Authentication: something you know

Daniel Bosk

Department of Information and Communication Systems,
Mid Sweden University, Sundsvall

5th November 2020

1 Something you know

- 'Proof of knowledge'
- Guessing secrets
- Online or offline?
- Storing secrets

Idea: Something you know

- We have a prover and a verifier.
- Prover must convince verifier he knows some secret.

Idea: Password

- Prover and verifier shares a secret value.
- Prover tells verifier the value to convince the verifier.

Remark

- If the adversary learns the secret, he can convince the verifier he is the prover.

Example

- Adversary might 'overhear the conversation'.
- Adversary might 'trick' the prover to reveal the secret.
- Adversary might guess the secret.

Idea

- The secret x is chosen from a probability distribution.
- The probability of guessing correctly is $\Pr[X = x]$.

Remark

- The question is: what is the probability distribution?

Example (Cryptographic keys)

- The distribution is *very* close to the uniform distribution.
- *I.e.* $\Pr[X = x] = 1/n$, where X can take n possible values.
- In crypto, normally $n = 2^{128}$.

Example (Password)

- Distribution of passwords is affected by so many factors.
- The individual, situation, password policies, *etc.*

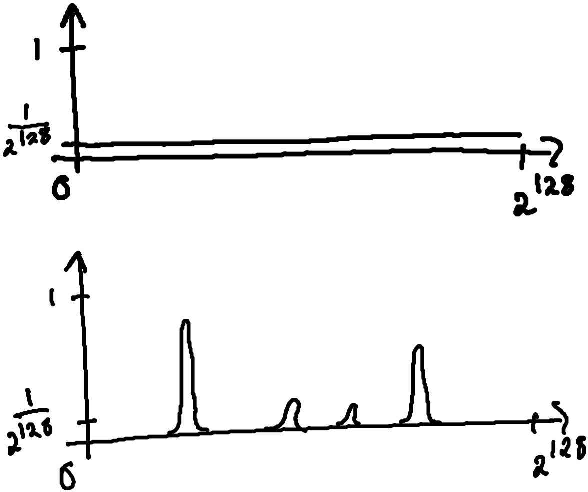
Example (Cryptographic keys)

- The distribution is *very* close to the uniform distribution.
- *I.e.* $\Pr[X = x] = 1/n$, where X can take n possible values.
- In crypto, normally $n = 2^{128}$.

Example (Password)

- Distribution of passwords is affected by so many factors.
- The individual, situation, password policies, *etc.*

Guessing secrets



Idea: Guessing passwords

- Find a way to approximate the distribution.

Example (Basic guessing)

- Using dictionaries of words.
- Adapt guesses to password policy, if known.
- . . .

Example (Improved guessing)

- Use leaked passwords as guesses.
- Take grammar into account, depending on the password type [Bon12; BS12].

Example (Machine learning)

- Use machine learning [Rip; Cas+17; Wei+09].
- Train algorithm on leaked password databases.
- Generate list of password-looking guesses.

Example (Improved guessing)

- Use leaked passwords as guesses.
- Take grammar into account, depending on the password type [Bon12; BS12].

Example (Machine learning)

- Use machine learning [Rip; Cas+17; Wei+09].
- Train algorithm on leaked password databases.
- Generate list of password-looking guesses.

Remark

- There is a PhD thesis on the topic of guessing passwords: **GuessingHumanChosenSecrets**.
- There is even a conference dedicated to passwords: PasswordsCon.

Remark

- This is relevant when the user has chosen a password.
- In many situations it's not.

Example

- There are many devices with default passwords.
- *E.g.* home routers, ...

Remark

- This is relevant when the user has chosen a password.
- In many situations it's not.

Example

- There are many devices with default passwords.
- *E.g.* home routers, ...

Example (Mirai botnet [Her16])

- Botnet infecting primarily surveillance cameras and home routers.
- Attempts default passwords and other vulnerabilities.
- Managed the largest distributed denial-of-service (DDoS) attack hitherto.

Remark

- These default passwords have very high probability.

Idea: Autogenerate passwords

- Generate passwords for users.
- This yields a uniform distribution.

