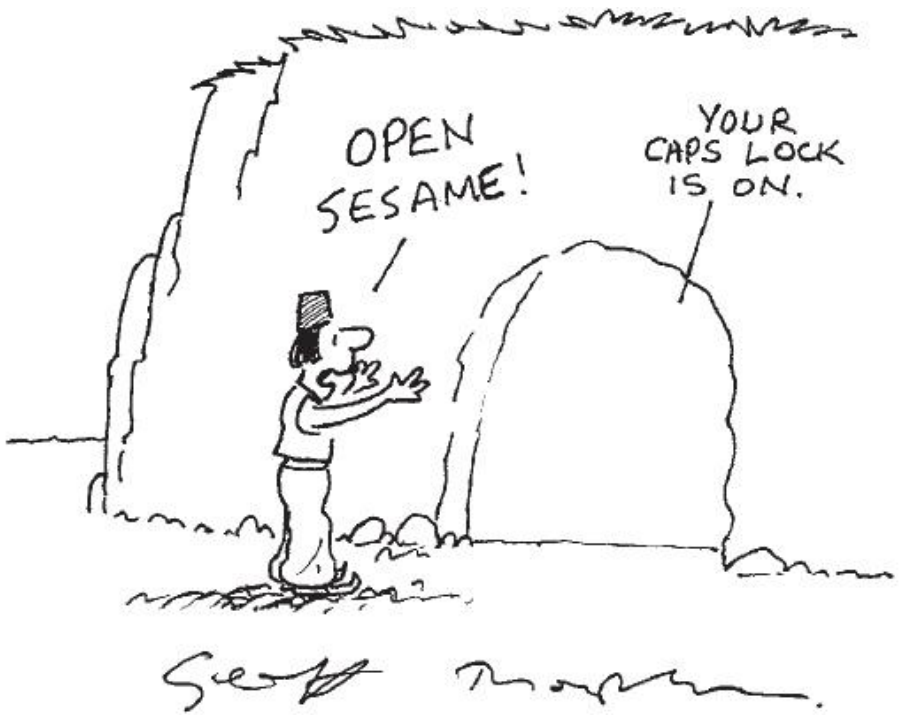


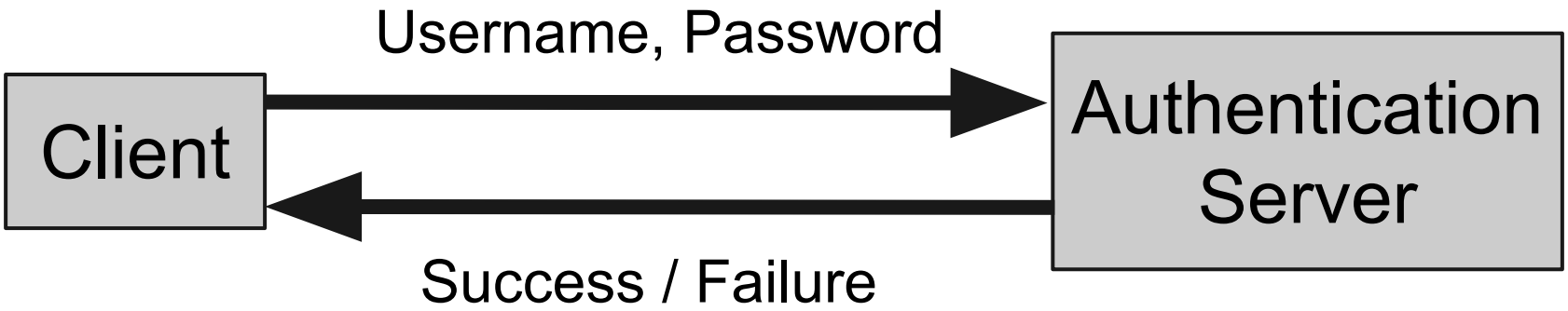
Passwords

https://shop.spectator.co.uk/wp-content/uploads/2015/03/open-sesame.jpg

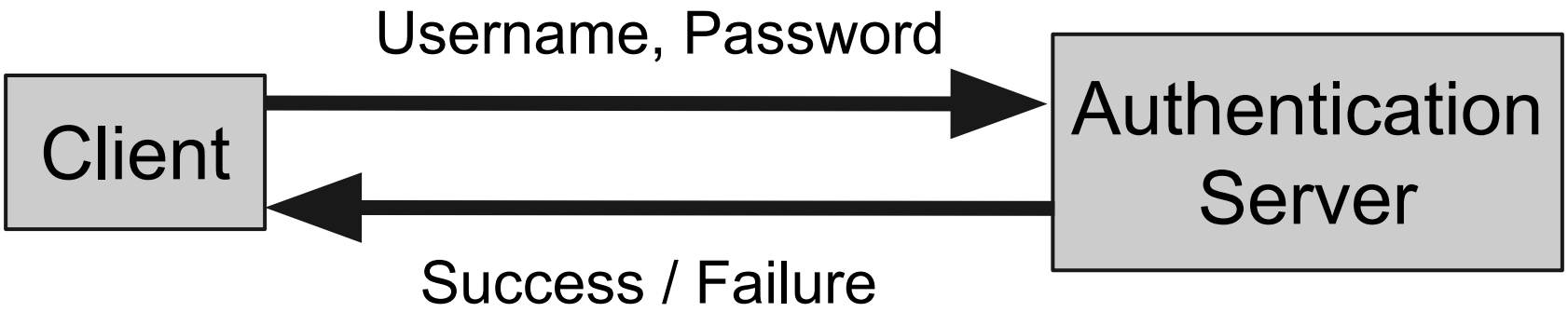
What are passwords?

