and Threat Model

- Assume we have a system storing usernames and passwords
- The attacker has access to the password database/file

Database

User	Password
charlie	p4ssW0rd
sandi	puppies
alice	3spr3ss0

I wanna login to those user accounts!

Cracked Passwords

User	Password
charlie	p4ssW0rd
sandi	puppies
alice	3spr3ss0

Checking Passwords

- System must validate passwords provided by users
- Thus, passwords must be stored somewhere
- Basic storage: plain text

password.txt	
charlie	p4ssw0rd
sandi	i heart doggies
alice	93Gd9#jv*0x3N
bob	security

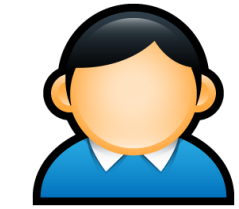
Problem: Password File Theft

- Attackers often compromise systems
- They may be able to steal the password file
  - Linux: /etc/shadow
  - Windows: c:\windows\system32\config\sam
- If the passwords are plain text, what happens?
  - The attacker can now log-in as any user, including root/administrator
  - Moreover, attacker will get access to other machines where users have access
- **Passwords should never be stored in plain text**

Hashed Passwords

- Key idea: store “scrambled” versions of passwords
  - Use one-way cryptographic hash functions
  - Examples: MD5, SHA1, SHA256, SHA512, bcrypt, PBKDF2, scrypt
- Cryptographic hash function transform input data into scrambled output data
  - Deterministic: H(A) is always the same
  - High entropy:
    - H(‘security’) = e91e6348157868de9dd8b25c81aebfb9
    - H(‘security1’) = 8632c375e9eba096df51844a5a43ae93
    - H(‘Security’) = 2fae32629d4ef4fc6341f1751b405e45
  - Collision resistant
    - Locating A’ such that H(A) = H(A’) takes a long time (hopefully)

Hashed Password Example



User: charlie

H(‘p4ssw0rd’) =
2a9d119df47ff993b662a8ef36f9ea20

H(‘password’) =
b35596ed3f0d5134739292faa04f7ca3

hashed_passw…xt	
charlie	2a9d119df47ff993b662a8ef36f9ea20
sandi	23eb06699da16a3ee5003e5f4636e79f
alice	98bd0ebb3c3ec3fbe21269a8d840127c
bob	e91e6348157868de9dd8b25c81aebfb9

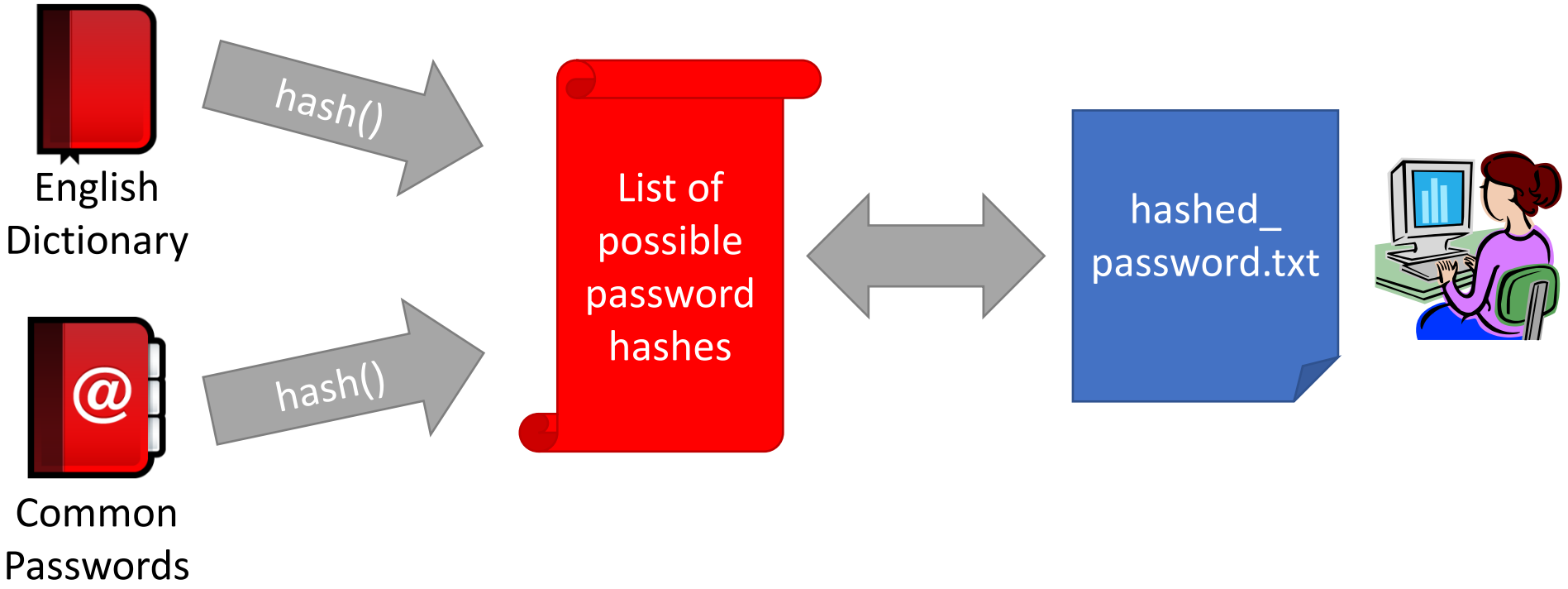
Attacking Password Hashes

- Recall: cryptographic hashes are collision resistant
  - Locating A’ such that H(A) = H(A’) takes a long time (hopefully)
- Are hashed password secure from cracking?
  - **No!**
- Problem: users choose poor passwords
  - Most common passwords: 123456, password
  - Username: alice, Password: alice
- Weak passwords enable dictionary attacks

