CNT4406/5412 Network Security

Authentication

Zhi Wang

Florida State University

Fall 2014

Introduction

- Authentication is the process of reliably verifying an entity's identity
  ➠ e.g., user/computer authentication, message authentication...

Introduction

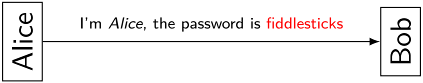
- Authentication is the process of reliably verifying an entity's identity
  ➠ e.g., user/computer authentication, message authentication...
- Authentication mechanisms

Introduction

- Authentication is the process of reliably verifying an entity's identity
  - ➠ e.g., user/computer authentication, message authentication...
- Authentication mechanisms
  - ➠ password-based authentication
  - ➠ address-based authentication
  - ➠ cryptographic authentication protocols

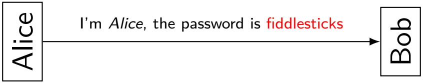
Password-based Authentication

- **Password-based authentication** uses a secret quantity that you state to prove you know it
  ➠ "It's not who you know. It's what you know."



Password-based Authentication

- **Password-based authentication** uses a secret quantity that you state to prove you know it
  ➠ "It's not who you know. It's what you know."
- Threats: eavesdropping, password guessing (dictionary attack)



Address-based Authentication

- **Address-based authentication** assumes the identity of the source can be inferred based on the address (network address/email address) from which messages arrive
  ➠ "It's not what you know. It's where you are."

Address-based Authentication

- **Address-based authentication** assumes the identity of the source can be inferred based on the address (network address/email address) from which messages arrive
  ➠ "It's not what you know. It's where you are."
- It was adopted in early UNIX systems such as *Berkeley rtools*

Address-based Authentication

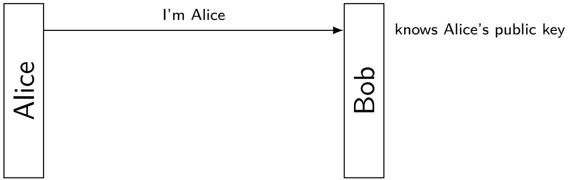
- **Address-based authentication** assumes the identity of the source can be inferred based on the address (network address/email address) from which messages arrive
  ➠ "It's not what you know. It's where you are."
- It was adopted in early UNIX systems such as *Berkeley rtools*
  ➠ `/etc/hosts.equiv`: computers with identical user accounts
  ➠ Per-user `.rhosts`: a list of <computer, account> pairs allowed to access the user's account

Address-based Authentication

- **Address-based authentication** assumes the identity of the source can be inferred based on the address (network address/email address) from which messages arrive
  ➠ "It's not what you know. It's where you are."
- It was adopted in early UNIX systems such as *Berkeley rtools*
  ➠ `/etc/hosts.equiv`: computers with identical user accounts
  ➠ Per-user `.rhosts`: a list of <computer, account> pairs allowed to access the user's account
- Threats: network address spoofing

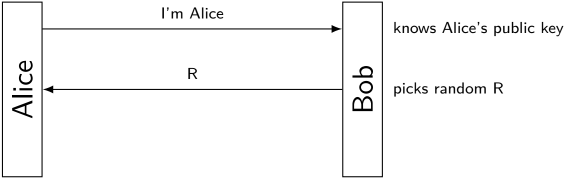
Cryptographic Authentication Protocol

- **Cryptographic authentication** proves one's identity by performing a cryptographic operation on a quantity the verifier supplies
  ➠ secret key cryptography, public key cryptography, hash function



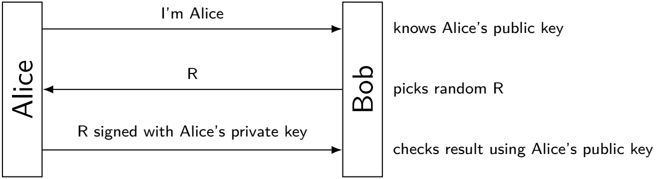
Cryptographic Authentication Protocol

- **Cryptographic authentication** proves one's identity by performing a cryptographic operation on a quantity the verifier supplies
  ➠ secret key cryptography, public key cryptography, hash function



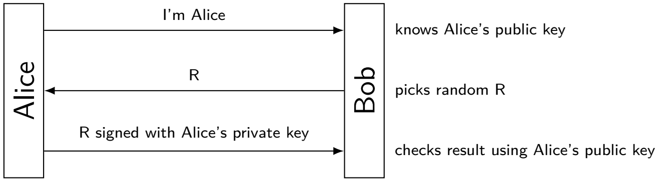
Cryptographic Authentication Protocol

- **Cryptographic authentication** proves one's identity by performing a cryptographic operation on a quantity the verifier supplies
  ➠ secret key cryptography, public key cryptography, hash function



Cryptographic Authentication Protocol

- **Cryptographic authentication** proves one's identity by performing a cryptographic operation on a quantity the verifier supplies
  ➠ secret key cryptography, public key cryptography, hash function
- Threats: brute-force, eavesdropping, server database breach



User Authentication

User authentication can be based on:

- What the user knows (knowledge factor)
  - ➠ passwords, personal information, credit card numbers...

User Authentication

User authentication can be based on:

- What the user knows (knowledge factor)
  ➠ passwords, personal information, credit card numbers...
- What the user has (possession factor)
  ➠ ATM card, keys, USB token

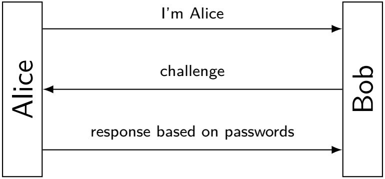
User Authentication

User authentication can be based on:

- What the user knows (knowledge factor)
  - ➠ passwords, personal information, credit card numbers...
- What the user has (possession factor)
  - ➠ ATM card, keys, USB token
- What the user is (inherence factor)
  - ➠ bio-metrics such as voice, fingerprint, iris pattern
  - ➠ benefits and problems?

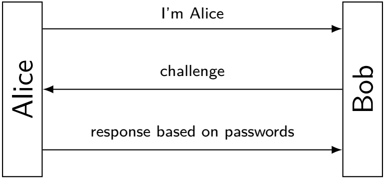
Password-based User Authentication

- **Password-based authentication** uses a secret quantity that you state to prove you know it
  ➠ most command method of user authentication



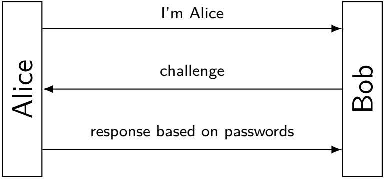
Password-based User Authentication

- **Password-based authentication** uses a secret quantity that you state to prove you know it
  ➠ most command method of user authentication
- Threats to password-based authentication?



Password-based User Authentication

- **Password-based authentication** uses a secret quantity that you state to prove you know it
  ➠ most command method of user authentication
- Threats to password-based authentication?
  ➠ eavesdropping, leaking of stored passwords, online/offline password guessing, memorizing user-unfriendly passwords, password reuse



Issues for Password-based Systems

- Passwords should be easy to remember but hard to guess, so
  - ➠ users can memorize their passwords (instead of writing them down)
  - ➠ online/offline password guessing won't be effective

Issues for Password-based Systems

- Passwords should be easy to remember but hard to guess, so
  - ➠ users can memorize their passwords (instead of writing them down)
  - ➠ online/offline password guessing won't be effective
- Passwords should be securely stored
  - ➠ always assume stored passwords can and will be leaked!!