- Secret token used to prove that you're you
- Key property: only need to be able to verify that it's correct

Password Verification

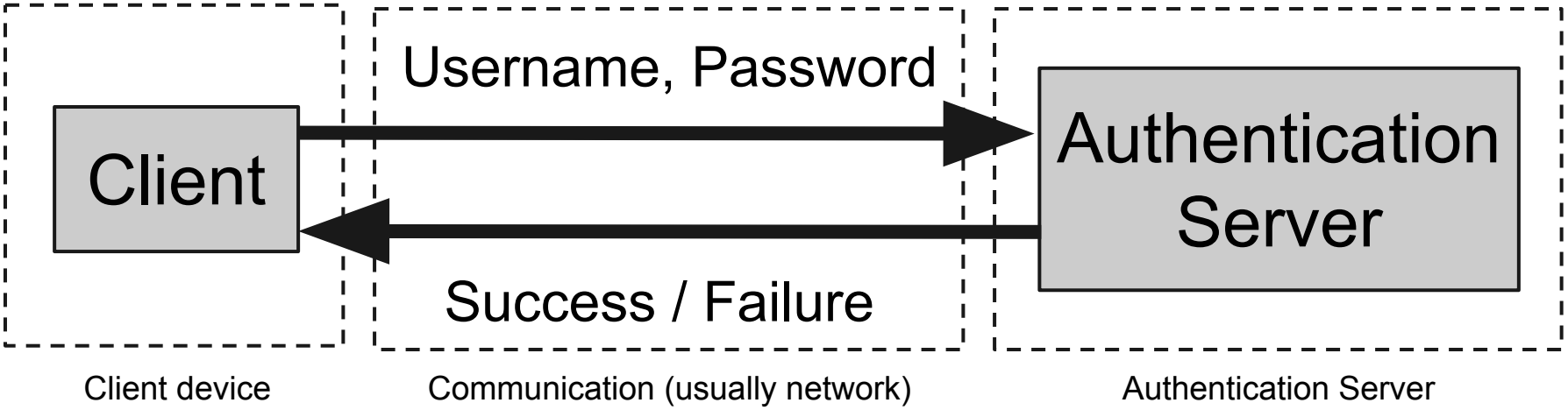


Password Verification

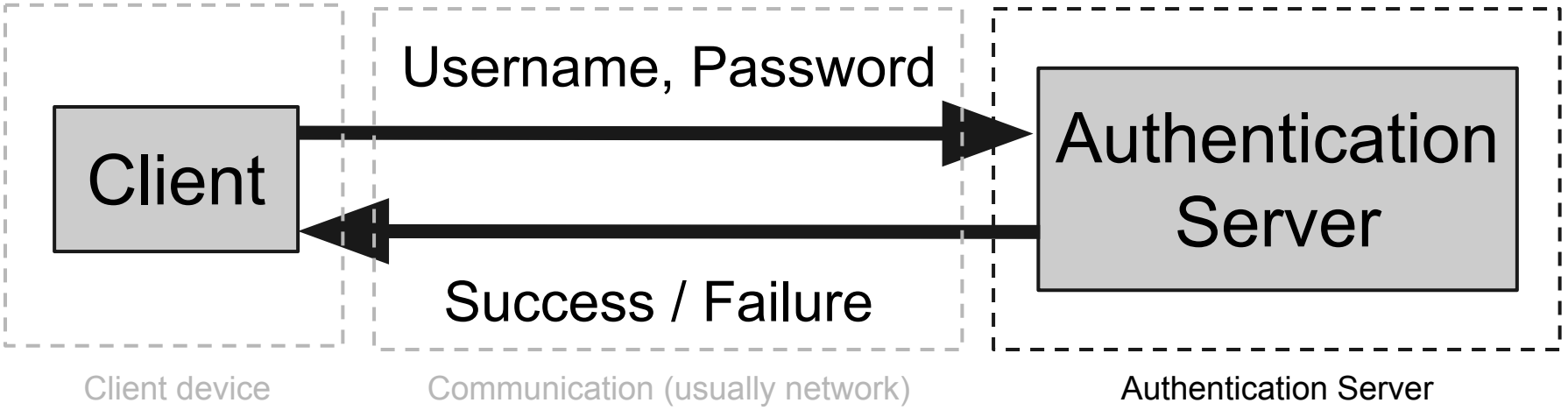


Where could we attack?

