Aalto University

User authentication

Tuomas Aura
CS-C3130 Information security

Aalto University, 2022 course

Outline

1. Password storage on server
2. Password guessing attacks
3. Entropy and password strength
4. Other password security issues
5. Better user authentication?
6. Physical authentication tokens, two-factor authentication

User authentication

- Verifying user identity
- Needed for access control and auditing

access control = authentication + authorization

- User authentication is based on credentials
  – Password, key, smart card etc.

Something you know,
something you have, or
something you are

Username and password

- Password and PIN code are the most common types of authentication credentials
- Password is a shared secret between the user and computer system
  - Limitations arise from the reliance on human memory and input methods, and from the lack of cryptographic computing capability in humans
- What attacks are there against passwords?

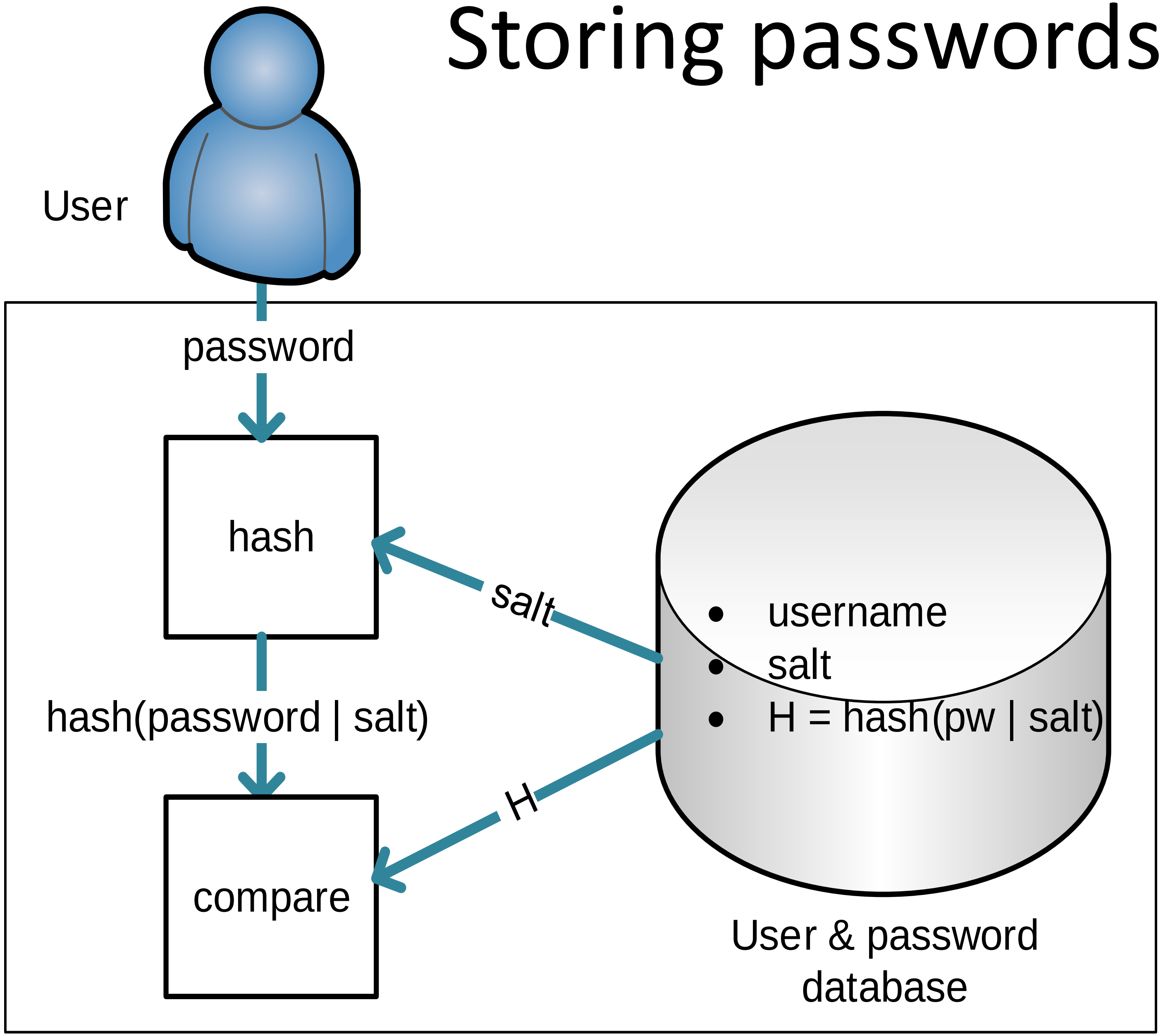
PASSWORD STORAGE ON SERVER

Storing passwords on server

- Assume that your password database is public!
  - Unix `/etc/passwd` is traditionally world readable
  - Attackers often read server files or database tables e.g. with SQL injection

- How to store in a public database?