Issues for Password-based Systems

- Passwords should be easy to remember but hard to guess, so
  - ➠ users can memorize their passwords (instead of writing them down)
  - ➠ online/offline password guessing won't be effective
- Passwords should be securely stored
  - ➠ always assume stored passwords can and will be leaked!!
- Passwords shall not be reused among accounts
  - ➠ bad idea for services to use email address as user name!

Password Storage

- Storing unencrypted passwords is high risk

Password Storage

- Storing unencrypted passwords is high risk
  - ➠ flexible in the authentication protocol design
  - ➠ capturing stored passwords enables impersonating any user

Password Storage

- Storing unencrypted passwords is high risk
  - ➠ flexible in the authentication protocol design
  - ➠ capturing stored passwords enables impersonating any user
- Better idea: encrypting the password database

Password Storage

- Storing unencrypted passwords is high risk
  ➠ flexible in the authentication protocol design
  ➠ capturing stored passwords enables impersonating any user
- Better idea: encrypting the password database
  ➠ as flexible as storing plaintext passwords
  ➠ The key to decrypt the database and/or decrypted passwords may be stored as plaintext in the memory at run-time

Password Storage

- Storing unencrypted passwords is high risk
  ➠ flexible in the authentication protocol design
  ➠ capturing stored passwords enables impersonating any user
- Better idea: encrypting the password database
  ➠ as flexible as storing plaintext passwords
  ➠ The key to decrypt the database and/or decrypted passwords may be stored as plaintext in the memory at run-time
- Better idea 2: storing the cryptographic hash of the password

Password Storage

- Storing unencrypted passwords is high risk
  ➠ flexible in the authentication protocol design
  ➠ capturing stored passwords enables impersonating any user
- Better idea: encrypting the password database
  ➠ as flexible as storing plaintext passwords
  ➠ The key to decrypt the database and/or decrypted passwords may be stored as plaintext in the memory at run-time
- Better idea 2: storing the cryptographic hash of the password
  ➠ less flexible than storing plaintext passwords (why?)
  ➠ capturing stored passwords allows for offline password guessing

Dictionary Attack

Dictionary attack is a technique to determine decryption key or pass-phrase by exhaustively searching a pre-defined list of values.

Dictionary Attack

Dictionary attack is a technique to determine decryption key or pass-phrase by exhaustively searching a pre-defined list of values.

- The pre-defined list used to come from words in real dictionaries + simple transformations (e.g., appending digits)

Dictionary Attack

Dictionary attack is a technique to determine decryption key or pass-phrase by exhaustively searching a pre-defined list of values.

- The pre-defined list used to come from words in real dictionaries + simple transformations (e.g., appending digits)
- Millions of leaked passwords (plaintext) have provided both a comprehensive real-world list and patterns of password construction

Dictionary Attack

Dictionary attack is a technique to determine decryption key or pass-phrase by exhaustively searching a pre-defined list of values.

- The pre-defined list used to come from words in real dictionaries + simple transformations (e.g., appending digits)
- Millions of leaked passwords (plaintext) have provided both a comprehensive real-world list and patterns of password construction
  ➠ A SQL injection exposed 32M RockYou.com passwords in 2009

Example: Leaked Yahoo Passwords (Hashed)

442, 773 passwords were leaked, 342, 478 of them were unique, 100, 295 (22.65%) passwords were used by more than one person

Example: Leaked Yahoo Passwords (Hashed)

442, 773 passwords were leaked, 342, 478 of them were unique, 100, 295 (22.65%) passwords were used by more than one person

123456	1666	0.38%
password	780	0.18%
welcome	436	0.1%
ninja	333	0.08%
abc123	250	0.06%
123456789	222	0.05%
12345678	208	0.05%
sunshine	205	0.05%
princess	202	0.05%
qwerty	172	0.04%

top 10 passwords

Example: Leaked Yahoo Passwords (Hashed)

442,773 passwords were leaked, 342,478 of them were unique, 100,295 (22.65%) passwords were used by more than one person

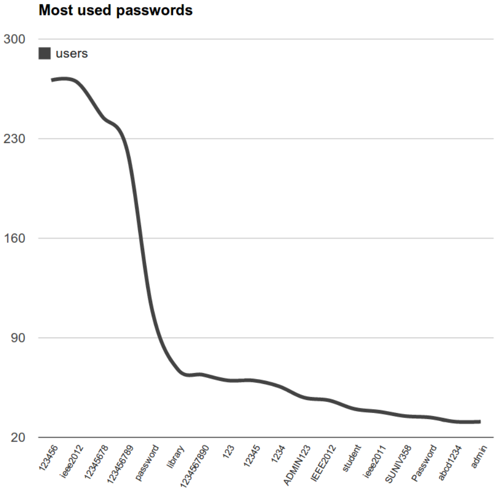
123456	1666	0.38%
password	780	0.18%
welcome	436	0.1%
ninja	333	0.08%
abc123	250	0.06%
123456789	222	0.05%
12345678	208	0.05%
sunshine	205	0.05%
princess	202	0.05%
qwerty	172	0.04%

top 10 passwords

password	1373	0.31%
welcome	534	0.12%
qwerty	464	0.1%
monkey	430	0.1%
jesus	429	0.1%
love	421	0.1%
money	407	0.09%
freedom	385	0.09%
ninja	380	0.09%
writer	367	0.08%

top 10 base words

IEEE Data Leak (2012)

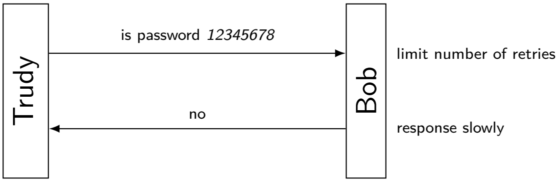


Online Dictionary Attack

- Try the passwords from the list online one-by-one
- Easy to prevent, how?

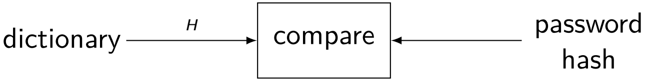
Online Dictionary Attack

- Try the passwords from the list online one-by-one
- Easy to prevent, how?
  - ➠ limit number of retries (e.g., ATM card)
  - ➠ process the password really s l o w l y



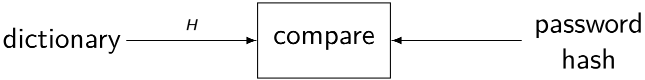
Offline Dictionary Attack

- Obtain the hash(es) of the password(s), compute the hashes of the dictionary, then look for matches



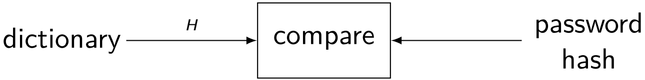
Offline Dictionary Attack

- Obtain the hash(es) of the password(s), compute the hashes of the dictionary, then look for matches
- Hashes of the dictionary can be pre-computed, then used to match one or many password hashes (design challenges?)