Remark: Usability

- This will likely reduce security by use of post-it notes.
- Not a problem for a home router.
- Otherwise: will require password managers.

Idea: Password policy

- Introduce rules to affect how users choose passwords.
- We require upper, lower case, numbers, special characters.
- Then passwords will be more uniform-looking.

Remark: Usability

- This has been proven a bad idea.
- Research has estimated the distribution under various policies [Kom+11].
- Better to only require length.

Idea: Password ageing

- Let passwords age and expire.
- Then users change them frequently.
- If it takes six months to guess and we change every three . . .

Remark: Usability

- This has been proven a bad idea.
- Annoying with too short intervals.
- Will reduce security once users introduce systems to remember their last changed password.

- Grassi et al. [Gra+19] summarizes the current recommendations.
- At least 10 characters.
- Force renewal only after security breach.

Definition (Online)

- The adversary must interact with the system for each guess.

Example (Online)

- Guessing the password of a Google account.
- Must submit each guess to Google.

Definition (Online)

- The adversary must interact with the system for each guess.

Example (Online)

- Guessing the password of a Google account.
- Must submit each guess to Google.

Definition (Offline)

- The adversary can verify the guess himself.

Example (Offline)

- Guessing the password of an encrypted file.
- For each guess, try to decrypt.

Definition (Offline)

- The adversary can verify the guess himself.

Example (Offline)

- Guessing the password of an encrypted file.
- For each guess, try to decrypt.

Solution (Rate limiting)

- *For online guessing, rate limit the attempts.*
- *This makes guessing too slow.*

Remark

- This works for targeted attacks.
- Introduces possibility for denial-of-service.

Solution (Rate limiting)

- *For online guessing, rate limit the attempts.*
- *This makes guessing too slow.*

Remark

- This works for targeted attacks.
- Introduces possibility for denial-of-service.

Exercise

- How will rate limiting affect these?

```
for u in users:
  for p in passwds:
    try_login(u, p)
```

```
for p in passwds:
  for u in users:
    try_login(u, p)
```

Remark

- Maybe the adversary doesn't care about which user.
- If a password is common, then it's likely that *some* user chose it.
- And if the adversary tries one password for each user, that might not trigger the rate limiting.

Remark: Offline

- Consider data which is encrypted with a password.
- You cannot change a password for data that is already stolen.
- You cannot limit the number of attempts either.
- You can just control the guessability of the password.

Remark

- The user can store the secret in its mind.
- This is assumed inaccessible (for now).

Question

- The verifier is a machine.
- The verifier must verify what the prover says.
- This means that the verifier must have some data to check against.
- How should this be stored?

Remark

- The user can store the secret in its mind.
- This is assumed inaccessible (for now).

Question

- The verifier is a machine.
- The verifier must verify what the prover says.
- This means that the verifier must have some data to check against.
- How should this be stored?

Remark

- Our concern is that someone can read this data.
- This helps better approximate the distribution.
- Password reuse for other services?

Solution (Passwords)

- *We want to compare user-entered and stored password.*
- *We do an irreversible one-way transformation on both.*
- *Then they are still comparable.*
- *The preimage cannot be gained from storage.*

Example

- Cryptographic hash function $h\colon \{0,1\}^* \to \{0,1\}^n$.
- On registration, store $h(p)$.
- User authenticates with p', check if $h(p') \stackrel{?}{=} h(p)$ equals what we stored.

Solution (Passwords)

- *We want to compare user-entered and stored password.*
- *We do an irreversible one-way transformation on both.*
- *Then they are still comparable.*
- *The preimage cannot be gained from storage.*

Example

- Cryptographic hash function $h\colon \{0,1\}^* \to \{0,1\}^n$.
- On registration, store $h(p)$.
- User authenticates with p', check if $h(p') \stackrel{?}{=} h(p)$ equals what we stored.

Remark

- Consider guessing again.
- The used password space is small.
- We only need to evaluate a subset: $h\colon \{0,1\}^m \to \{0,1\}^n$.
- With faster computers we can guess a lot.

