openHPI Course: Digital Identities – Who am I on the Internet?

# Password Security – Authentication using Passwords

**Prof. Dr. Christoph Meinel**

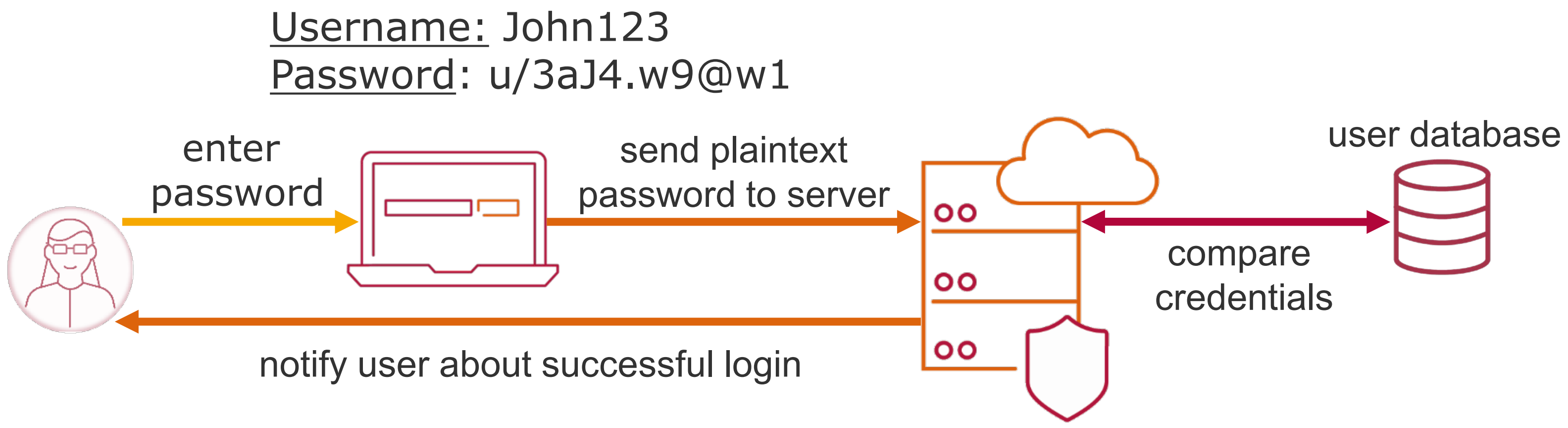
Hasso Plattner Institute
University of Potsdam, Germany

## Authentication using Passwords (1/2)

A very popular method for binding a digital identity to a user is password authentication

- User proves that a digital identity belongs to them by entering the correct **password** (authentication by knowledge)
- Online service checks the correctness of the password by comparing it with the information collected during registration and stored in the user database
  - if it matches, the user is given access to the service and all authorizations related to the digital identity
- Passwords stored in the user database should never be stored in plaintext, but only in a disguised form
  - obfuscation by means of **cryptographic hash functions**

# Authentication using Passwords (2/2)



- User enters username and plaintext password on the login form of the service’s website displayed in the browser
- Website sends password together with the username to the (server of the) online service
- Server *hashes* password and compares it with the hashed password stored for the user with that username in the user database

## Vulnerability: Plaintext Passwords

- Provider stores password in plaintext and not hashed in the user database when registering and setting up a user account (digital identity)
- If cybercriminals succeed in accessing the IT system of the online service via the Internet, they can **download user database with all passwords**
- **Why are stolen passwords so dangerous?**
  - Attacker can log in to the online service with a stolen password
  - Attacker can try to use the same or slightly modified username/password combination with other services
  - Users usually know nothing about the theft of their identity data

## Secure Storage of Passwords

**How passwords can be stored securely?**

- Storage of passwords in a cryptographic encrypted form by means of hash methods
  - **hashing** – concealment
  - transformation in a string of fixed length
- Experience with the **HPI Identity Leak Checker**
  - many providers still store passwords in plaintext
  - often if passwords are stored in a hashed form, outdated and relatively easily crackable hashing methods are used, such as MD5