Offline Dictionary Attack

- Obtain the hash(es) of the password(s), compute the hashes of the dictionary, then look for matches
- Hashes of the dictionary can be pre-computed, then used to match one or many password hashes (design challenges?)
  - ➠ need efficient ways to store and search pre-computed hash
  - ➠ hash chain and rainbow table are used to reduce storage space



Hash Chains

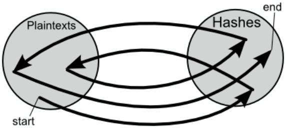
A hash chain is the successive application of a cryptographic hash function (H) and a reduction function (R) to a piece of data

- Reduction function maps a hash value to a (different) password

Hash Chains

A hash chain is the successive application of a cryptographic hash function (H) and a reduction function (R) to a piece of data

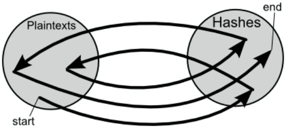
- Reduction function maps a hash value to a (different) password
- To create a hash chain:



Hash Chains

A hash chain is the successive application of a cryptographic hash function (H) and a reduction function (R) to a piece of data

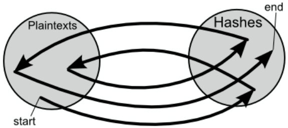
- Reduction function maps a hash value to a (different) password
- To create a hash chain:
  - ➠ starts with an initial passwords (start point)



Hash Chains

A hash chain is the successive application of a cryptographic hash function (H) and a reduction function (R) to a piece of data

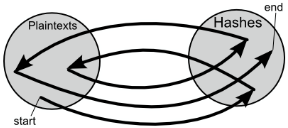
- Reduction function maps a hash value to a (different) password
- To create a hash chain:
  - ➠ starts with an initial passwords (start point)
  - ➠ successively apply the hash and reduction function (n times)



Hash Chains

A hash chain is the successive application of a cryptographic hash function (H) and a reduction function (R) to a piece of data

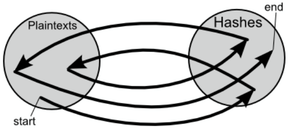
- Reduction function maps a hash value to a (different) password
- To create a hash chain:
  - ➠ starts with an initial passwords (start point)
  - ➠ successively apply the hash and reduction function (n times)
  - ➠ a hash chain is stored as $< startpoint, endpoint >$



Hash Chains

A hash chain is the successive application of a cryptographic hash function (H) and a reduction function (R) to a piece of data

- Reduction function maps a hash value to a (different) password
- To create a hash chain:
  - ➠ starts with an initial passwords (start point)
  - ➠ successively apply the hash and reduction function (n times)
  - ➠ a hash chain is stored as $< startpoint, endpoint >$



$$123456 \xrightarrow{H} 0x873D \xrightarrow{R} 2@kdu \xrightarrow{H} 0x6CE7 \xrightarrow{R} \cdots \xrightarrow{R} jStn4 \xrightarrow{H} 0x854D$$

Hash Chains...

- To match against hash chains:

$$123456 \xrightarrow{H} 0x873D \xrightarrow{\color{red}R} 2@kdu \xrightarrow{H} 0x6CE7 \xrightarrow{\color{red}R} \cdots \xrightarrow{\color{red}R} jStn4 \xrightarrow{H} 0x854D$$

Hash Chains...

- To match against hash chains:
  ➠ starts with the hash of a password

$$123456 \xrightarrow{H} 0x873D \xrightarrow{R} 2@kdu \xrightarrow{H} 0x6CE7 \xrightarrow{R} \cdots \xrightarrow{R} jStn4 \xrightarrow{H} 0x854D$$

Hash Chains...

- To match against hash chains:
  ➠ starts with the hash of a password
  ➠ successively apply the reduction and hash function until hitting an end point, assume its $< s_n, e_n >$

$$123456 \xrightarrow{H} 0x873D \xrightarrow{R} 2@kdu \xrightarrow{H} 0x6CE7 \xrightarrow{R} \cdots \xrightarrow{R} jStn4 \xrightarrow{H} 0x854D$$

Hash Chains...

- To match against hash chains:
  ➠ starts with the hash of a password
  ➠ successively apply the reduction and hash function until hitting an end point, assume its $< s_n, e_n >$
  ➠ start from s_n, successively apply the hash and reduction function until the original hash of the password is reached.

$$\text{123456} \xrightarrow{H} \text{0x873}D \xrightarrow{R} \text{2@kdu} \xrightarrow{H} \text{0x6}CE\text{7} \xrightarrow{R} \cdots \xrightarrow{R} \text{jStn4} \xrightarrow{H} \text{0x854}D$$

Hash Chains...

- To match against hash chains:
  ➠ starts with the hash of a password
  ➠ successively apply the reduction and hash function until hitting an end point, assume its $< s_n, e_n >$
  ➠ start from s_n, successively apply the hash and reduction function until the original hash of the password is reached.
  ➠ The last password is the answer.

$$123456 \xrightarrow{H} 0x873D \xrightarrow{R} 2@kdu \xrightarrow{H} 0x6CE7 \xrightarrow{R} \cdots \xrightarrow{R} jStn4 \xrightarrow{H} 0x854D$$

Rainbow Table

- If two hash chains collide, they will merge thus reducing the passwords covered

*image from Cryptohaze.com

Rainbow Table

- If two hash chains collide, they will merge thus reducing the passwords covered
- Rainbow table uses multiple reduction functions (like colors in a rainbow) to mitigate hash chain collision*

* image from Cryptohaze.com

Rainbow Table

- If two hash chains collide, they will merge thus reducing the passwords covered
- Rainbow table uses multiple reduction functions (like colors in a rainbow) to mitigate hash chain collision*

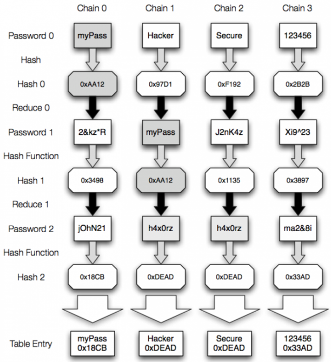
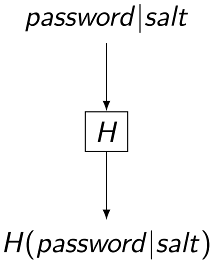


*image from Cryptohaze.com

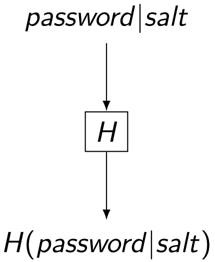
Password Salt

- Pre-computed rainbow table makes dictionary attack more effective, to mitigate it, apply salt



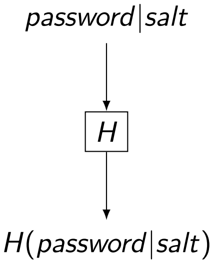
Password Salt

- Pre-computed rainbow table makes dictionary attack more effective, to mitigate it, apply salt
- Salt is a per-user random value append to the password



Password Salt

- Pre-computed rainbow table makes dictionary attack more effective, to mitigate it, apply salt
- Salt is a per-user random value append to the password
  - ➠ password file contains ($username$, $salt$, $H(password|salt)$)*
  - ➠ to verify the password, retrieve the salt from the password file



Password Salt...

Does password salt help to mitigate...?

- online dictionary attack?

Password Salt...

Does password salt help to mitigate...?

- online dictionary attack?
- offline dictionary attack without a rainbow table?

Password Salt...

Does password salt help to mitigate...?

- online dictionary attack?
- offline dictionary attack without a rainbow table?
- offline dictionary attack against one account with a rainbow table?

Password Salt...

Does password salt help to mitigate...?

- online dictionary attack?
- offline dictionary attack without a rainbow table?
- offline dictionary attack against one account with a rainbow table?
- offline dictionary attack against many accounts with a rainbow table?