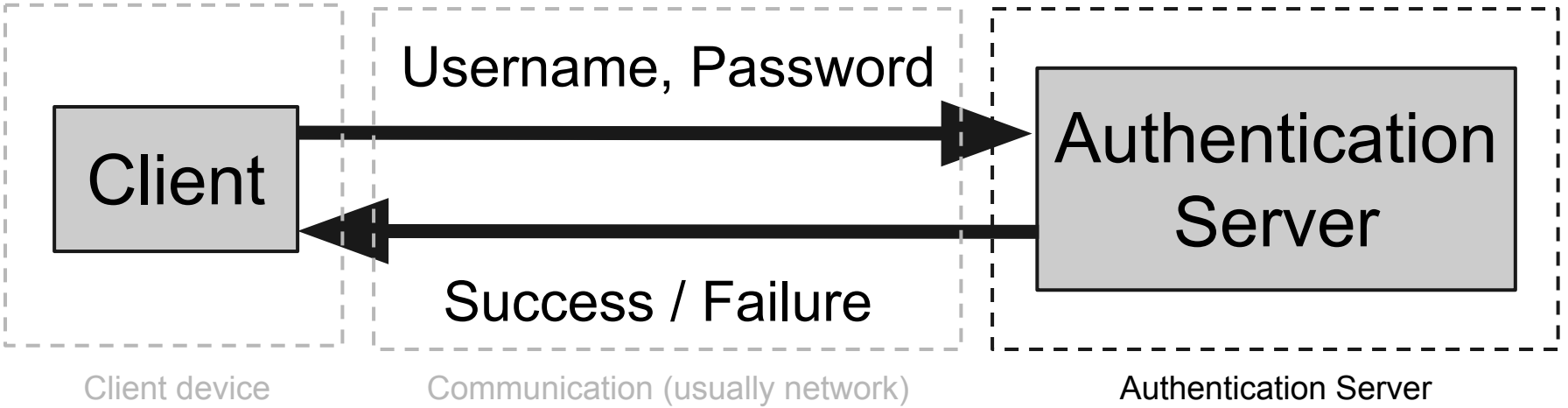
Password Verification



Password Verification



Password Verification

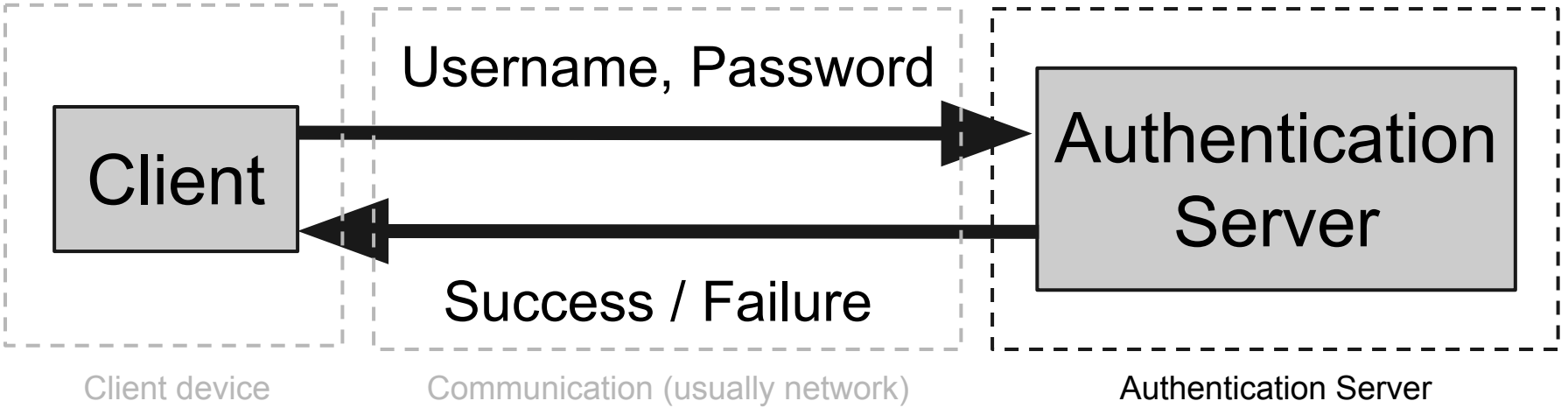


How should we store the passwords on the server?

Threat Model Assumptions

- Attacker's primary method is exfiltration, and
- Attacker can't modify code
  - These assumptions model the typical case and
  - If the attacker can make more important changes, the incursion is probably complete

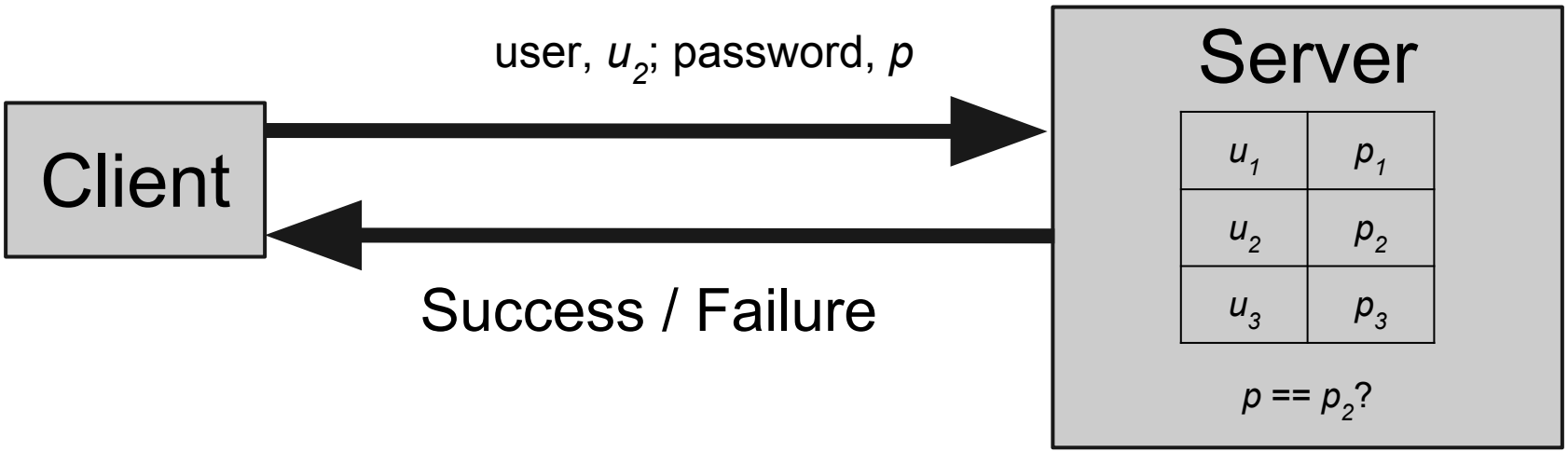
Password Storage



How should we store the passwords on the server?

Attempt #1 - Plaintext

- Store passwords in plaintext



Attempt #1 - Plaintext

- Store passwords in plaintext
- What could go wrong?