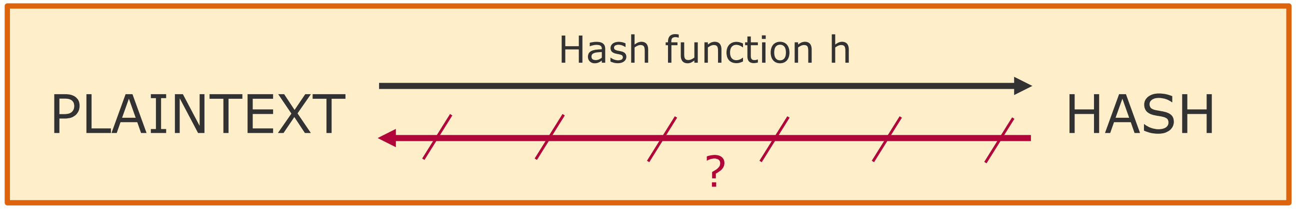
## Password Concealment by Hashing (1/2)

- Passwords are hidden by means of **cryptographic hash functions**
- A hash function transforms a password into a cipher text of fixed length ("hash"), e.g.

$h_{MD5}$(u/3aJ4.w9@w1) = 64038fd70a32bf67436dfdd2ab44dfb0

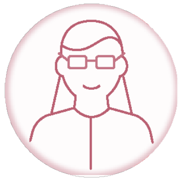
# Password Concealment by Hashing (2/2)

- Hash functions are designed in such a way that it is practically impossible (i.e. only possible with an extremely high effort) to recover the original password from the hash (**one-way function**)



- Commonly used hash methods are MD5 and SHA1
  - **but**: MD5 and SHA1 are now considered unsecure because they are reversible with the computational power of recent computers
- Hash methods SHA-2, SHA-3 and bcrypt are considered safe

## Validation of a Password



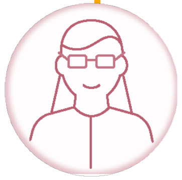
Username: John123

Password: u/3aJ4.w9@w1

- User enters his/her plaintext password

# Validation of a Password

| user | password_hash |
|---|---|
| Hans74 | 8b95abb0f2d7dddec8cd0069001f2899 |
| Anna4 | 6f67bb3d829b408fce3859a24e902fbc |
| Paul89 | 42fdbeb6772cc3e427cf1f5f63a836a2 |
| John123 | **64038fd70a32bf67436dfdd2ab44dfb0** |



Username: **John123**

Password: u/3aJ4.w9@w1

- User`s hashed password is retrieved from database

# Validation of a Password

| *user* | *password_hash* |
|---|---|
| Hans74 | 8b95abb0f2d7dddec8cd0069001f2899 |
| Anna4 | 6f67bb3d829b408fce3859a24e902fbc |
| Paul89 | 42fdbeb6772cc3e427cf1f5f63a836a2 |
| John123 | **64038fd70a32bf67436dfdd2ab44dfb0** |