Example: Linux Password

- A Linux password has the format of `username:$id$salt$encrypted`

Example: Linux Password

- A Linux password has the format of `username:$id$salt$encrypted`
  - `id` identifies the hash function used

id	algorithm
1	MD5
2a	Blowfish
5	SHA-256
6	SHA-512

Example: Linux Password

- A Linux password has the format of `username:$id$salt$encrypted`
  - `id` identifies the hash function used

id	algorithm
1	MD5
2a	Blowfish
5	SHA-256
6	SHA-512

  - `salt` is the salt, a random string up to 16 characters

Example: Linux Password

- A Linux password has the format of `username:$id$salt$encrypted`
  - `id` identifies the hash function used

id	algorithm
1	MD5
2a	Blowfish
5	SHA-256
6	SHA-512

  - `salt` is the salt, a random string up to 16 characters
  - `encrypted` is the hash of `password|salt`

Example: Linux Password...

- Linux passwords are shadowed in `/etc/shadow`
  - ➠ They used to be stored in `/etc/passwd` and universally readable

Example: Linux Password...

- Linux passwords are shadowed in `/etc/shadow`
  - ➠ They used to be stored in `/etc/passwd` and universally readable

file	uid	gid	permissions
/etc/passwd	root	root	`-rw-r--r--`
/etc/shadow	root	shadow	`-rw-r-----`

Example: Linux Password...

- Linux passwords are shadowed in `/etc/shadow`
  - ➠ They used to be stored in `/etc/passwd` and universally readable

file	uid	gid	permissions
/etc/passwd	root	root	`-rw-r--r--`
/etc/shadow	root	shadow	`-rw-r-----`

Example (crack it!):

```
test:$6$Rtp8odu0$/wklQb4fmKvRQVPbAOx2UHJrjfQSxeBF8f
yLqMhxgmqZTGFQNiBG5LqyRDJ9MNoqRC0Vq3gIHIGUHkTIPhVCb.
```

Other Attacks to Passwords

- Eavesdropping on traffic that may contain unencrypted passwords

Other Attacks to Passwords

- Eavesdropping on traffic that may contain unencrypted passwords
- Man-in-the-middle network attack

Other Attacks to Passwords

- Eavesdropping on traffic that may contain unencrypted passwords
- Man-in-the-middle network attack
- Key logger on the password entry system

Other Attacks to Passwords

- Eavesdropping on traffic that may contain unencrypted passwords
- Man-in-the-middle network attack
- Key logger on the password entry system
- Phishing password entry programs

Other Attacks to Passwords

- Eavesdropping on traffic that may contain unencrypted passwords
- Man-in-the-middle network attack
- Key logger on the password entry system
- Phishing password entry programs
- Social engineering

Password Guidelines

- Generate initial passwords randomly by the system

Password Guidelines

- Generate initial passwords randomly by the system
- Periodically change the passwords

Password Guidelines

- Generate initial passwords randomly by the system
- Periodically change the passwords
- Never reuse the same passwords on multiple sites
  ➠ accounts have different importance, news $<$ email $<$ bank

Password Guidelines

- Generate initial passwords randomly by the system
- Periodically change the passwords
- Never reuse the same passwords on multiple sites
  ➠ accounts have different importance, news $<$ email $<$ bank
- Reject passwords vulnerable to dictionary attacks
  ➠ remember the top 10 leaked Yahoo passwords?

Password Guidelines

- Generate initial passwords randomly by the system
- Periodically change the passwords
- Never reuse the same passwords on multiple sites
  ➠ accounts have different importance, news $<$ email $<$ bank
- Reject passwords vulnerable to dictionary attacks
  ➠ remember the top 10 leaked Yahoo passwords?
- What else?

Lamport's Hash

Lamport's hash is a one-time password scheme

- Initialization: Alice picks her password and a number $n_\circ$ (e.g., 1000); Bob stores $(n_\circ,\ H^{n_\circ}(pwd))$

Lamport's Hash

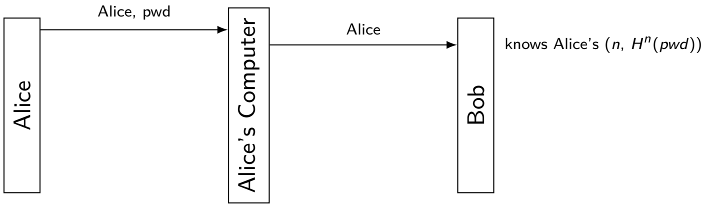
Lamport's hash is a one-time password scheme

- Initialization: Alice picks her password and a number $n_\circ$ (e.g., 1000); Bob stores $(n_\circ,\ H^{n_\circ}(pwd))$
- Authentication: Bob challenges Alice with n; Alice responds with $H^{n-1}(pwd)$

Lamport's Hash

Lamport's hash is a one-time password scheme

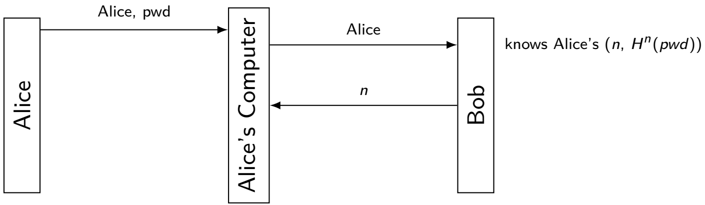
- Initialization: Alice picks her password and a number $n_\circ$ (e.g., 1000); Bob stores $(n_\circ,\ H^{n_\circ}(pwd))$
- Authentication: Bob challenges Alice with n; Alice responds with $H^{n-1}(pwd)$



Lamport's Hash

Lamport's hash is a one-time password scheme

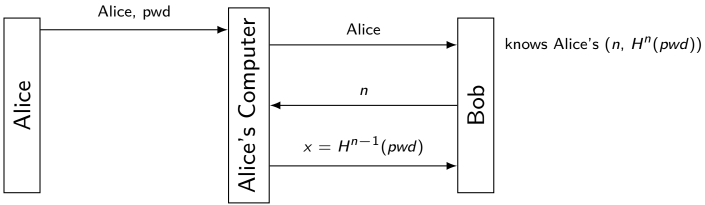
- Initialization: Alice picks her password and a number $n_\circ$ (e.g., 1000); Bob stores $(n_\circ,\ H^{n_\circ}(pwd))$
- Authentication: Bob challenges Alice with n; Alice responds with $H^{n-1}(pwd)$



Lamport's Hash

Lamport's hash is a one-time password scheme

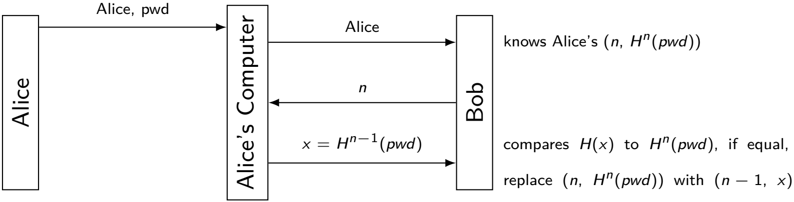
- Initialization: Alice picks her password and a number $n_\circ$ (e.g., 1000); Bob stores $(n_\circ,\ H^{n_\circ}(pwd))$
- Authentication: Bob challenges Alice with n; Alice responds with $H^{n-1}(pwd)$



Lamport's Hash

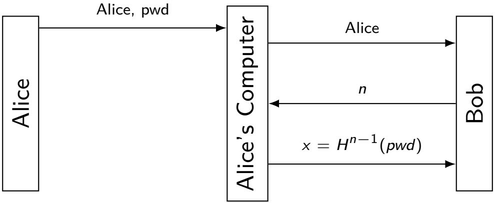
Lamport's hash is a one-time password scheme

- Initialization: Alice picks her password and a number $n_\circ$ (e.g., 1000); Bob stores $(n_\circ,\ H^{n_\circ}(pwd))$
- Authentication: Bob challenges Alice with n; Alice responds with $H^{n-1}(pwd)$



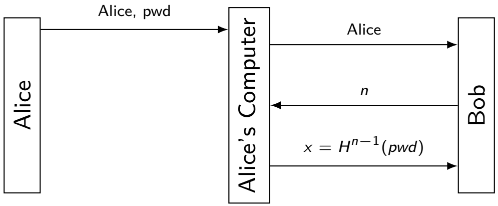
Lamport's Hash...