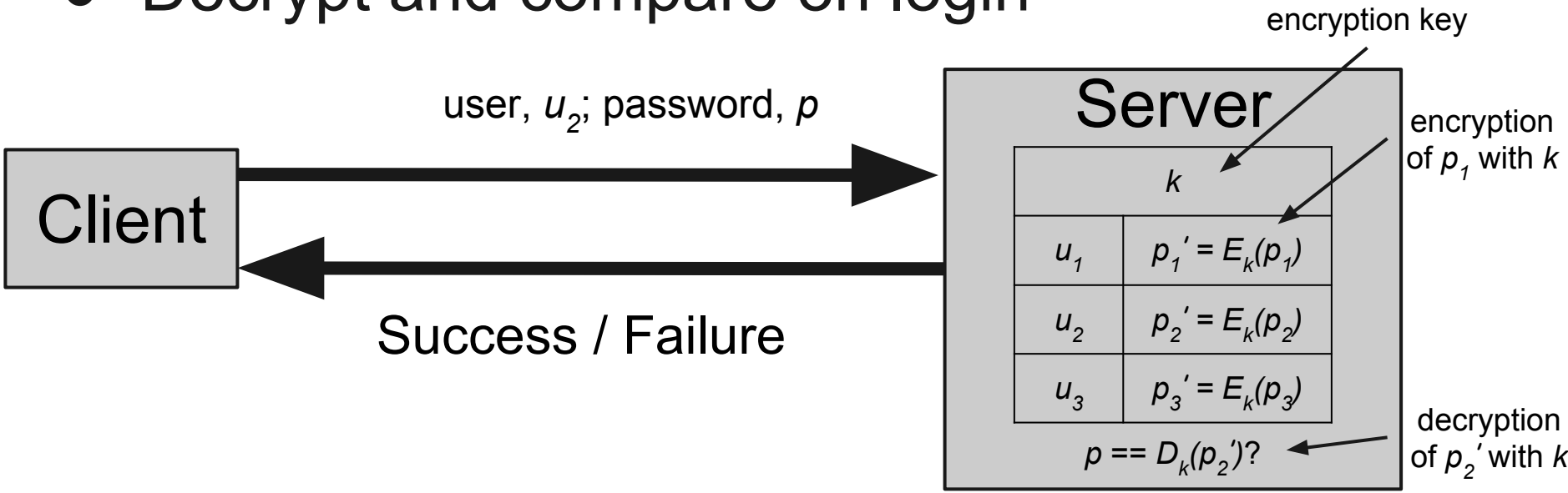
Attempt #1 - Plaintext

- Store passwords in plaintext
- What could go wrong?
  - If database is stolen, so are passwords!
  - Admins have access to passwords.

- ex. Reddit (2006)

Attempt #2 - Encryption

- Encrypt the passwords
- Decrypt and compare on login



Attempt #2 - Encryption

- Encrypt the passwords
- Decrypt and compare on login
- What advantages does this scheme have?

Attempt #2 - Encryption

- Encrypt the passwords
- Decrypt and compare on login
- What advantages does this scheme have?
  - If database is stolen, passwords can’t be read
  - Only administrators with the encryption key can read the passwords

Attempt #2 - Encryption

- Encrypt the passwords
- Decrypt and compare on login
- What advantages does this scheme have?
  - If database is stolen, passwords can’t be read
  - Only administrators with the encryption key can read the passwords
- What could go wrong?

Attempt #2 - Encryption

- Encrypt the passwords
- Decrypt and compare on login
- What advantages does this scheme have?
  - If database is stolen, passwords can’t be read
  - Only administrators with the encryption key can read the passwords
- What could go wrong?
  - If the database is stolen, what is to keep the key from being stolen?
  - Anyone with the key (admins) can view passwords
- ex. Adobe (2013)

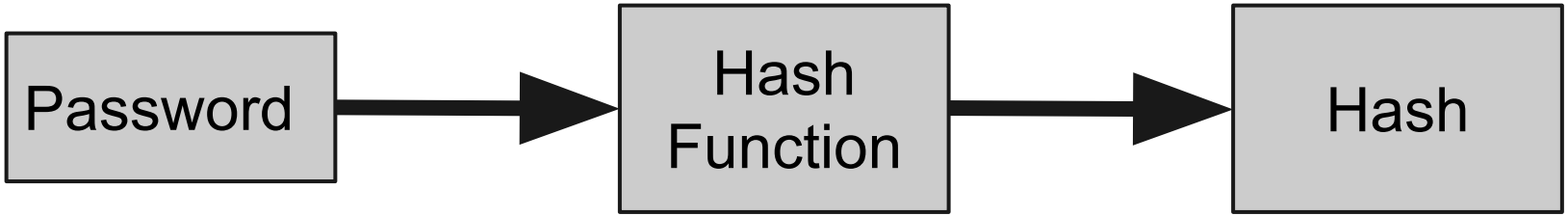
Attempt #2 - Encryption

- Even without the key, it’s still bad. Why?