Solution

- *Choose h to be slow to compute.*
- E.g. *iterate it over itself 10 000 times ($h^{10000}(p)$).*
- *This will slow down guessing attacks.*

Remark

- Consider guessing again.
- The used password space is small.
- We only need to evaluate a subset: $h\colon \{0,1\}^m \to \{0,1\}^n$.
- With faster computers we can guess a lot.

Solution

- *Choose h to be slow to compute.*
- E.g. *iterate it over itself 10 000 times ($h^{10000}(p)$).*
- *This will slow down guessing attacks.*

Remark

- A list of password hashes reveals if two users have the same password.
- Can guess the password for all users at once:
  1. Make a guess, compute the hash.
  2. Check if it matches *any* user's password.

Solution

- *Add a salt: a small random value (e.g. 128 bits) unique for each user.*
- *Salt* $s \xleftarrow{\$} \{0,1\}^{128}$, *change hash to* $h(s, p)$.
- *Now all hashes will be unique.*

Remark

- A list of password hashes reveals if two users have the same password.
- Can guess the password for all users at once:
  1. Make a guess, compute the hash.
  2. Check if it matches *any* user's password.

Solution

- *Add a salt: a small random value (e.g. 128 bits) unique for each user.*
- *Salt* $s \xleftarrow{\cent} \{0,1\}^{128}$, *change hash to* $h(s, p)$.
- *Now all hashes will be unique.*

Remark

- The salt is not a secret, it just adds uniqueness.
- It can be stored in plain text along with the password hash.

Example

- There are many libraries.
- bcrypt [PM99] implements all this functionality.
- Argon2 is another, more recent technique.
- They should also be available in most languages and libraries.

[Bon12] Joseph Bonneau. 'Guessing human-chosen secrets'. PhD thesis. University of Cambridge, May 2012. URL: http://www.cl.cam.ac.uk/~jcb82/doc/2012-jbonneau-phd_thesis.pdf.

[BS12] Joseph Bonneau and Ekaterina Shutova. 'Linguistic properties of multi-word passwords'. In: *USEC*. 2012. URL: http://www.cl.cam.ac.uk/~jcb82/doc/BS12-USEC-passphrase_linguistics.pdf.

[Cas+17] Claude Castelluccia, Abdelberi Chaabane, Markus Dürmuth and Daniele Perito. *When Privacy meets Security: Leveraging personal information for password cracking*. 15th Feb. 2017. arXiv: 1304.6584 [cs.CR].

[Gra+19] Paul A. Grassi, James L. Fenton, Elaine M. Newton, Ray A. Perlner, Andrew R. Regenscheid, William E. Burr, Justin P. Richer, Naomi B. Lefkovitz, Jamie M. Danker, Yee-Yin Choong, Kristen K. Greene and Mary F. Theofanos. *Digital Identity Guidelines: Authentication and Lifecycle Management*. NIST Special Publication 800-63B. Mar. 2019. DOI: 10.6028/NIST.SP.800-63c. (Visited on 07/03/2019).

[Her16] Ben Herzberg. *Breaking Down Mirai: An IoT DDoS Botnet Analysis*. Oct. 2016. URL: https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html (visited on 18/02/2017).

[Kom+11] Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L. Mazurek, Lujo Bauer, Christin Nicolas, Lorrie Faith Cranor and Serge Egelman. 'Of passwords and people: Measuring the effect of password-composition policies'. In: *CHI*. 2011. URL: http://cups.cs.cmu.edu/rshay/pubs/passwords_and_people2011.pdf.

[PM99] Niels Provos and David Mazieres. 'A Future-Adaptable Password Scheme.'. In: *USENIX Annual Technical Conference, FREENIX Track*. 1999, pp. 81–91.

[Rip] John the Ripper community. *John the Ripper bleeding jumbo*. URL: `https://github.com/magnumripper/JohnTheRipper`.

[Wei+09] Matt Weir, Sudhir Aggarwal, Breno De Medeiros and Bill Glodek. 'Password cracking using probabilistic context-free grammars'. In: *Security and Privacy, 2009 30th IEEE Symposium on*. IEEE. 2009, pp. 391–405.