Most Common Passwords

Rank	2013	2014
1	123456	123456
2	password	password
3	12345678	12345
4	qwerty	12345678
5	abc123	qwerty
6	123456789	123456789
7	111111	1234
8	1234567	baseball
9	iloveyou	dragon
10	adobe123	football

Dictionary Attacks



- Common for 60-70% of hashed passwords to be cracked in <24 hours

Basic Password Cracking

- Problem: humans are terrible at generating/remembering random strings
- Passwords are often weak enough to be brute-forced
  - Naïve way: systematically try all possible passwords
  - Slightly smarter way: take into account non-uniform distribution of characters
- Dictionary attacks are also highly effective
  - Select a baseline wordlist/dictionary full of **likely** passwords
    - Today, the best wordlists come from lists of breached passwords
  - Rule-guided word mangling to look for slight variations
    - E.g. password → Password → p4ssword → passw0rd → p4ssw0rd → password1 → etc.
- Many password cracking tools exist (e.g. John the Ripper, hashcat)

Speeding Up Brute-Force Cracking

- Brute force attacks are slow because hashing is CPU intensive
  - Especially if a strong function (SHA512) is used
- Idea: why not pre-compute and store all hashes?
  - You would only need to pay the CPU cost once…
- Given a hash function *H*, a target hash *h*, and password space *P*, goal is to recover $p \in P$ such that $H(p) = h$
- Problem: naïve approach requires *lots of storage*
  - *O(|P||H|)*

Hash Chains: Cracking Tough Passwords

- Hash chains enable time-space efficient reversal of hash functions
- Key idea: pre-compute chains of passwords of length *k*…
  - … but only store the start and end of each chain
  - Larger *k* → fewer chains to store, more CPU cost to rebuild chains
  - Small *k* → more chains to store, less CPU cost to rebuild chains
- Building chains require *H*, as well as a reduction $R : H \mapsto P$
  - Begin by selecting some initial set of password $P' \subset P$
  - For each $p' \in P'$, apply $H(p') = h', R(h') = p''$ for *k* iterations
  - Only store p' and p'^k
- To recover hash *h*, apply R and H until the end of a chain is found
  - Rebuild the chain using p' and p'^k
  - *H(p) = h* may be within the chain

Uncompressed Hash Chain Example

Only these two columns get stored on disk

p'	H(p') = h'	R(h') = p''	H(p'') = h''	R(h'') = p'''	H(p''') = h'''	R(h''') = p*
abcde	\\WPNP_	vlsfqp	_QOZLR	eusrqv	CMRQ5X	cjldar
passw	VZDGEF	gfnxsk	ZLGEKV	yookol	EBOTHT	zvxscs
12345	SM-QK\9	sawtzg	RHKP_D	gvmdwm	BYE4LB	wjizbn
secrt	OKFTaY	btweoz	WA15HK	ttgovl	Q_4\6ZB	eivlqc

K = 3

Hash Chain Example

p'	p*
abcde	cjldar
passw	zvxscs
12345	wjizbn
secrt	eivlqc

K = 3

- Size of the table is dramatically reduced…
- … but some computation is necessary once a match is found

p'	H(p') = h'	R(h') = p''	H(p'') = h''	R(h'') = p'''	H(p''') = h'''	R(h''') = p*
12345	SM-QK\9	sawtzg	RHKP_D			wjizbn

p	H(p) = h	R(h) = p'	H(p') = h'	R(h') = p''	H(p'') = h''	R(h'') = p'''	H(p''') = h'''
	RHKP_D	gvmdwm	BYE4LB	wjizbn			

Desired password

Hash to recover

Problems with Hash Chains

- Hash chains are prone to collisions
  - Collisions occur when *H(p') = H(p")* or *R(h') = R(h")* (the latter is more likely)
  - Causes the chains to merge or overlap
- Problems caused by collisions
  - Wasted space in the file, since the chains cover the same password space
  - False positives: a chain may not include the password even if the end matches
- Proper choice of *R()* is critical
  - Goal is to cover ***likely*** password space, not entire password space
  - *R* cannot be collision resistant (like *H*) since it has to map into likely plaintexts
  - Rainbow tables use multiple reduction functions and can be downloaded for free (for passwords up to 14 characters)
- Conclusion: storing hashes of passwords is vulnerable to brute forcing and rainbow table attacks