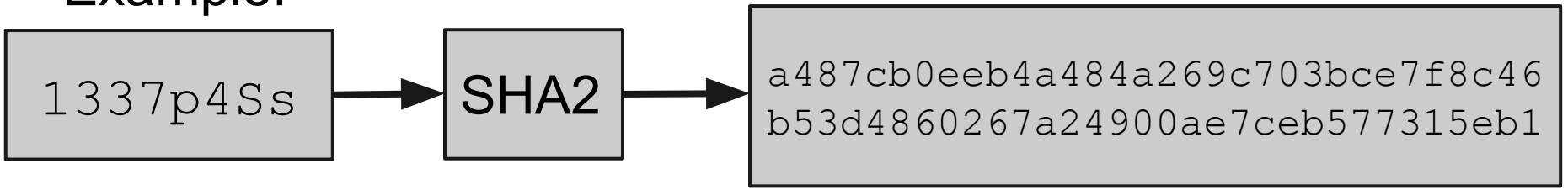
Attempt #2 - Encryption

- Even without the key, it's still bad. Why?
  - *Identical passwords produce identical ciphertexts*
  - If you know one password, you know all with the same ciphertext
  - Frequency analysis (0.5% of users use `password`)
  - Password hints ("numbers 123456")

Attempt #3 - Hashing



Example:



Attempt #3 - Hashing

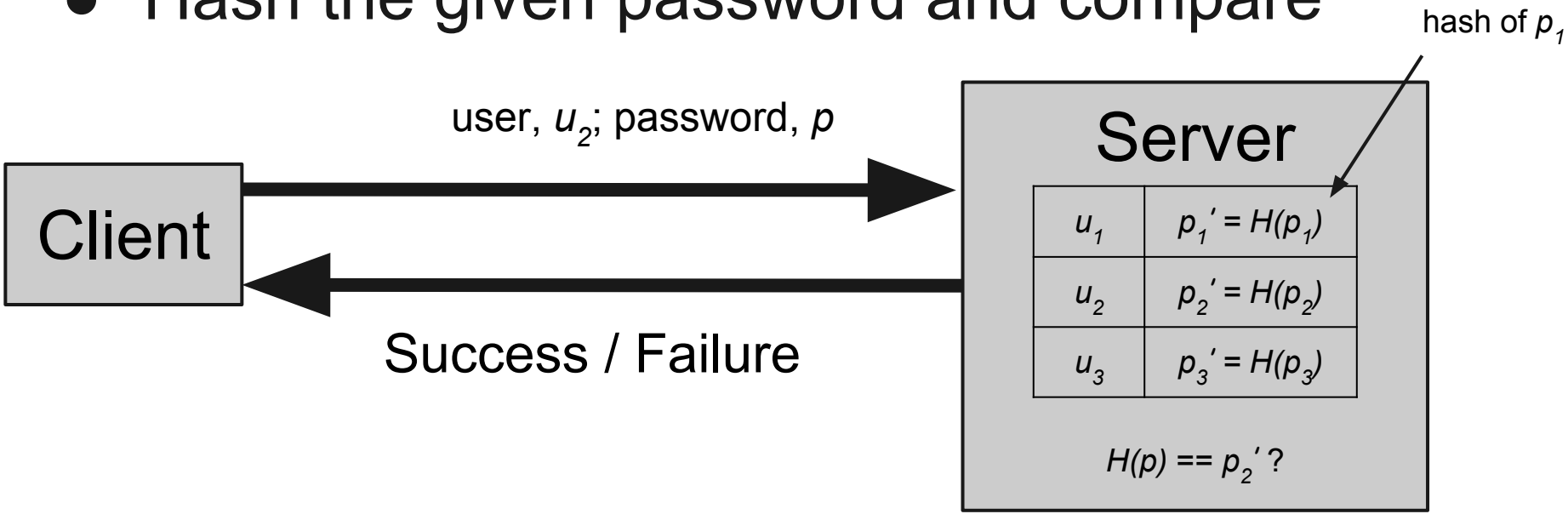
- Recall cryptographic hashing:
  - Variable length input, fixed length “random” output
  - One-way
    - Given hash x, hard to find p such that $H(p) = x$
  - Weak collision resistance
    - Given input p, hard to find q such that $H(p) = H(q)$
  - Strong collision resistance
    - Hard to find distinct p, q such that $H(p) = H(q)$

Attempt #3 - Hashing

- Recall cryptographic hashing:
  - Variable length input, fixed length "random" output
  - One-way
    - Given hash x, hard to find p such that $H(p) = x$
  - Weak collision resistance
    - Given input p, hard to find q such that $H(p) = H(q)$
  - Strong collision resistance
    - Hard to find distinct p, q such that $H(p) = H(q)$

Attempt #3 - Hashing

- Hash the password, store the hash
- Hash the given password and compare



Attempt #3 - Hashing

- Hash the password, store the hash
- Hash the given password and compare
- What advantages does this scheme have?

Attempt #3 - Hashing

- Hash the password, store the hash
- Hash the given password and compare
- What advantages does this scheme have?
  - If database is stolen, hashes need to be cracked
  - Cracking is hard, so attackers get fewer passwords

Attempt #3 - Hashing

- What could go wrong?

Attempt #3 - Hashing

- What could go wrong?
  - *Identical passwords produce identical hashes*
  - Once you've cracked a given hash, you can trivially crack it every time you see the same hash again
  - Frequency analysis
  - Precompute massive tables for popular hash functions
    - These are called rainbow tables
  - Common passwords are *very common*
    - Even a small table cracks most passwords

Attempt #4 - Salting

- What is salt?