Storing passwords on server



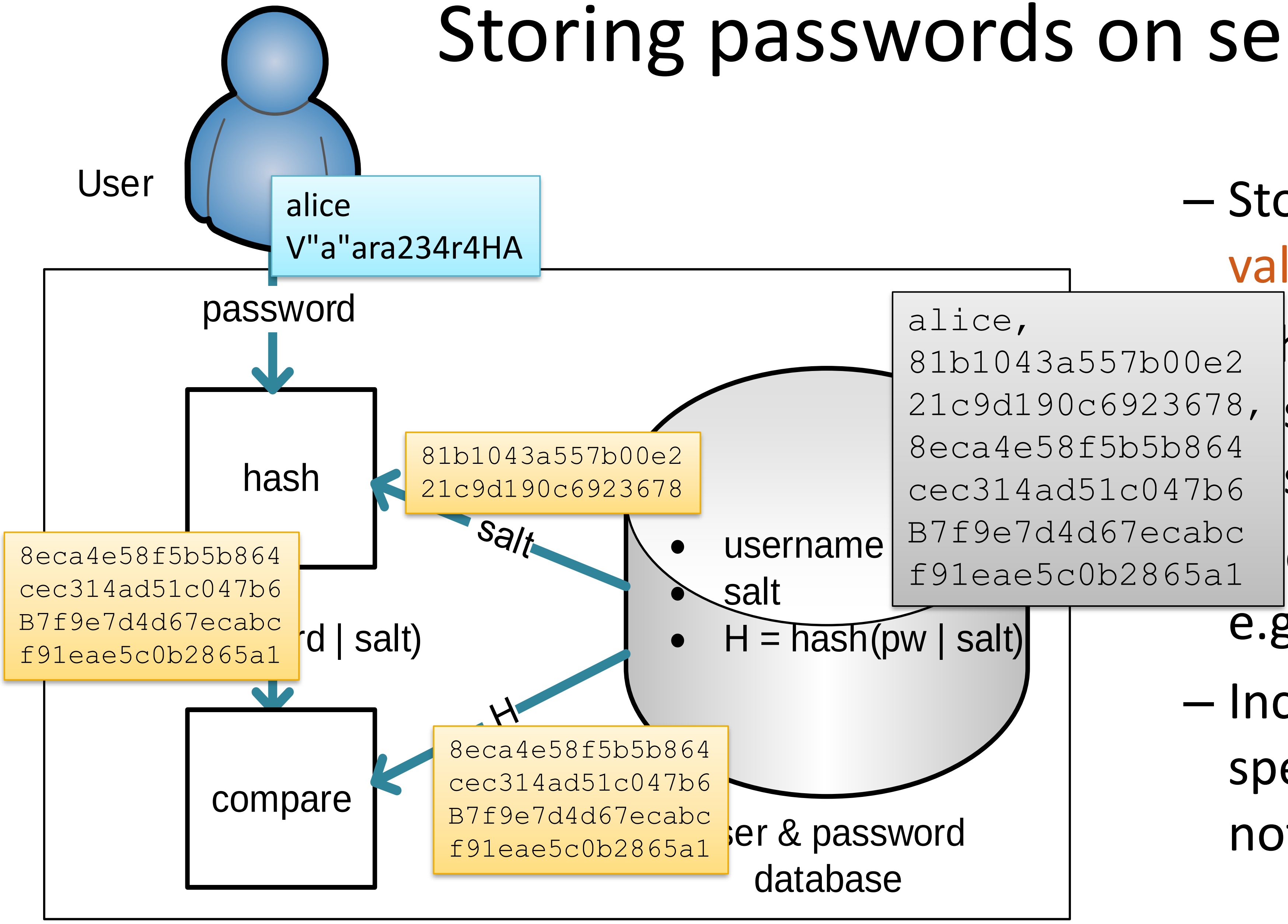
– Store a one-way hash value of the password

– When user enters a password, compute its hash and compare

– Use a slow hash function, e.g. PBKDF2, Argon2

– Include salt: a user-specific random string. not secret

Storing passwords on server



– Store a one-way hash value of the password

hen user enters a ssword, compute its sh and compare

e a slow hash function, e.g. PBKDF2, Argon2

– Include salt: a user-specific random string. not secret

Storing passwords on server

- How to store passwords in a public database?
  Database record:

  username, salt, *slowhash*(password | salt)

  – Store a cryptographic hash i.e. one-way hash value of the password
  – When user enters a password, compute its hash and compare
  – Use a slow hash function to make brute-force cracking slower
  – Include salt: user-specific random string, not secret

One-way function

- Cryptographic hash functions have the one-way property: Easy to compute the hash h(M) for a given message M, but difficult to compute M given h(M)
  - Attacker can only guess M and compare the hashes
- Examples: SHA-256, SHA-3 (old ones: SHA-1, MD5)

Slow hash function

- Standards hash functions are unnecessarily fast!
- Iterative hash:
  - hash(pw|hash(pw|salt)) takes twice as long as hash(pw|salt)
  - Iterate N times (N > 100 000) for desired delay
- Why? Not a significant cost when verifying user login, but increases a brute-force attacker's work by factor N
- Slow functions designed specifically for password hashing: PBKDF2, Argon2

Use these; do not invent your own!

Salt in password hash

- Why salt?

  username, salt, slow_hash(password | salt)

- Salt prevents
  - Simultaneous brute-force cracking of many passwords
  - Pre-computation attacks including rainbow tables
  - Equality comparison between passwords

PBKDF2

- PBKDF2 (P, S, c, dkLen)

  P = password
  S = salt
  c = iteration count
  dkLen = length of the result

  PRF = keyed pseudorandom function

  i.e. keyed hash function

  $F(P, S, c, i) = U_1 \text{ xor } U_2 \text{ xor } \ldots \text{ xor } U_c$

  $U_1 = PRF(P, S \,||\, i)$
  $U_2 = PRF(P, U_1)$
  ...
  $U_c = PRF(P, U_{c-1})$

  Repeat for i=1,2,3... until dkLen output bytes produced

Standard function for slow hashing of passwords

Many iterations to make the computation slower

Used in WPA2-Personal for deriving keys from Wi-Fi passphrase (makes offline cracking more difficult)

https://tools.ietf.org/html/rfc2898

Extra material

Password hashing details

Extra material

- Password-based key derivation function PBKDF2 [PKCS#5,RFC2898]*
  - Good practical function; uses any standard hash function, at least 64-bit salt, any number of iterations
- Argon2 uses a configurable amount of memory and data-dependent memory access patterns
  - harder to crack with GPUs and vector processors

- Unix crypt(3) [Morris and Thompson 1978]*
  - Historical function for hashing passwords stored in /etc/passwd

```
aura:1W90gEpaf4wuk:19057:100:Tuomas Aura:/home/aura:/bin/zsh
```

  - Password = eight 7-bit characters = 56-bit DES key (too short, can be brute-forced)
  - Encrypt a zero block 25 times with modified DES
  - 12-bit salt used to modify DES key schedule (rainbow tables work because the salt is too short)
  - Stored value includes the salt and encryption result
  - Too short salt enables e.g. rainbow table attacks
- Shadow passwords: crypt(3) is replaced by more modern hash functions and the file /etc/shadow is read-protected

PASSWORD GUESSING ATTACKS

Offline cracking

!