- Lamport's hash relies on the one-way property of the hash function
  ➠ it is computationally difficult to find $H^{n-1}(pwd)$ from $H^n(pwd)$



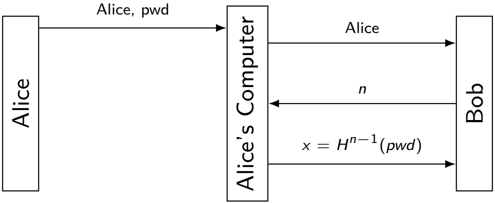
Lamport's Hash...

- Lamport's hash relies on the one-way property of the hash function
  - ➠ it is computationally difficult to find $H^{n-1}(pwd)$ from $H^n(pwd)$
  - ➠ what is the impact of eavesdropping and compromised servers?



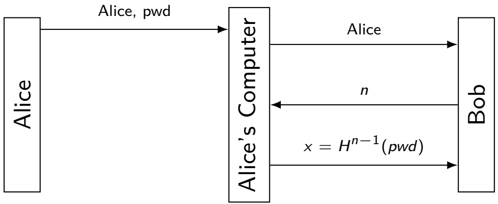
Lamport's Hash...

- Lamport's hash relies on the one-way property of the hash function
  - ➠ it is computationally difficult to find $H^{n-1}(pwd)$ from $H^n(pwd)$
  - ➠ what is the impact of eavesdropping and compromised servers?
- New password has to be installed after authenticating Alice ?? times



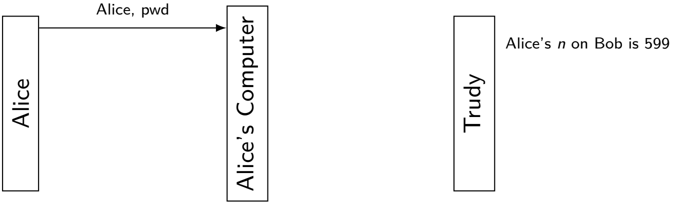
Lamport's Hash...

- Lamport's hash relies on the one-way property of the hash function
  - ➠ it is computationally difficult to find $H^{n-1}(pwd)$ from $H^n(pwd)$
  - ➠ what is the impact of eavesdropping and compromised servers?
- New password has to be installed after authenticating Alice ?? times
- Lamport's hash does not authenticate the server!
  - ➠ man-in-the-middle attack



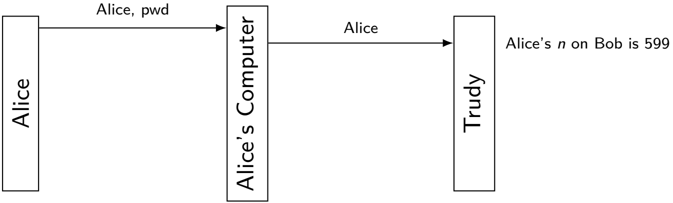
Lamport's Hash: Small *n* Attack

- Trudy impersonates Bob and waits for Alice to connect



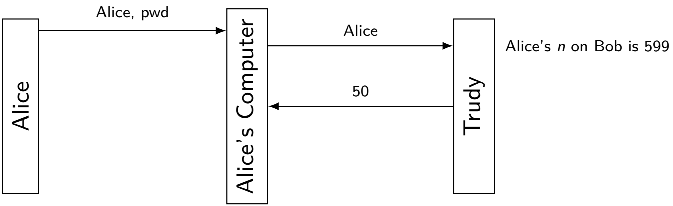
Lamport's Hash: Small *n* Attack

- Trudy impersonates Bob and waits for Alice to connect
- Alice connects to Trudy to authenticate herself



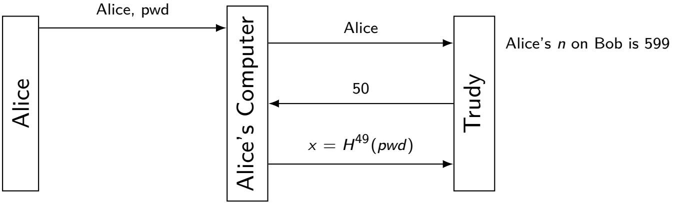
Lamport's Hash: Small *n* Attack

- Trudy impersonates Bob and waits for Alice to connect
- Alice connects to Trudy to authenticate herself
- Trudy sends a small *n* to Alice



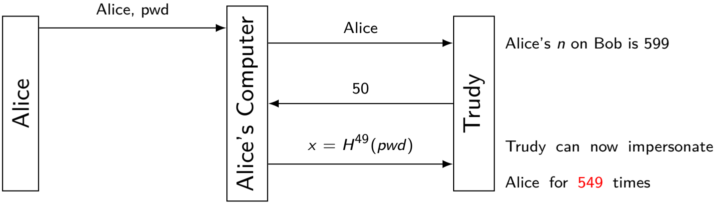
Lamport's Hash: Small n Attack

- Trudy impersonates Bob and waits for Alice to connect
- Alice connects to Trudy to authenticate herself
- Trudy sends a small n to Alice
- Alice responses with $H^{n-1}(pwd)$
  - ➠ Trudy can now response to any challenge $n \geq 50$



Lamport's Hash: Small n Attack

- Trudy impersonates Bob and waits for Alice to connect
- Alice connects to Trudy to authenticate herself
- Trudy sends a small n to Alice
- Alice responses with $H^{n-1}(pwd)$
  ➠ Trudy can now response to any challenge $n \geq 50$



Two Factor Authentication (TFA)

TFA requires the presentation of at least two of the three authentication factors: a *knowledge factor*, a *possession factor*, and an *inherence factor*

Two Factor Authentication (TFA)

TFA requires the presentation of at least two of the three authentication factors: a *knowledge factor*, a *possession factor*, and an *inherence factor*

- Knowledge factor is a password or PIN

Two Factor Authentication (TFA)

TFA requires the presentation of at least two of the three authentication factors: a *knowledge factor*, a *possession factor*, and an *inherence factor*

- Knowledge factor is a password or PIN
- Possession factor has many forms
  - ➠ tokens with a display (disconnected token): RSA Secure ID token
  - ➠ connected tokens: a USB token, smart cards...
  - ➠ mobile phones: Google authenticator

Two Factor Authentication (TFA)

TFA requires the presentation of at least two of the three authentication factors: a *knowledge factor*, a *possession factor*, and an *inherence factor*

- Knowledge factor is a password or PIN
- Possession factor has many forms
  - ➠ tokens with a display (disconnected token): RSA Secure ID token
  - ➠ connected tokens: a USB token, smart cards...
  - ➠ mobile phones: Google authenticator
- Inherence factor is biometrics

Two Factor Authentication (TFA)

TFA requires the presentation of at least two of the three authentication factors: a *knowledge factor*, a *possession factor*, and an *inherence factor*

- Knowledge factor is a password or PIN
- Possession factor has many forms
  - ➠ tokens with a display (disconnected token): RSA Secure ID token
  - ➠ connected tokens: a USB token, smart cards...
  - ➠ mobile phones: Google authenticator
- Inherence factor is biometrics
- Is it considered a TFA if two passwords are required??

Two Factor Authentication...