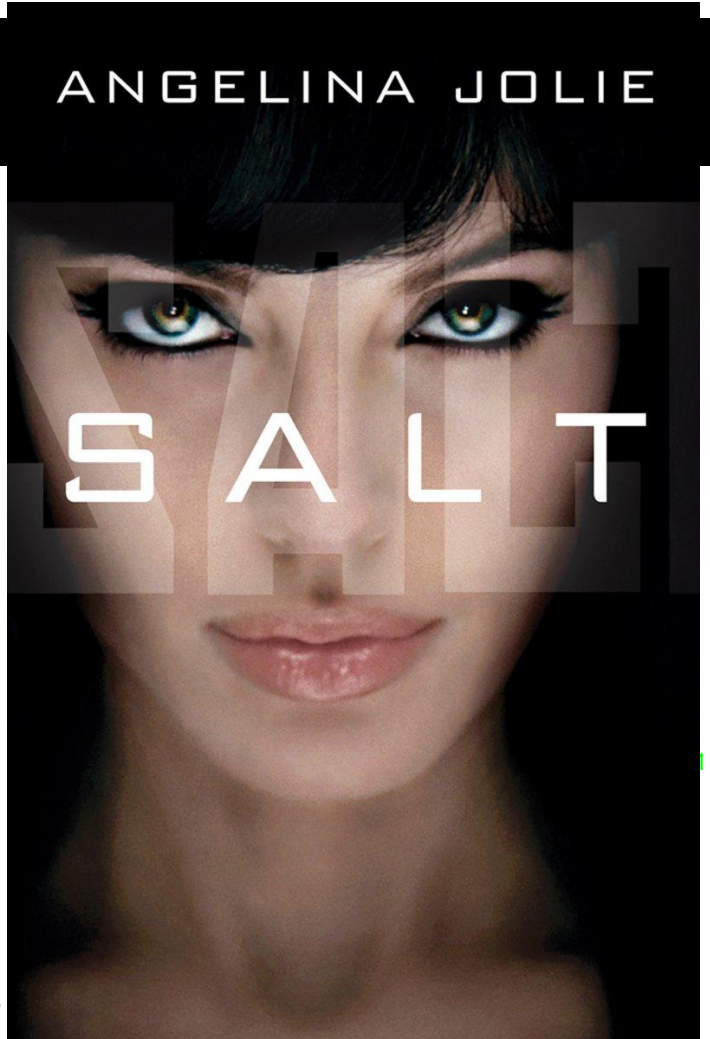
Salt

Common salt is a mineral substance composed primarily of sodium chloride (NaCl), a chemical compound belonging to the larger class of ionic salts; salt in its natural form as a crystalline mineral is known as rock salt or halite. Salt is present in vast quantities in the sea where it is the main mineral constituent, with the open ocean having about 35 grams (1.2 oz) of solids per litre, a salinity of 3.5%. Salt is essential for animal life, and saltiness is one of the basic human tastes. The tissues of animals contain larger quantities of salt than do plant tissues; therefore the typical diets of nomads who subsist on their flocks and herds require little or no added salt, whereas cereal-based diets require supplementation. Salt is one of the oldest and most ubiquitous of food seasonings, and salting is an important method of food preservation.

source: http://www.sunrisefood.com.vn/images/intro/icon_Salt.png

source: https://en.wikipedia.org/wiki/Salt

Salt (2010)

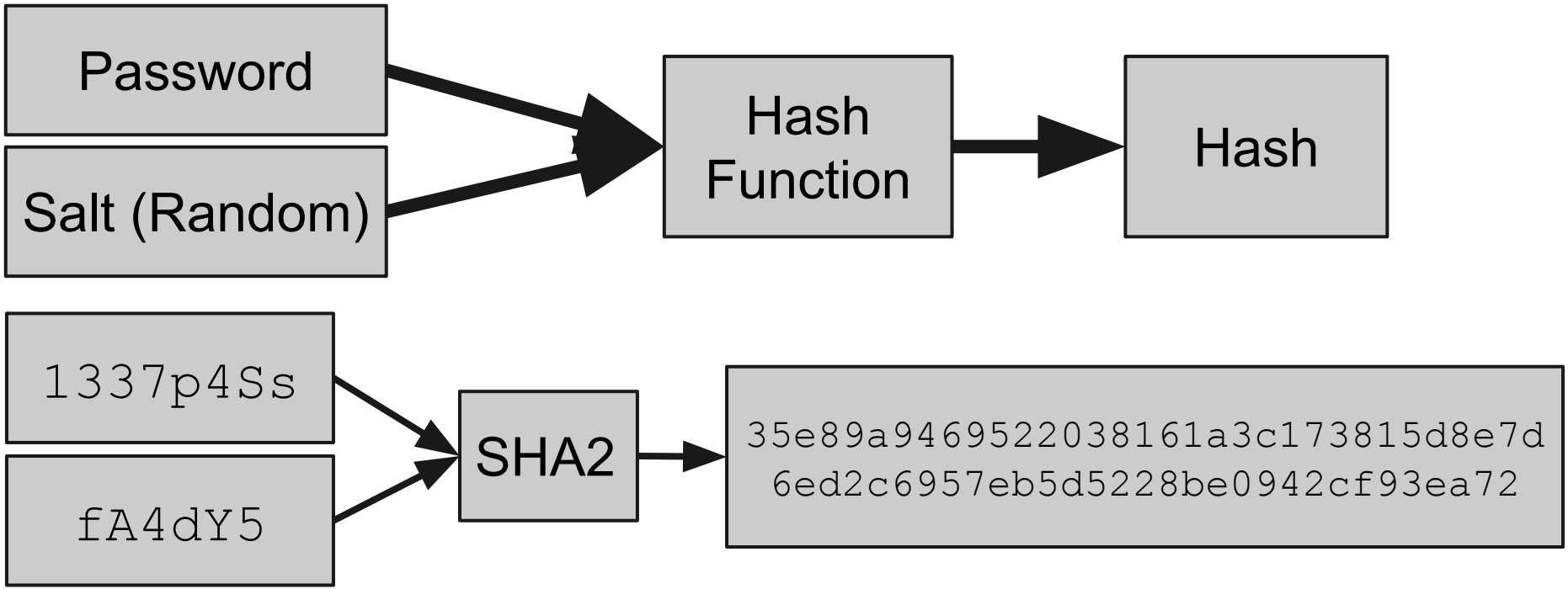


As a CIA officer, Evelyn Salt (Jolie) swore an oath to duty, honor and country. Her loyalty will be tested when a defector accuses her of being a Russian spy. Salt goes on the run, using all her skills and years of experience as a covert operative to elude capture. Salt's efforts to prove her innocence only serve to cast doubt on her motives, as the hunt to uncover the truth behind her identity continues and the question remains: "Who Is Salt?"