- Attacker obtains the password hashes or other data for verifying password guesses, then starts guessing
- Brute-force attacks vs. intelligent dictionary attacks
  - Most password crackers combine both strategies
- Attacker has great advantages:
  - Unlimited number of guesses
  - Can rent elastic computing capacity for quick results
- To resist cracking, passwords must have cryptographic strength (~128 bits of entropy)

Easy to crack some passwords; hard to crack them all. Why?

Online trials – much harder

!

- Online trials: attacker tries to login many times
  - Try PIN codes on a phone or cash machine
  - Guess passwords for a web site
  - Port scan ssh servers and guess root password
- System can limit the number or rate of login attempts
  - Possible in online services, smartcards, phone, Microsoft account
  - Huge improvement in security: success probability ≈ number of allowed guesses / number of possible passwords
  - Denial of service (DoS) is a danger, e.g. bricking a phone; use delay rather than a fixed limit on the number of trials when possible

Cost of offline password cracking

!

- Time to crack a random 10-character (printable ASCII) password from its SHA-256 hash?
  - High-end multi-core CPU on a PC computes up to 500 MH/s
  - High-end graphics card computes up to 7 GH/s, same cost
  - Bitcoin miner computes 15 TH/s
- Always measure cracking cost in money, not in time, because brute-force cracking parallelizes easily and computing capacity can be rented on demand
  - One CPU or GPU day ≈ $1 (cloud CPUs may be cheaper)

Cost of password cracking - continued

- How long does it take / how much does it cost to crack a random 10-character password (printable 8-bit ASCII) from its SHA-256 hash?
- $95^{10} = 2^{65.7} = 6.0 \cdot 10^{19}$ possible passwords. Thus, brute-force cracking takes at most this many trials (50% on average)
- High-end CPU on a PC computes up to 0.5 GH/s (SHA-256)
  - Thus, cracking the password takes $6.0 \cdot 10^{19} / 0.5 \cdot 10^{9} = 1.2 \cdot 10^{11}$ CPU seconds = 1.3M CPU days
  - One CPU day on PC ≈ $1; Thus, cost of cracking the password is about $1.3M
- High-end gaming graphics card computes up to 7 GH/s and costs about the same as PC
  - Thus, cracking the password takes about 90000 GPU days and costs about $90000
- Bitcoin mining rig can compute 15 TH/s (but supports only a specific hash function)
  - Thus, cracking the password takes $6.0 \cdot 10^{19} / 15 \cdot 10^{12} = 4.0$M seconds = 46 days
  - Rig rental online costs $1.50 per day = $69 per password
- Time can be shortened by parallelizing; cost remains the same!
- What is the effect of 1000 hash iterations? Changing password length to 8 or 20 characters?

http://hashcat.net/oclhashcat/
https://www.miningrigrentals.com/rigs/sha256

Cost data
updated 2020

Calculations with powers of 2 and 10

Mental arithmetic for every computer scientist!

- Converting between bases 2 and 10:

  kilo k = $2^{10} \approx 10^3$
  mega M = $2^{20} \approx 10^6$
  giga G = $2^{30} \approx 10^9$
  tera T = $2^{40} \approx 10^{12}$

- Conversion examples:

  Upper and lower bound

  $300M \approx 300 \cdot 10^6 \quad (< 256 \cdot 2^{20} = 2^{28}, \; > 128 \cdot 2^{20} = 2^{27})$
  $2^{34} = 2^4 \cdot 2^{30} = 16G \approx 16 \cdot 10^9$

- Approximate mental arithmetic example:
  - Number of passwords: $95^8 \approx 100^8 = 10^{16}$

    Warning! Potentially big error when approximating the base in exponentiation
  - Hashing speed: 100 MH/s = 10^8 hash/s
  - Cracking time: $10^{16} / 10^8 = 10^8$ CPU seconds
    $= 10^8 / (24 \cdot 60 \cdot 60) = 10^8 / 86400 = 10^8 \approx 10^8 / 10^5 = 1000$ CPU days
  - The exact results with a calculator is 770 CPU days, so we got close
- Convert to base 2 or 10, depending on which is easier

ENTROPY AND PASSWORD STRENGTH

Measuring password strength

- Many possible metrics:
  - Number of possible passwords
  - Entropy = amount of missing information
  - Average/median cost to crack a specific password / any one password
  - Success probability / number of cracked passwords as function of cost
- Metrics are useful for system designers and setting policies
- Measuring strength of user-chosen passwords is impossible

Password entropy

- Entropy = the amount of missing information

  Entropy $H = -\sum_{x \in \text{passwords}} \left(P(x) \cdot \log_2 P(x)\right)$

  $\leq \log_2(\text{number of possible passwords})$

- With even probability distribution:

  $H = \log_2(\text{number of possible passwords})$

  – Example: random 8-character alphanumeric passwords:
  $H = \log_2(62^8) = 8 \cdot \log_2(62) = 47.6$ bits

- One-bit increase in entropy approximately halves the success probability or doubles the cost of guessing attacks (exactly so with even probability distribution)

Sufficient PIN and password entropy

!