Disconnected token:

- It usually has a display to show a changing passcode

Two Factor Authentication...

Disconnected token:

- It usually has a display to show a changing passcode
- Passcodes are derived from a shared secret, and

Two Factor Authentication...

Disconnected token:

- It usually has a display to show a changing passcode
- Passcodes are derived from a shared secret, and
  - ➠ sequence-based token: times that a button has been pushed

Two Factor Authentication...

Disconnected token:

- It usually has a display to show a changing passcode
- Passcodes are derived from a shared secret, and
  - ➠ sequence-based token: times that a button has been pushed
  - ➠ time-based token: current time of a wall clock

Two Factor Authentication...

Disconnected token:

- It usually has a display to show a changing passcode
- Passcodes are derived from a shared secret, and
  - ➠ sequence-based token: times that a button has been pushed
  - ➠ time-based token: current time of a wall clock
  - ➠ keypad token: a keypad to input the challenge

Two Factor Authentication...

Disconnected token:

- It usually has a display to show a changing passcode
- Passcodes are derived from a shared secret, and
  - ➠ sequence-based token: times that a button has been pushed
  - ➠ time-based token: current time of a wall clock
  - ➠ keypad token: a keypad to input the challenge

Mobile-phone based token:

Two Factor Authentication...

Disconnected token:

- It usually has a display to show a changing passcode
- Passcodes are derived from a shared secret, and
  - ➠ sequence-based token: times that a button has been pushed
  - ➠ time-based token: current time of a wall clock
  - ➠ keypad token: a keypad to input the challenge

Mobile-phone based token:

- It turns a mobile phone into a token using SMS messaging or downloadable apps

Two Factor Authentication...

Disconnected token:

- It usually has a display to show a changing passcode
- Passcodes are derived from a shared secret, and
  - ➠ sequence-based token: times that a button has been pushed
  - ➠ time-based token: current time of a wall clock
  - ➠ keypad token: a keypad to input the challenge

Mobile-phone based token:

- It turns a mobile phone into a token using SMS messaging or downloadable apps
  - ➠ Google authenticator

Biometrics

Biometric authenticates people by measuring their physical characteristics and matching them against a profile

Biometrics

Biometric authenticates people by measuring their physical characteristics and matching them against a profile

- Biometric authentication should be...

Biometrics

Biometric authenticates people by measuring their physical characteristics and matching them against a profile

- Biometric authentication should be...
  - ➠ uniquely identifying with high accuracy
  - ➠ difficult to forge/mimic
  - ➠ simply and fast to use

Biometrics

Biometric authenticates people by measuring their physical characteristics and matching them against a profile

- Biometric authentication should be...
  - ➠ uniquely identifying with high accuracy
  - ➠ difficult to forge/mimic
  - ➠ simply and fast to use
- Example biometric devices
  - ➠ retinal scanner, fingerprint reader, face recognition, iris scanner, handprint reader, voiceprints, keystroke timing, signatures

Biometrics...

- Biometrics work by turning physical characteristics into a string of data, then match it against a profile
  - ➠ Accuracy, security, and speed depend on the algorithm
  - ➠ the strength of biometric devices is difficult to quantify

Biometrics...

- Biometrics work by turning physical characteristics into a string of data, then match it against a profile
  - ➠ Accuracy, security, and speed depend on the algorithm
  - ➠ the strength of biometric devices is difficult to quantify
- Biometric information may be mechanically copied and cannot be as easily replaced

Biometrics...

- Biometrics work by turning physical characteristics into a string of data, then match it against a profile
  - ➠ Accuracy, security, and speed depend on the algorithm
  - ➠ the strength of biometric devices is difficult to quantify
- Biometric information may be mechanically copied and cannot be as easily replaced
  - ➠ using a picture to bypass face recognition!

Biometrics...

- Biometrics work by turning physical characteristics into a string of data, then match it against a profile
  - ➠ Accuracy, security, and speed depend on the algorithm
  - ➠ the strength of biometric devices is difficult to quantify
- Biometric information may be mechanically copied and cannot be as easily replaced
  - ➠ using a picture to bypass face recognition!
- Biometric information leads to privacy concerns

Google 2-Step Verification

Google allows 2-step verification (TFA) to add an extra layer of security to Google Accounts. Many real-world design considerations:

Google 2-Step Verification

Google allows 2-step verification (TFA) to add an extra layer of security to Google Accounts. Many real-world design considerations:

- Initial setup

Google 2-Step Verification

Google allows 2-step verification (TFA) to add an extra layer of security to Google Accounts. Many real-world design considerations:

- Initial setup
- Smart phone based two factor authentication

Google 2-Step Verification

Google allows 2-step verification (TFA) to add an extra layer of security to Google Accounts. Many real-world design considerations:

- Initial setup
- Smart phone based two factor authentication
  ➠ what if the user lost his phone?

Google 2-Step Verification

Google allows 2-step verification (TFA) to add an extra layer of security to Google Accounts. Many real-world design considerations:

- Initial setup
- Smart phone based two factor authentication
  ➠ what if the user lost his phone?
- Apps that don't support 2-step verification

Google 2-Step Verification...

- Initial SMS/voice setup:

Google 2-Step Verification...

- Initial SMS/voice setup:
  ➠ **web:** link your Google account to a phone number

Google 2-Step Verification...

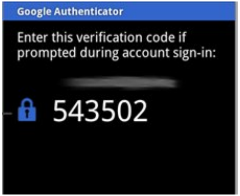
- Initial SMS/voice setup:
  - **web:** link your Google account to a phone number
  - **phone:** receive the initial code by the phone via SMS or voice

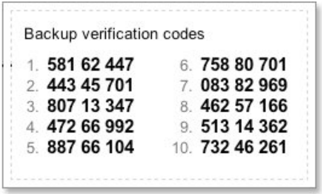
Google 2-Step Verification...

- Initial SMS/voice setup:
  - **web:** link your Google account to a phone number
  - **phone:** receive the initial code by the phone via SMS or voice
  - **web:** verify the code to enable 2-setup verification

Google 2-Step Verification...

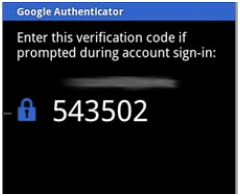
- Smart-phone based two factor authentication

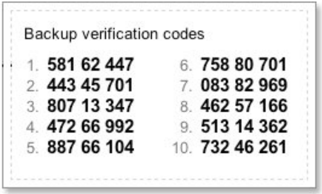




Google 2-Step Verification...

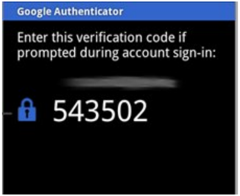
- Smart-phone based two factor authentication
  - ➠ Google Authenticator is an app for Android/iOS/BlackBerry

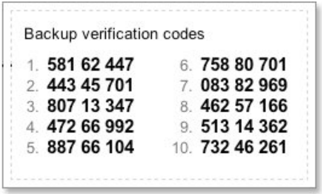




Google 2-Step Verification...

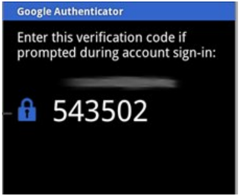
- Smart-phone based two factor authentication
  - ➠ Google Authenticator is an app for Android/iOS/BlackBerry
  - ➠ it is a time-based token that works w/o Internet access

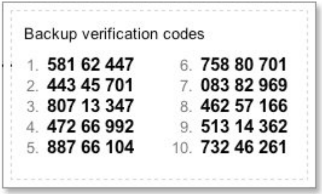




Google 2-Step Verification...