Rating: SHA-13 (for intense sequences of violence and cryptography)

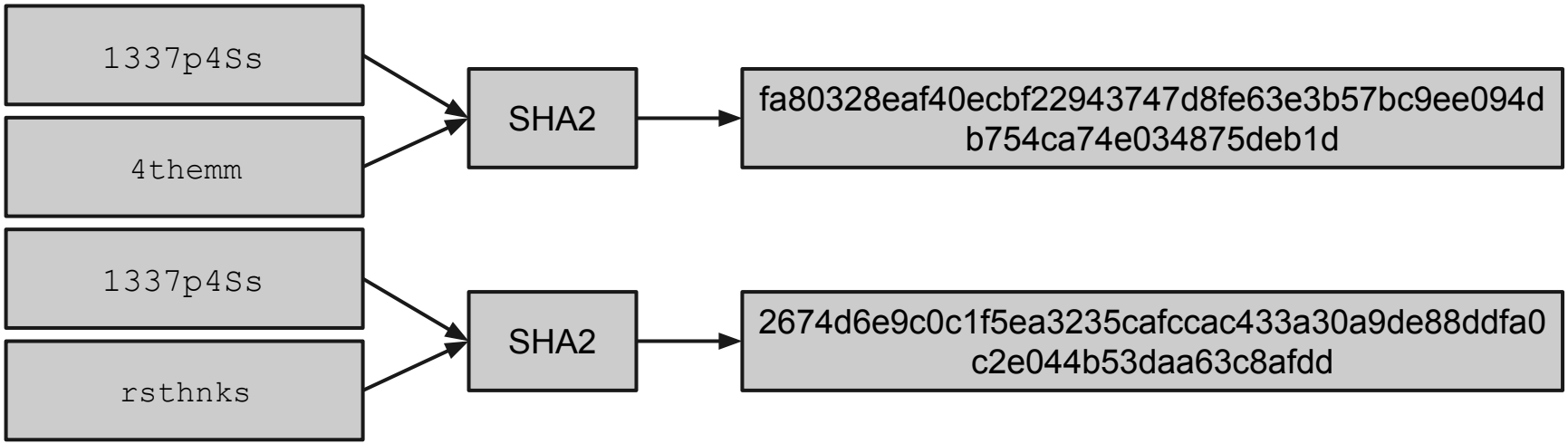
source: https://itunes.apple.com/us/movie/salt/id386251756

Attempt #4 - Salting



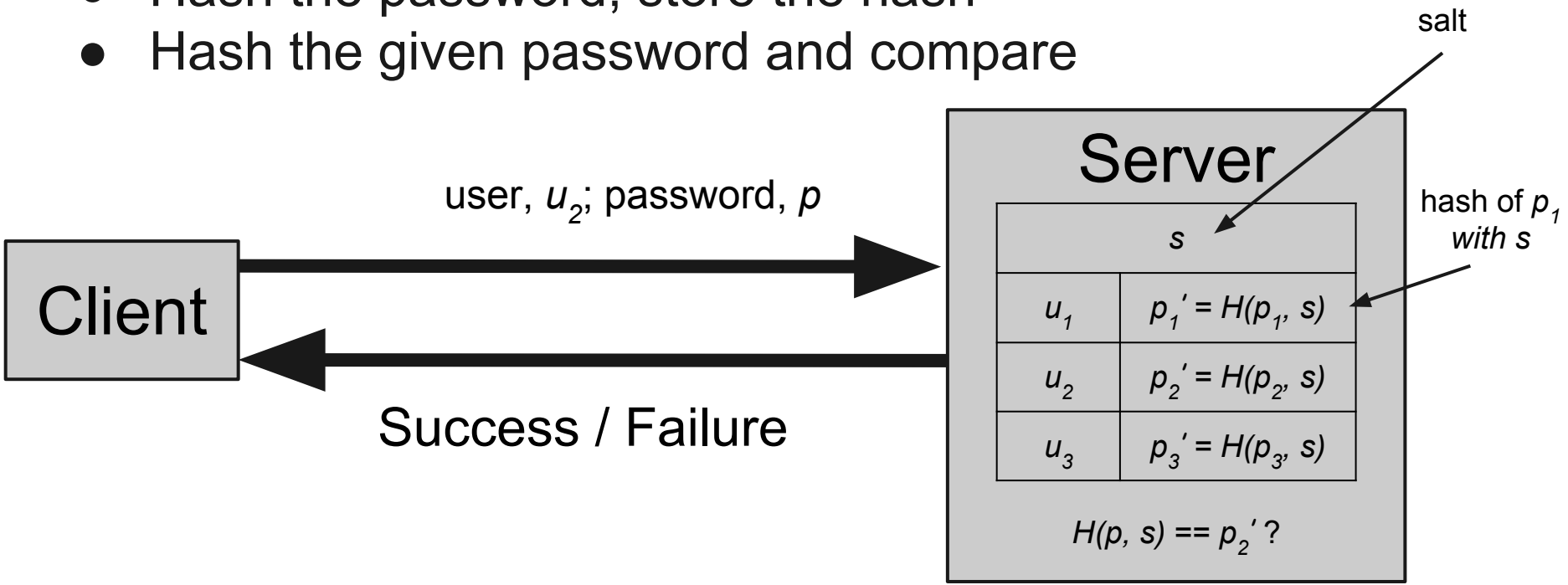
Attempt #4 - Salting

- Hashing same password with different salt will produce different hashes



Attempt #4 - Salting

- Hash the password, store the hash
- Hash the given password and compare



Attempt #4 - Salting

- Hash the password, store the hash
- Hash the given password and compare
- What advantages does this scheme provide?