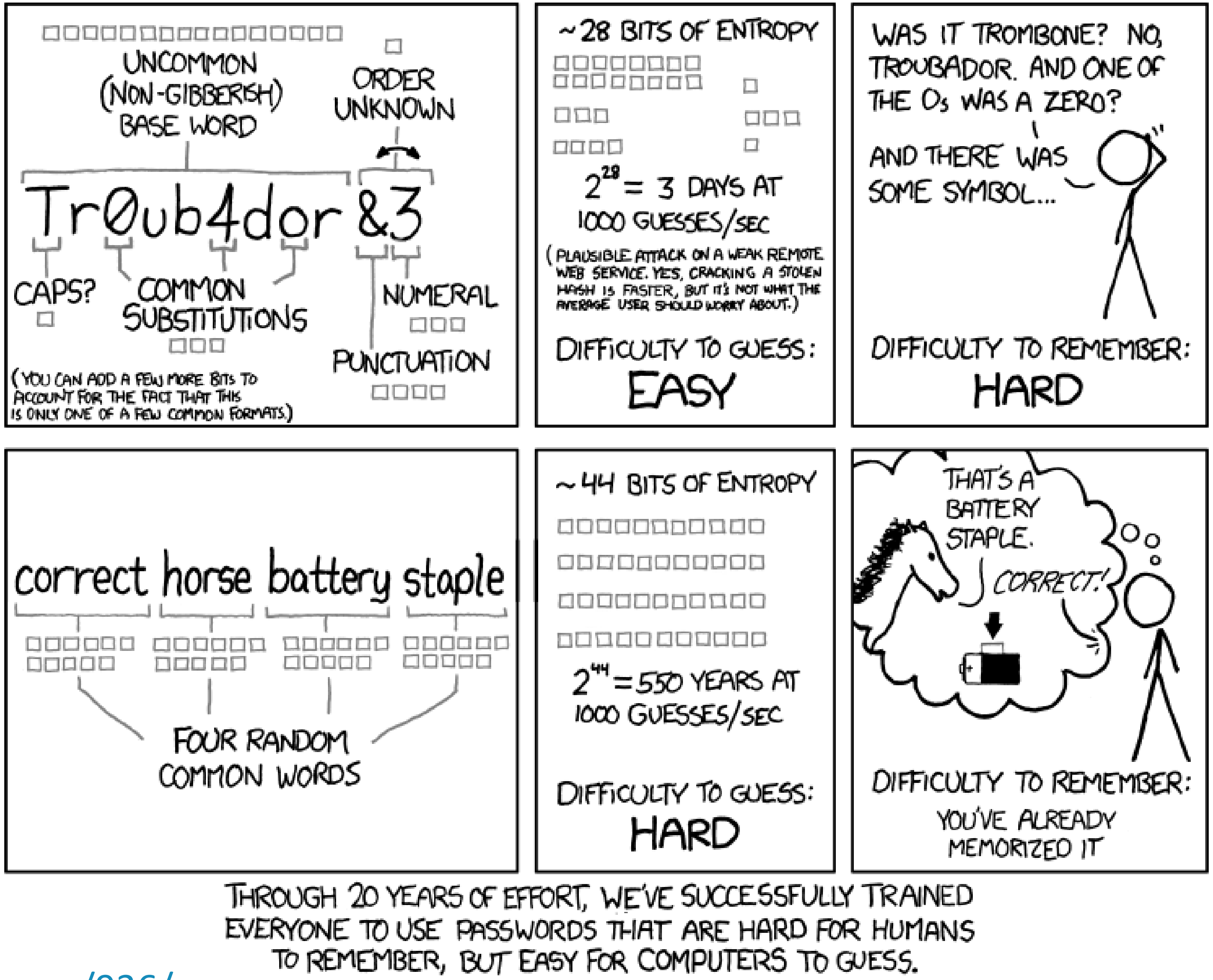
- What is sufficient entropy to resist online guessing?
  1. Determine the maximum number of guesses, e.g. $K = 3$
  2. Decide acceptable success probability, e.g. $P = 10^{-6}$
  3. Required entropy $H = \log_2(K/P)$ = 21.5 bits

Assuming machine-generated passwords with even probability distribution

- What is sufficient entropy to resist offline cracking?
  1. Estimate maximum hash rate, e.g. Bitcoin network $R = 1.2 \cdot 10^{20}$ H/s (SHA-256) in 2020
  2. Decide how long the attack could take, e.g. $T = 1$ year $= 31.5 \cdot 10^{6}$ s
  3. Decide acceptable success probability, e.g. $P = 10^{-6}$
  4. Required entropy $H = \log_2(R \cdot T/P)$ = 66.7+24.9+20 bits = 111.6 bits

     ➔ Human effort can crack 92-bit passwords and threaten 112-bit ones.
  – Traditionally, 128 bits has been considered cryptographically strong.

Human-chosen passwords



https://xkcd.com/936/

PIN entropy examples

Note: Entropy is not a perfect metric for password strength, but it is such a fundamental concept in security - and computer-science in general - that you should have a feel of it.
(Please point out any errors in the examples.)

- PIN entropy examples:
  - Random 4-digit PIN: $H = -\sum_{1 \ldots 10000}(1/10000 \cdot \log_2(1/10000)) = \log_2(10000) = 13.3$ bits
  - PIN with a date (format DDMM): $H = \log_2(365) = 8.5$ bits
  - Assume only 30% of users replace the random PIN with a date:

    $P_{date} = 30\% \cdot 1/365 + 70\% \cdot 1/10000 = 0.00089$, $P_{other} = 70\% \cdot 1/10000 = 0.00007$

    $H = -365 \cdot P_{date} \cdot \log_2(P_{date}) - (10000-365) \cdot P_{other} \cdot \log_2(P_{other}) = 12.6$ bits
- Password entropy examples:
  - Random 18-character (printable ASCII) passwords: $H = \log_2(95^{10}) = 119.3$ bits - Resist offline cracking!
  - Random 10-character (printable ASCII) passwords: $H = \log_2(95^{10}) = 65.7$ bits
  - Random 22-character alphanumeric passwords: $H = \log_2(62^{8}) = 125.0$ bits - Resist offline cracking!
  - Random 8-character alphanumeric passwords: $H = \log_2(62^{8}) = 47.6$ bits
  - Random 8 lower-case characters: $H = \log_2(26^{8}) = 37.6$ bits
  - Random 6 lower-case characters + two digits (e.g. okwrsn91): $H = \log_2(26^{6} \cdot 10^{2}) = 34.8$ bits
  - Random 6-character English word + two digits (e.g. banana28): $H = \log_2(15222 \cdot 10^{2}) = 20.5$ bits