Username: John123

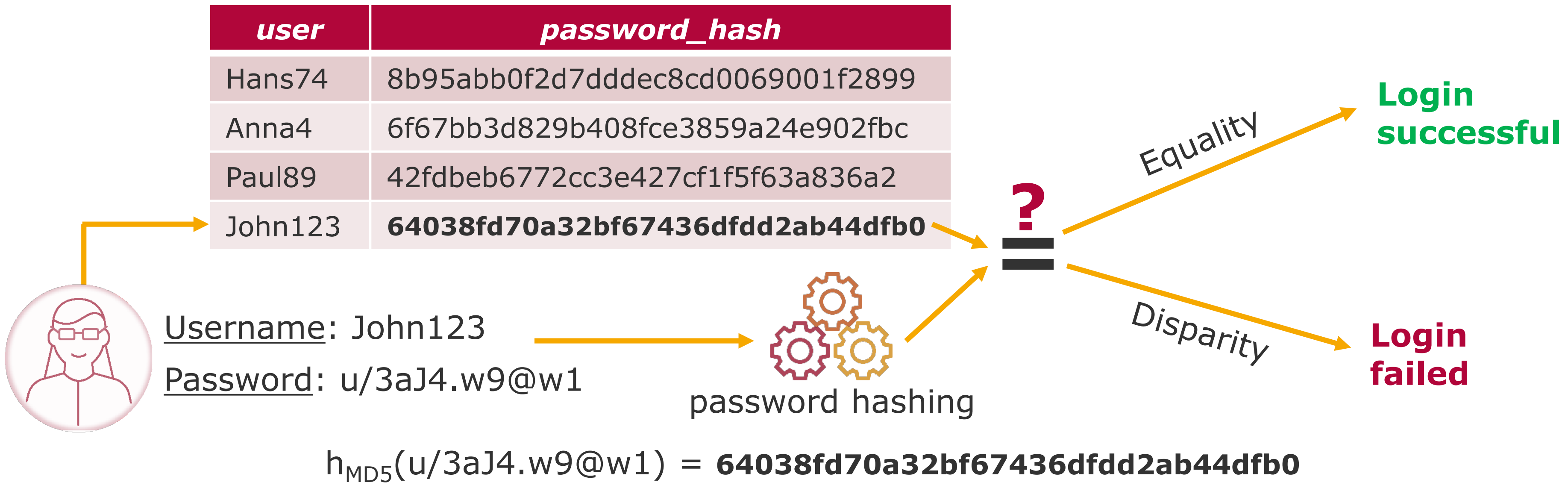
Password: u/3aJ4.w9@w1

password hashing

$h_{MD5}$(u/3aJ4.w9@w1) = **64038fd70a32bf67436dfdd2ab44dfb0**

- Calculation of the hash for the plaintext password entered

# Validation of a Password



| *user* | *password_hash* |
|---|---|
| Hans74 | 8b95abb0f2d7dddec8cd0069001f2899 |
| Anna4 | 6f67bb3d829b408fce3859a24e902fbc |
| Paul89 | 42fdbeb6772cc3e427cf1f5f63a836a2 |
| John123 | **64038fd70a32bf67436dfdd2ab44dfb0** |

Username: John123
Password: u/3aJ4.w9@w1

$h_{MD5}$(u/3aJ4.w9@w1) = **64038fd70a32bf67436dfdd2ab44dfb0**

- Comparison of both hashes
- Login is successful if both hashes match

# Secure Hashes with Salt (1/2)

- Password hiding by hashing has further weak spots

| *user* | *password_hash* | *password_plain* |
|---|---|---|
| User A | 5f4dcc3b5aa765d61d8327deb882cf99 | password |
| User B | 5f4dcc3b5aa765d61d8327deb882cf99 | password |

- Two users with the same password also have the same hash value in the database
- If an attacker knows a user's password, he knows the passwords of all users with the same password
- Hash can be guessed by hashing all words in a dictionary, e.g. "password"

## Secure Hashes with Salt (2/2)

**Solution**:

The same password is disguised differently every time

- **Idea**: Before the hash value of a password is calculated, the password is extended by a random string (salt)

$$h_{MD5}(PASSWORD+SALT) = HASH$$

- Salt is then stored together with the password hash

| *user* | *password_hash* | *salt* | *password_plain* |
|---|---|---|---|
| User A | ec58ef6c6f345d99054e419d19be6e3f | 5<§ | password |
| User B | 188623e1be9c480214838bf596b22d0d | =3( | password |

## Authentication with Passwords
## **Summary**

- Authentication through **knowledge**
  - by entering the correct password a user proves, that a digital identity belongs to him/her
- Online service **stores** the password when registering a digital identity and **validates** the password during the login process
  - passwords should never be stored in plaintext, but only in a disguised form (**password hashing**)
- To increase security, only secure hash methods and salts should be used for password concealment