Attempt #4 - Salting

- Hash the password, store the hash
- Hash the given password and compare
- What advantages does this scheme provide?
  - In order to precompute, need password *and salt*
  - Since salts are random, guessing a salt is useless
  - Even if salt is known, computation must be redone for every site

Attempt #4 - Salting

- What could go wrong?

Attempt #4 - Salting

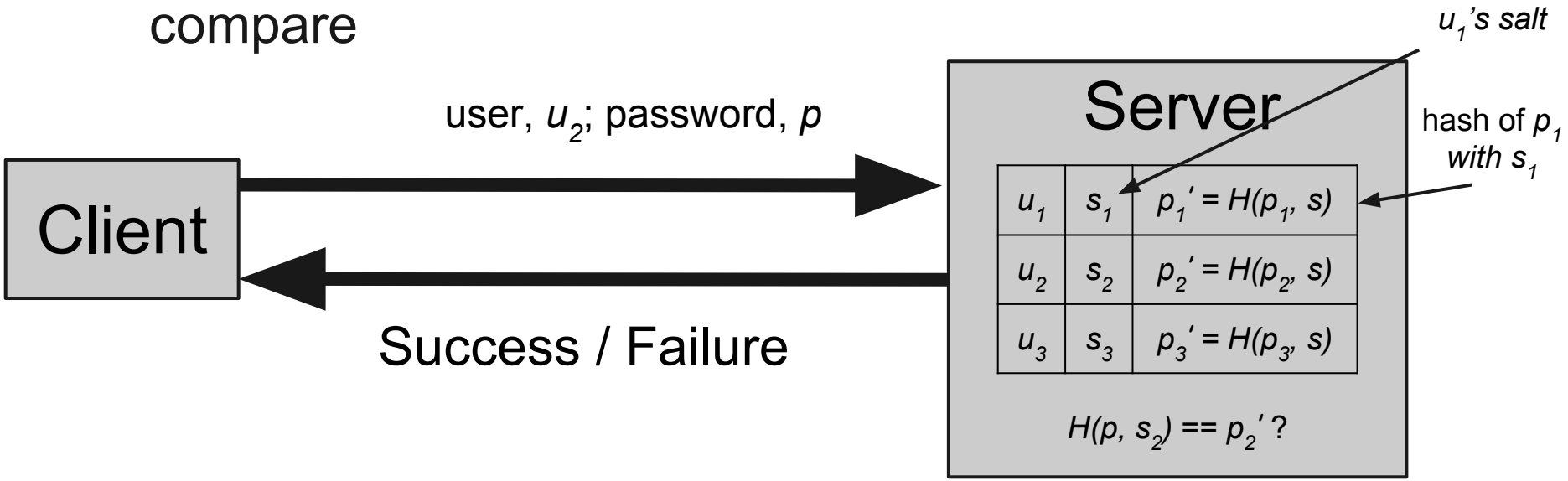
- What could go wrong?
  - *Identical passwords and identical salts produce identical hashes*
  - Frequency analysis
  - If you crack one password, you crack them all
  - For big sites, precomputation is worth it

Attempt #4 - Salting

- How could we make the analysis even harder?

Attempt #5 - Per-User Salting

- Generate a salt, hash the password, store salt and hash
- Hash the given password with the user's salt and compare



Attempt #5 - Per-User Salting

- Generate a salt, hash the password, store the hash
- Hash the given password with the user's salt and compare
- What are the advantages of this scheme?

Attempt #5 - Per-User Salting

- Generate a salt, hash the password, store the hash
- Hash the given password with the user's salt and compare
- What are the advantages of this scheme?
  - Since every user has a different salt, *identical passwords will not have identical hashes*
  - No frequency analysis
  - No using known passwords to crack other passwords
  - No precomputation - much harder to crack

Work Factor

- Measure of “work” needed **per cracked password**
- Intuitively, higher work factor means greater security

- Work factor for plaintext model is nominally 1.
- For non-salted hashes, construction of rainbow table requires a large but fixed pre-computation. Work per cracked password decreases with number of crackings.

Work Factor

- Measure of “work” needed **per cracked password**
- Intuitively, higher work factor means greater security

- Work factor for plaintext model is nominally 1.
- For non-salted hashes, construction of rainbow table requires a large but fixed pre-computation. Work per cracked password decreases with number of crackings.
- For individually salted passwords, work is maximal.

Slow Hashes

- Work factor increases if hashing is slow

- ex., bcrypt, PBKDF2 are designed to be slow
- Both feature a work factor parameter
- This work factor tunes how many times the hash is computed (*H(p)* vs *H(H(p))*, etc)

Intelligent Guessing

- For any scheme that involves guessing, work factor is reduced by **guessing intelligently**
- Key insight: *not all passwords are equally likely*
- Idea: try most likely passwords first
- In reality, it doesn't take 9 years to brute force most passwords

How Unequal?

- From a study of the Adobe breach
  - `password` - 0.5%
  - a password in the top 10 - 1.6%
  - a password in the top 100 - 4.4%
  - a password in the top 1,000 - 13.2%
  - a password in the top 10,000 - 30%

Intelligent Guessing Methods

- Try the top *n* most common passwords
- Dictionary of words, names, etc
- Syntax model
  - e.g., “dictionary word with some letters replaced by numbers” - `elitenoob`, `e1iten00b`, `31it3n00b`
- Markov chain model