Password entropy examples

Extra material

- Random 8-character (printable ASCII) passwords: $H = \log_2(95^8) = 52.6$ bits
- Random 8-character passwords with exactly two upper case, two lower case, two digits, two special characters:
  - 26 capitals, 26 non-capitals, 10 digits, 33 other
  - Orderings $8!/(2!\cdot 2!\cdot 2!\cdot 2!) = 2520$
  - Different passwords: $26^2\cdot 26^2\cdot 10^2\cdot 33^2\cdot 2520$
  - $H = \log_2(26^2\cdot 26^2\cdot 10^2\cdot 33^2\cdot 2520) = 46.8$ bits
- Random 8-character alphanumeric password with at least one upper case and at least one digit:
  - All 8-character alphanumeric passwords: 62^8
  - Those with no upper case: $(62-26)^8 = 36^8$
  - Those with no digit: $(62-10)^8 = 52^8$
  - Those with with no upper case and no digit: $(62-26-10)^8 = 26^8$
  - Allowed passwords: $62^8 – (36^8 + 52^8) + 26^8$ (inclusion exclusion principle)
  - $H = \log_2(62^8 – (36^8 + 52^8) + 26^8) = 47.2$ bits
- Random alphanumeric passwords with one special character:
  - 7-character alphanumeric passwords: 62^7
  - 33 special characters to choose from, 8 possible locations to insert it
  - $H = \log_2(62^7 \cdot 33 \cdot 8) = 49.7$ bits
- So what? The rules have different effect on user-chosen and random passwords

Password entropy and humans

- Human-selected passwords have less entropy than random ones because some are chosen more often than others
  - Should banks allow the customer to choose the PIN?
  - Do password quality guidelines and checks increase entropy?

- Passwords rely on human memory → password entropy cannot grow over time → human memory cannot compete with brute-force cracking by computers

Human-chosen 4-digit PINs

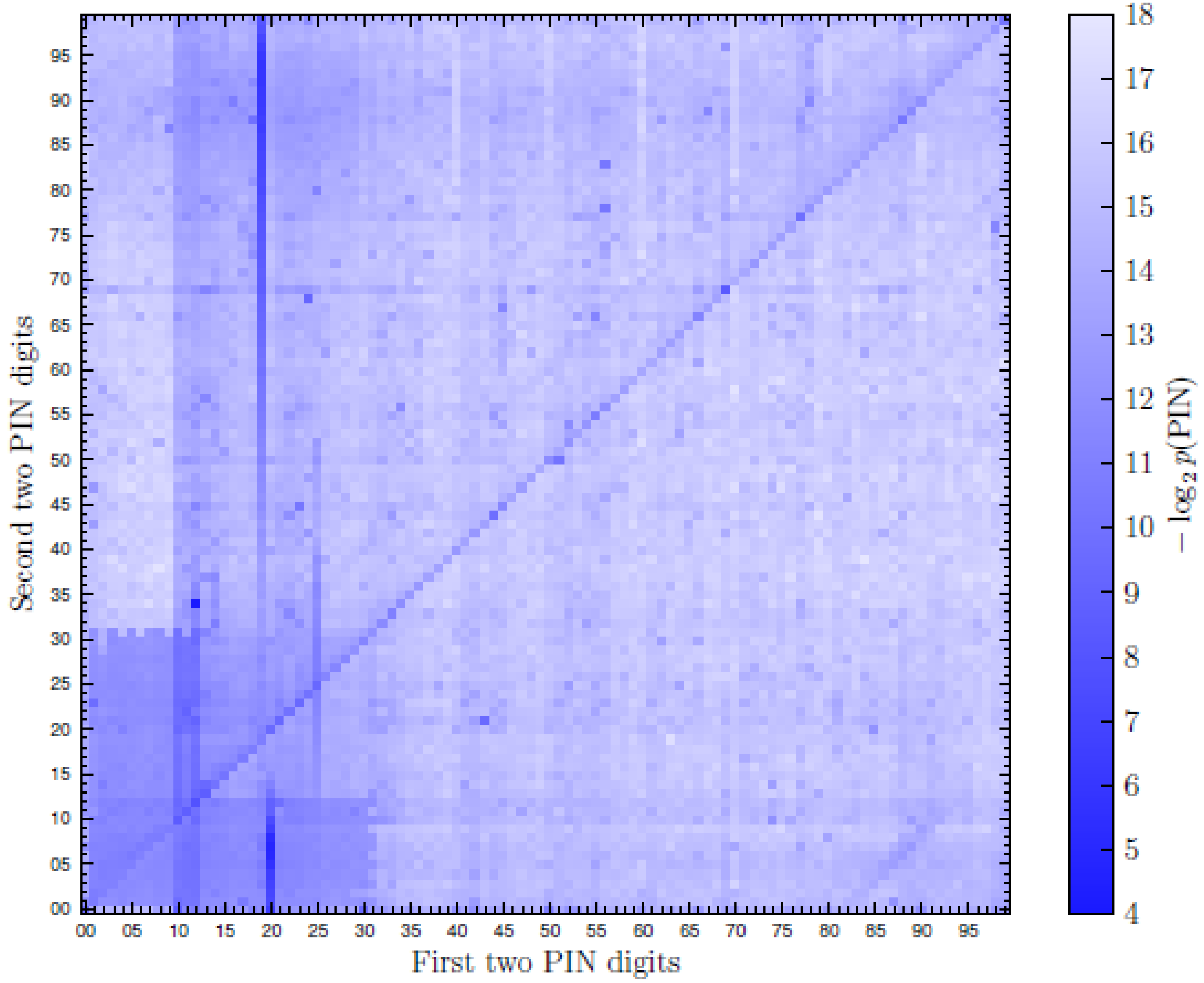


Figure 4.2: The distribution of 4-digit sequences within RockYou passwords (RockYou-4). Each cell shows the frequency of an individual PIN.

Bonneau, Joseph: [Guessing Human-Chosen Secrets](), PhD Thesis, University of Cambridge, 2012.

NIST Password Guidelines

Extra material

- NIST Special Publication 800-63 Digital Identity Guidelines: B: Authentication and Lifecycle Management (June 2017)
  - "Verifiers SHALL require subscriber-chosen memorized secrets to be at least 8 characters in length. Verifiers SHOULD permit subscriber-chosen memorized secrets at least 64 characters in length."
  - "When processing requests to establish and change memorized secrets, verifiers SHALL compare the prospective secrets against a list that contains values known to be commonly-used, expected, or compromised."
  - "Verifiers SHOULD NOT impose other composition rules (e.g., requiring mixtures of different character types or prohibiting consecutively repeated characters) for memorized secrets."
  - "Verifiers SHOULD NOT require memorized secrets to be changed arbitrarily (e.g., periodically). However, verifiers SHALL force a change if there is evidence of compromise of the authenticator."
  - "In order to assist the claimant in successfully entering a memorized secret, the verifier SHOULD offer an option to display the secret — rather than a series of dots or asterisks — until it is entered."