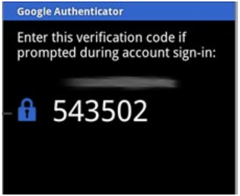
- Smart-phone based two factor authentication
  - ➠ Google Authenticator is an app for Android/iOS/BlackBerry
  - ➠ it is a time-based token that works w/o Internet access
  - ➠ you can have only one active Google Authenticator app

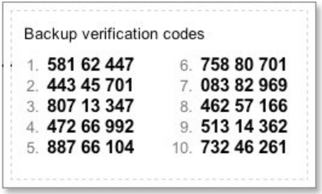




Google 2-Step Verification...

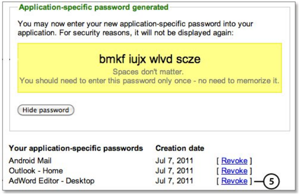
- Smart-phone based two factor authentication
  - ➠ Google Authenticator is an app for Android/iOS/BlackBerry
  - ➠ it is a time-based token that works w/o Internet access
  - ➠ you can have only one active Google Authenticator app
  - ➠ use a backup phone or printable one-time passwords if phone is lost





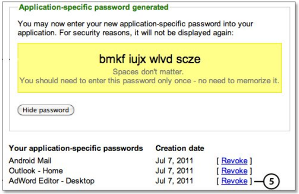
Google 2-Step Verification...

- Generate app-specific keys for apps that don't support 2-step verification
  - lots of apps: Google+, Google Chrome, third-party email clients...
  - most painful part of 2-step verification experience



Google 2-Step Verification...

- Generate app-specific keys for apps that don't support 2-step verification
  - ➠ lots of apps: Google+, Google Chrome, third-party email clients...
  - ➠ most painful part of 2-step verification experience

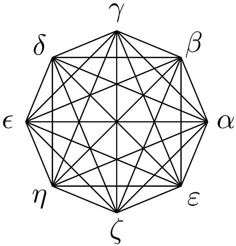


- Mark a computer/device as trusted to avoid further 2-step verification

Key Explosion

Number of keys for pair-wise authentication explodes in large networks:

- Each node needs to know $n-1$ keys

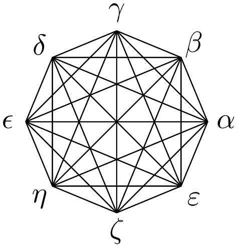


$n = 8$, 28 shared keys

Key Explosion

Number of keys for pair-wise authentication explodes in large networks:

- Each node needs to know $n - 1$ keys
- n new keys need to be installed if a new node joins the network

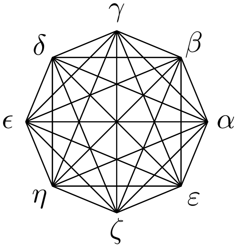


$n = 8$, 28 shared keys

Key Explosion

Number of keys for pair-wise authentication explodes in large networks:

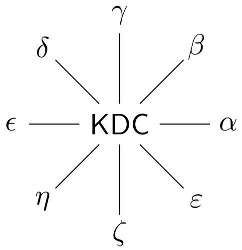
- Each node needs to know $n - 1$ keys
- n new keys need to be installed if a new node joins the network
- in total, $\frac{n(n-1)}{2}$ keys need to be securely distributed!



$n = 8$, 28 shared keys

Key Distribution Center (KDC)

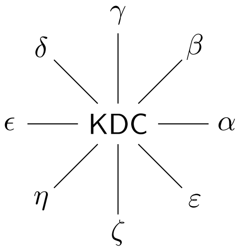
KDC is a trusted node that manages secret keys for the network



Key Distribution Center (KDC)

KDC is a trusted node that manages secret keys for the network

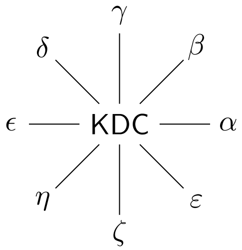
- KDC knows the master key for each node
  - ➠ master key is used for communication between KDC and the node
  - ➠ adding a new node only need to install its master key on KDC



Key Distribution Center (KDC)

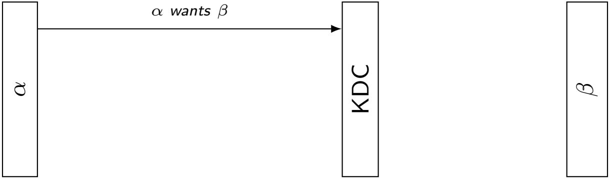
KDC is a trusted node that manages secret keys for the network

- KDC knows the master key for each node
  - ➠ master key is used for communication between KDC and the node
  - ➠ adding a new node only need to install its master key on KDC
- KDC creates and distributes session keys for communications between nodes (how?)



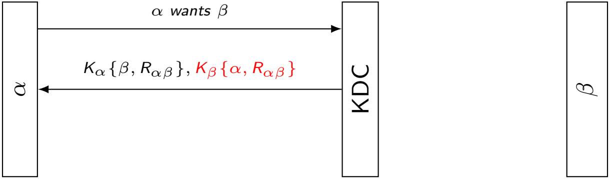
Key Distribution Center: Session Key

- α talks to KDC (securely) to request a key with β



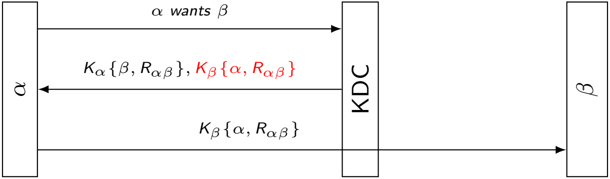
Key Distribution Center: Session Key

- α talks to KDC (securely) to request a key with β
- KDC generates the session key $R_{\alpha\beta}$ and a ticket for β, send them to α



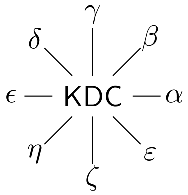
Key Distribution Center: Session Key

- α talks to KDC (securely) to request a key with β
- KDC generates the session key $R_{\alpha\beta}$ and a ticket for β, send them to α
- α forwards the ticket to β, β decrypts it with K_β and get $R_{\alpha\beta}$



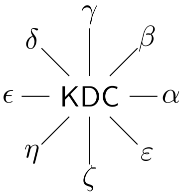
Key Distribution Center: Limitations

- KDC is security critical, it can impersonate any node to any node



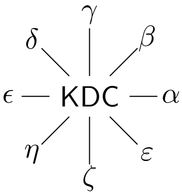
Key Distribution Center: Limitations

- KDC is security critical, it can impersonate any node to any node
- KDC is a single point of failure



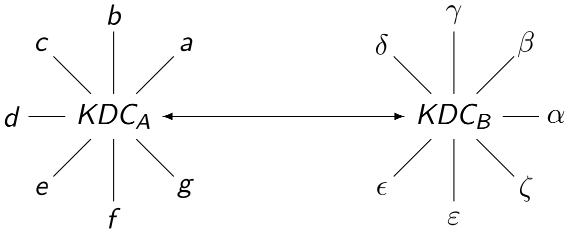
Key Distribution Center: Limitations

- KDC is security critical, it can impersonate any node to any node
- KDC is a single point of failure
- KDC might be a performance bottleneck ➠ replicate KDCs?