https://pages.nist.gov/800-63-3/sp800-63b.html

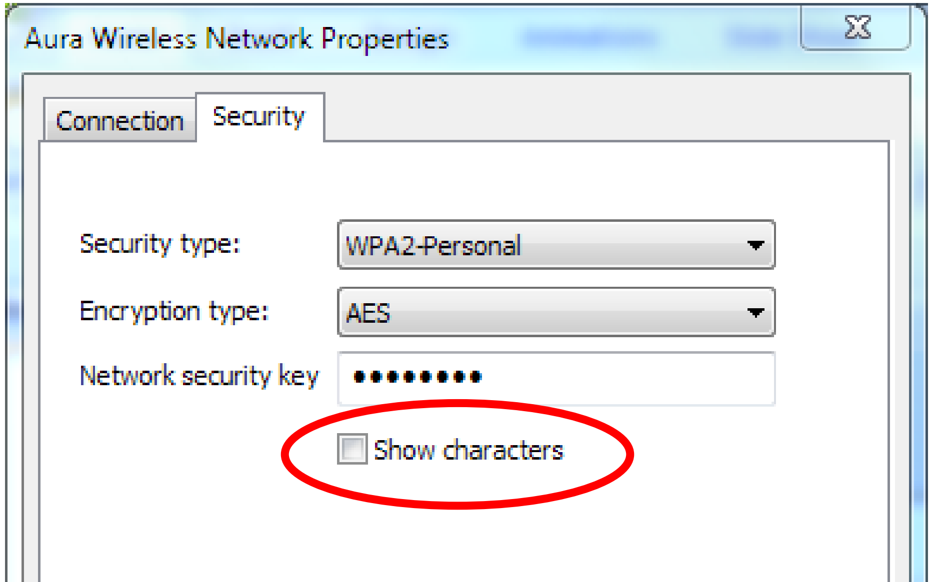
OTHER PASSWORD SECURITY ISSUES

Sniffing and key loggers

- Password sniffing on the local network is prevented by cryptographic authentication (SSH, HTTPS, MS-CHAPv2,…)
- Key logger: software or hardware that stores all keystrokes typed on the computer
  - Problem in public-access computers
  - Malware can sniff passwords on any infected computer

Shoulder surfing

- Keyboards and screens are highly visible
  → Others may see what you are typing
- Password and PIN inputs are usually masked   *******
  – Does masking always make sense?
    Increasingly, option to show the characters if in a safe place
- Remember also hidden cameras and telephoto lenses



Spoofing and phishing attacks

- For console login, attacker tries to spoof the login dialog; how do you know when it is safe to type in the password?
- For web login, attacker tries to spoof the login page for a web site
- For mobile apps, one app tries to spoof the login interface of another (e.g. online bank)

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

Trusted path

!

- What if attacker spoofs the login dialog?
- Trusted path is any mechanism that ensures direct and secure communication between user and a trusted part of the system
  – Crtl+Alt+Del in Windows (secure attention key / sequence)
  – Reset button in all kinds of devices
  – Web browser address bar
- With malware, virtualization and full-screen apps, it is increasingly hard to know what is real

Password reuse

- Same or related passwords on multiple accounts
  → compromise of one system or account leads to compromise of the user's other accounts

Solutions:

- Password manager that stores and generates random passwords

  User solution

- Single sign-on (SSO)
  - Shibboleth SSO to university web pages
  - Microsoft AD, IBM Tivoli Access Manager, etc.
  - Facebook, Google, etc. login on many websites

  Organization solution

Password recovery

- Humans are prone to forget things → need a process for recovering from password loss

> Failure-recovery often enables new attacks!
> This applies to security mechanisms in general

- Some password recovery methods:
  - Physical visit to helpdesk
  - Security question or memorable secret, e.g. mother's maiden name, birthdate
  - Email or text message with authorization code or link
  - Paper notebook, sticky note under the keyboard
  - USB memory stick with a password recovery file
  - Print recovery code as QR code

What are the advantages and disadvantages?

Other threats

- No system is perfectly secure: system designers have a specific threat model in mind, but the attacker can break these rules

"The attacker does not agree with the threat model."
(Bruce Christianson)

- Some other attacks against PINs and passwords:
  - Phishing emails and social engineering
  - User mistakes: using the password on wrong site
  - Side channels: heat camera, acoustic emanations

BETTER USER AUTHENTICATION?

One-time passwords

- Use each password only once. Protects against password sniffers and key loggers
  - Random one-time passwords
  - Lamport hash chain
  - Unix S/KEY or OTP

    1: HOLM BONG VARY TIP JUT ROSY

    2: LAIR MEMO BERG DARN ROWE RIG

    3: FLEA BOP HAUL CLAD DARK ITS

    4: MITT HUM FADE CREW SLOG HAST
  - Many commercial products such as RSA SecurID
  - Code apps and devices for Finnish banks

- Which attacks do one-time passwords prevent and which not?

One-time password implementation

Extra material

- One-time passwords can be random strings, but most practical implementations use pseudorandom values and cryptographic (one-way) hash functions
- Hash-based one-time passwords HOTP [RFC4226], OPTW

  $HOTP(K,i) = HMAC\text{-}SHA\text{-}1(K,i) \bmod 10^D$

  - Produces one-time PIN codes of D decimal digits from master secret K and counter i
  - Server and user's authentication device only remember K and i
- Time-based one-time passwords: instead of counter, use the current time
  - Many commercial products such as RSA SecurID
- Lamport hash chain:

  $H_1 = hash(secret\ seed)$; $H_{i+1} = hash(H_i)$

  - Convenient storage: server stores initially H_{100} and asks user to enter H_{99}. Next, it stores H_{99} and ask for H_{98}, and so on
  - Unix S/KEY [RFC1760] and OTP [RFC1938]

    1: HOLM BONG VARY TIP JUT ROSY
    2: LAIR MEMO BERG DARN ROWE RIG
    3: FLEA BOP HAUL CLAD DARK ITS
    4: MITT HUM FADE CREW SLOG HAST

  - Usability problem: hashes are long random numbers

Weak and low-entropy credentials

- PIN, graphical passwords, face recognition, fingerprints have recently replaced strong passwords. Why would that be ok?
- Only for physical access to device, not for remote access to the device or to related online services
- For access to online services, physical possession of the user device is considered one authentication factor, PIN the other
- Main threat now is lost and stolen mobile devices
  - Attacker does not know the user
  - Hardware feature to lock the device after a few trials

Online accounts

- User authentication delegated to online server
  - Device cryptographically locked, and server releases keys after successful authentication
  - Online server can limit the number of password guesses and implement risk-based additional authentication, e.g. 2FA
  - Device must not store the password database and must be online
- But are the password hashes cached locally?
  - e.g. Windows login with Microsoft account caches authentication information locally, unless disables by domain administator
- Authentication delegated to a secure hardware module can have similar benefits

Password manager

- Password manager for web service passwords
  - Generates long, random, services-specific passwords
  - Protects them all with a single master password
- e.g. LastPass, Dashlane, F-Secure Key
  - Can also synchronize the database between the user's devices

→ Solves the issues with human memory, weak passwords, and password reuse

→ Creates a new single point of failure

PHYSICAL AUTHENTICATION TOKENS, TWO-FACTOR AUTHENTICATION

Physical security tokens

- Smart card is a typical physical security token
  - Stores cryptographic keys to prove its identity
  - Tamperproof: secret keys will stay inside
- Used for door keys, computer login, bank cards
- Other security tokens: smart button, USB dongle, trusted chip in mobile phone

Two-factor authentication (2FA)

- Two-factor authentication =
  require both a physical token and a PIN or password
  - Attacker needs to both steal the physical device and learn the PIN
    → clear qualitative increase in security

- Context-aware or risk-based authentication:
  - Require additional authentication only when the user is suspicious or requested action requires stronger security
  - Online services can do this intelligently to avoid annoying the user

Issues with physical tokens

- Physical tokens require distribution
- Computers (or doors etc.) must have readers
- It is not easy to integrate cryptographic tokens to all systems
  - Application with cached credentials on the client or on a proxy server
  - Systems that need to start automatically after unexpected reboot
- Process needed for recovering from the loss of tokens
- Are the two factors really independent?
  - smart card + PIN
  - fingerprint swipe and bank code app on your phone

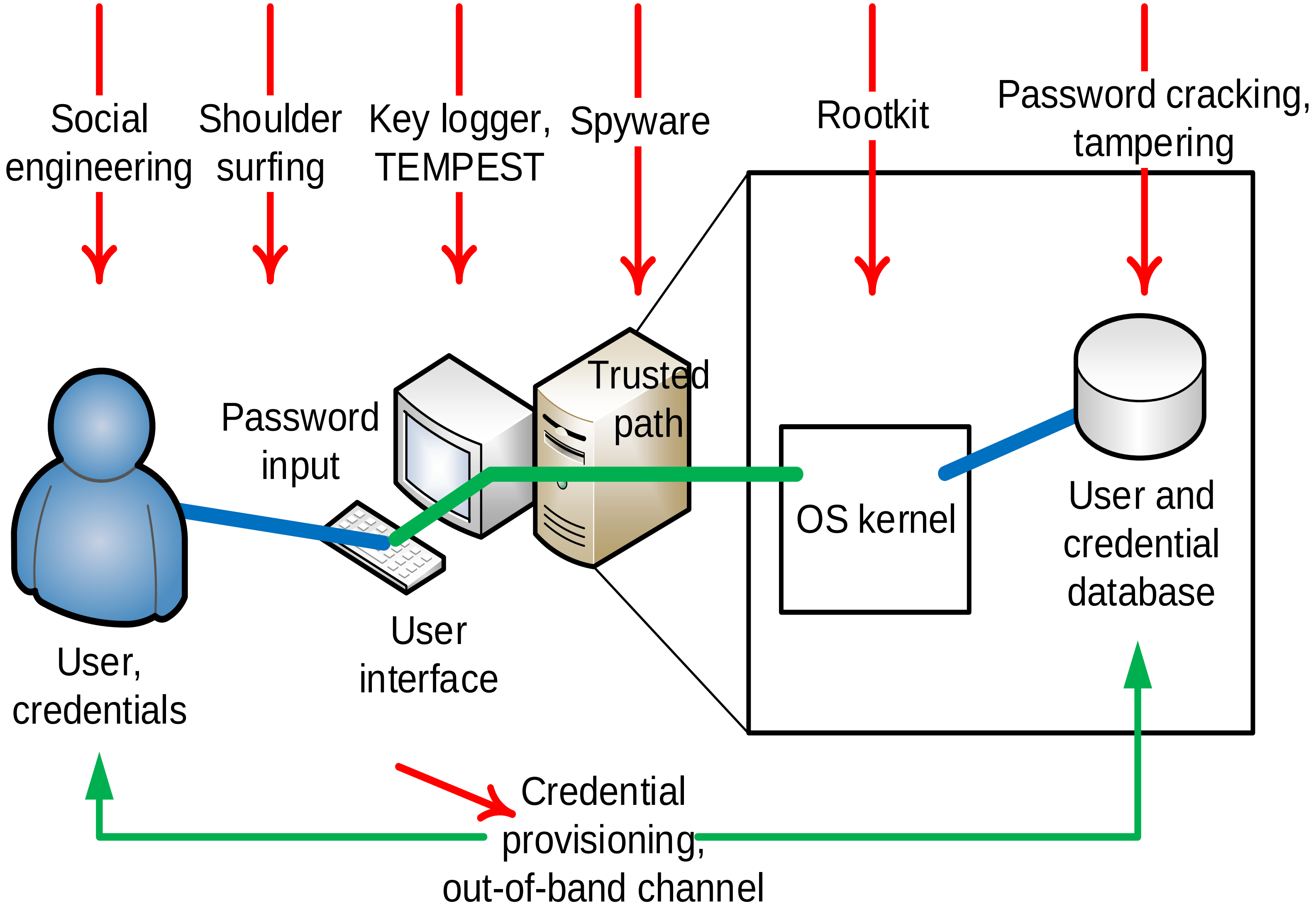
Authentication with mobile phone

- Two-channel authentication used by major online services:
  - Confirmation via telephone: callback, SMS
  - Confirmation via dedicated mobile app
  - Sending a second secret to a known address: SMS, email, post
  - Alerting user to potentially malicious events