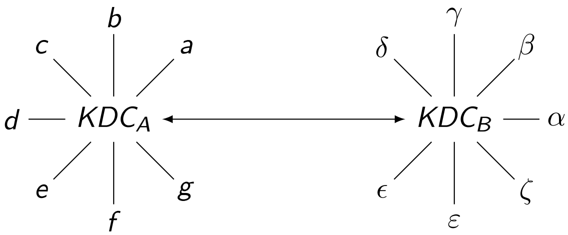
Multiple KDC Domains

- No single KDC will be trusted by all principles in the world
  - ➠ e.g., KGB and CIA, Apple and Google



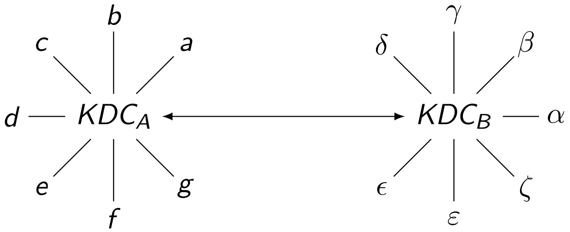
Multiple KDC Domains

- No single KDC will be trusted by all principles in the world
  - ➠ e.g., KGB and CIA, Apple and Google
- Break the world into domains, and let each domain have its own KDC
  - ➠ communication in the same domain remains unchanged



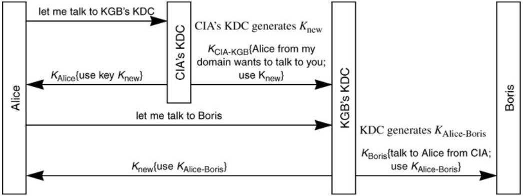
Multiple KDC Domains

- No single KDC will be trusted by all principles in the world
  - ➠ e.g., KGB and CIA, Apple and Google
- Break the world into domains, and let each domain have its own KDC
  - ➠ communication in the same domain remains unchanged
  - ➠ each KDC has a shared key with KDCs it's willing to talk to
  - ➠ communication cross domains require KDC's involvement



Multiple KDC Domains: Cross-domain Key Distribution

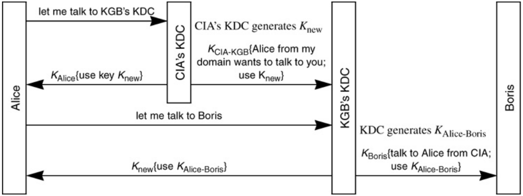
For node a in KDC_1 to communicate with β in KDC_2:



Multiple KDC Domains: Cross-domain Key Distribution

For node a in KDC_1 to communicate with β in KDC_2:

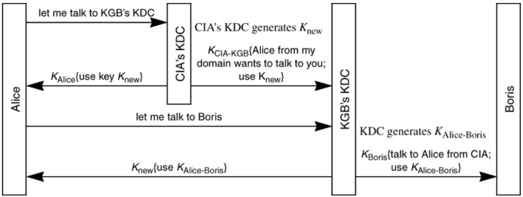
- a requests KDC_1 to create a session key with KDC_2



Multiple KDC Domains: Cross-domain Key Distribution

For node a in KDC_1 to communicate with β in KDC_2:

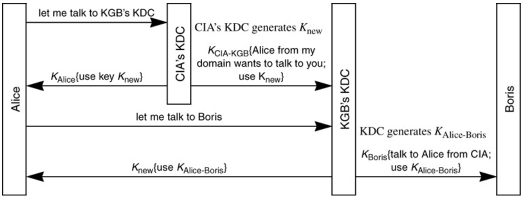
- a requests KDC_1 to create a session key with KDC_2
- a requests KDC_2 to create a session key with β



Multiple KDC Domains: Cross-domain Key Distribution

For node a in KDC_1 to communicate with β in KDC_2:

- a requests KDC_1 to create a session key with KDC_2
- a requests KDC_2 to create a session key with β
- a can now securely talk to β



Certification Authorities (CAs)

CA is a trusted node to generate (sign) certificates that specifies a name and its public key

Certification Authorities (CAs)

CA is a trusted node to generate (sign) certificates that specifies a name and its public key

- CA is the public key equivalent of KDC
  ➠ it solves the problem of how to securely distribute public keys

Certification Authorities (CAs)

CA is a trusted node to generate (sign) certificates that specifies a name and its public key

- CA is the public key equivalent of KDC
  ➠ it solves the problem of how to securely distribute public keys
- Compared to KDC:
  ➠ all nodes only need to be pre-configured with CA's public key

Certification Authorities (CAs)

CA is a trusted node to generate (sign) certificates that specifies a name and its public key

- CA is the public key equivalent of KDC
  ➠ it solves the problem of how to securely distribute public keys
- Compared to KDC:
  ➠ all nodes only need to be pre-configured with CA's public key
  ➠ CA does not need to be on-line: more secure

Certification Authorities (CAs)

CA is a trusted node to generate (sign) certificates that specifies a name and its public key

- CA is the public key equivalent of KDC
  - ➠ it solves the problem of how to securely distribute public keys
- Compared to KDC:
  - ➠ all nodes only need to be pre-configured with CA's public key
  - ➠ CA does not need to be on-line: more secure
  - ➠ network won't be disabled if CA were to crash

Certification Authorities (CAs)

CA is a trusted node to generate (sign) certificates that specifies a name and its public key

- CA is the public key equivalent of KDC
  - ➠ it solves the problem of how to securely distribute public keys
- Compared to KDC:
  - ➠ all nodes only need to be pre-configured with CA's public key
  - ➠ CA does not need to be on-line: more secure
  - ➠ network won't be disabled if CA were to crash
  - ➠ certificates are not security-sensitive, they can be stored anywhere

Certification Authorities (CAs)

CA is a trusted node to generate (sign) certificates that specifies a name and its public key

- CA is the public key equivalent of KDC
  - ➡ it solves the problem of how to securely distribute public keys
- Compared to KDC:
  - ➡ all nodes only need to be pre-configured with CA's public key
  - ➡ CA does not need to be on-line: more secure
  - ➡ network won't be disabled if CA were to crash
  - ➡ certificates are not security-sensitive, they can be stored anywhere
  - ➡ a compromised CA cannot decrypt conversations (why?)

Certificate Revocation

- A certificate has an expiration date, early revocation may be needed
  - ➠ never honor an expired certificate
  - ➠ e.g., an ex-employee

Certificate Revocation

- A certificate has an expiration date, early revocation may be needed
  - ➠ never honor an expired certificate
  - ➠ e.g., an ex-employee
- Certificates are distributed and not easy to revoke (unlike KDC)

Certificate Revocation

- A certificate has an expiration date, early revocation may be needed
  - ➠ never honor an expired certificate
  - ➠ e.g., an ex-employee
- Certificates are distributed and not easy to revoke (unlike KDC)
- **Certificate Revocation List (CRL)** contains a list of unexpired but revoked certificates

Certificate Revocation

- A certificate has an expiration date, early revocation may be needed
  - ➠ never honor an expired certificate
  - ➠ e.g., an ex-employee
- Certificates are distributed and not easy to revoke (unlike KDC)
- **Certificate Revocation List (CRL)** contains a list of unexpired but revoked certificates
  - ➠ CRL has an issue time, do not honor old CRLs

Certificate Revocation

- A certificate has an expiration date, early revocation may be needed
  - ➠ never honor an expired certificate
  - ➠ e.g., an ex-employee
- Certificates are distributed and not easy to revoke (unlike KDC)
- **Certificate Revocation List (CRL)** contains a list of unexpired but revoked certificates
  - ➠ CRL has an issue time, do not honor old CRLs

Summary

- Authentication
- Passwords: storage, dictionary attack, rainbow table, salt
- Lamport's hash
- Two factor authentication
- Biometrics
- Trusted intermediaries: KDC and CA

- Next lecture: Security Handshake