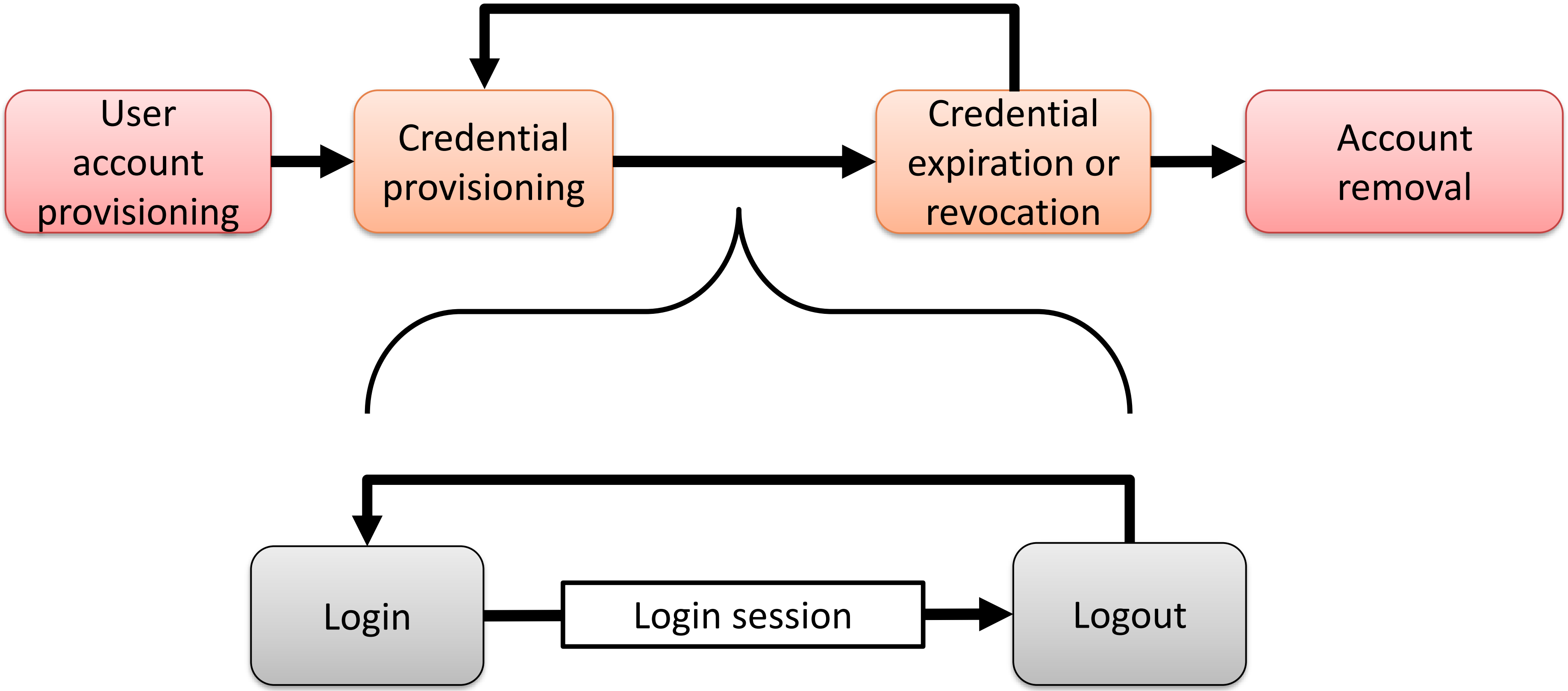
- Secure element in mobile phones can be used as a login token
  - The SIM is a smart card and could also act as the authentication token

SUMMARY

User authentication summary



Credential lifecycle



[source: Sanna Suoranta]

List of key concepts

- Entity authentication, user authentication, login, logout, session
- Credential, shared secret, username, password
- Issuing or enrollment, out-of-band channel
- Sniffing, spoofing, malware, trusted path
- Failure recovery
- Brute-force cracking, dictionary attacks, online vs. offline attacks, entropy, probability, security metrics
- Cryptographic hash function, one-way function, salt, PBKDF2, Argon2, one-time password, Lamport hash chain
- Smart card, two-factor authentication, second channel, context-aware or risk-based authentication
- Account and credential provisioning, revocation

Reading material

- Dieter Gollmann: Computer Security, 2nd ed., chapter 3; 3rd ed. chapter 4
- Matt Bishop: Introduction to computer security, chapter 11
- Ross Anderson: Security Engineering, 2nd ed., chapters 2, 15
- Stallings, Brown: Computer Security: Principles and Practice, 3rd/4th ed., chapter 3
- Bonneau, Joseph: Guessing Human-Chosen Secrets, PhD Thesis, University of Cambridge, 2012.

Exercises

- Why do you need both the username and password? Would not just one secret identifier (password) be sufficient for logging in?
- What effect do strict guidelines for password format (e.g. 8 characters, at least 2 capitals, at least 2 digits, at least 1 special symbol) have on the password entropy?
- What is the probability of guessing the code for a phone that allows 3 attempts to guess a 4-digit PIN code, then 10 attempts to guess an 8-digit PUK code?
- In what respects is PBKDF2 better for password hashing than the old crypt(3)? How does Argon2 improve on PBKDF2?
- How many hash values can a brute-force attacker test in a second with the latest GPUs? Check also the Bitcoin mining speeds on GPUs.
- How do mandatory periodic password changes increase security? What is the optimal interval for password expiry?
- How to limit the number of login attempts without creating a DoS vulnerability? Consider both an online service and a device like phone.
- Learn about graphical passwords and compare their entropy to passwords and PIN codes of various lengths.
- Learn about HTTP Digest Authentication [RFC2617] and MS-Chap-V2 [RFC2759]. Explain how to perform an offline password guessing attack after sniffing a login.
- Which attacks do one-time passwords / password managers / physical tokens / 2FA prevent, and which do they not?
- Could authentication be based on *who you know* (or who knows you), or *where you are*?