Best Practices for User Management

Kevin Dockx
Architect

@KevinDockx https://www.kevindockx.com

Coming Up

Where should user management screens live?

Implementing a user registration flow

- Registering a user
- Safely storing passwords
- Activating an account

Additional tasks and best practices

Where Should User Management Screens Live?

In a separate application?

At level of the IDP?

A combination of both?

Where Should User Management Screens Live?

Do you want to be able to scale up the IDP without having to scale up a user management / admin application?

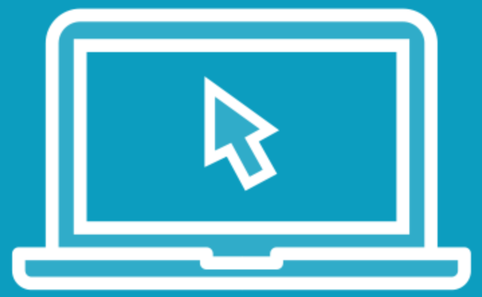
Do you want to be able to deploy them separately?

Are you ok with managing different solutions and code bases or do you want to avoid that?

Is the risk that changing something at level of the user management screens has an effect on your IDP worth it?

...

Demo



Implementing a user registration screen

Safely Storing Passwords

Passwords should be stored after being salted, hashed, and key-stretched

Salt

A cryptographically random piece of data that's attached to the password before it's hashed

A salt serves as additional input for a hashing function

- Stored next to the hashed password
- ... or attached to it

Use a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG) to generate it

Salting protects against dictionary attacks

- Lookup table attack
- Rainbow table attacks

Hashing

Performing a one-way transformation on a password which turns the password into another string

Hashing is a one-way transformation

- Almost impossible to turn the hashed password back into the original password

SHA256 / SHA512

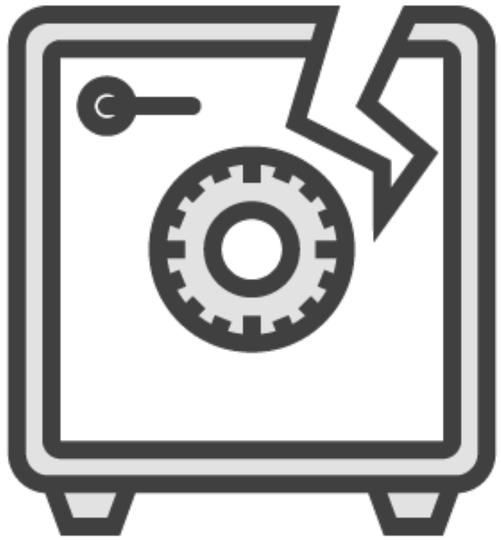
Hashing != encryption

- Encryption is a two-way transformation, you can decrypt something back to its original value after having it encrypted

Hashing ensures passwords cannot be reverted back to their original value

Salting protects against dictionary attacks

- Lookup table attack
- Rainbow table attacks

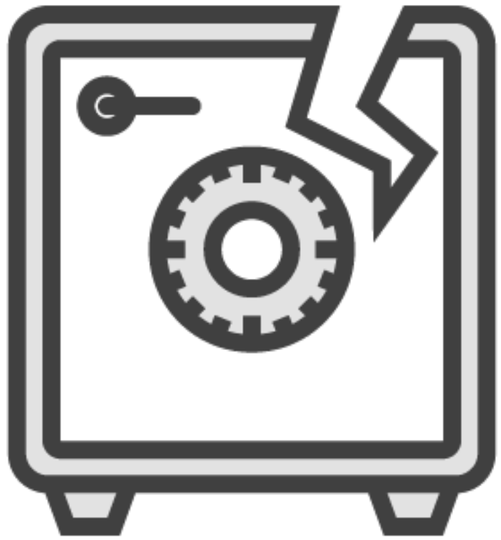


Lookup table attack

- Password dictionary contains a pre-computed list of hashes and their corresponding passwords

Rainbow table attack

- Smaller lookup table by sacrificing hash cracking speed (more effective as more hashes can be stored in the same space)



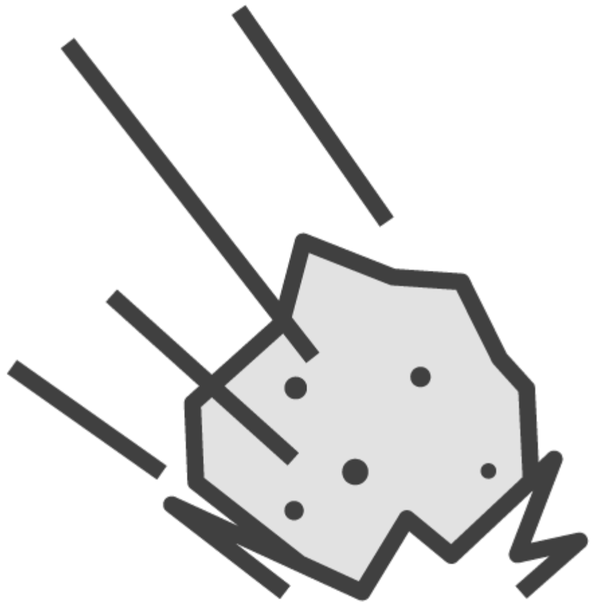
Salting protects against both dictionary attacks

- As the salt is applied to the password before hashing it, the hash in the lookup table will not match the hash in your database

Key stretching

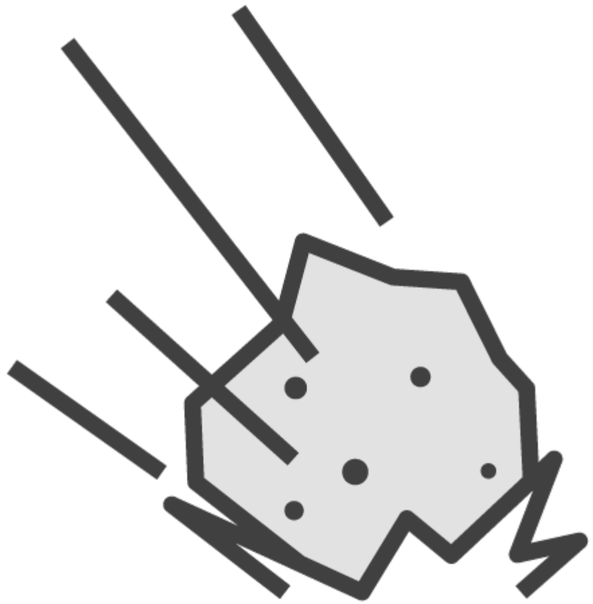
A technique to discourage brute forcing a password by hashing it 1000's of times instead of just once

Password protection techniques are not here to protect someone from another person logging in to an application. They are here to protect the password itself.



Brute force attack

- Trying every possible combination (starting with common passwords), applying the salt, hashing it, and comparing it with the stored hash
- If the new hash matches the stored hash, attackers now know the password



Brute force attack

- Requires a lot of computing power
- … but modern-day CPUs or GPUs can try millions of combinations each second

Key stretching

A technique to discourage brute forcing a password by hashing it 1000's of times instead of just once

PBKDF2 and Argon 2 implement key stretching / key derivation

- Results in a salted & stretched password hash

PBKDF2

- Salt (generated with a CSPRNG) is added to the password
- A pseudorandom function is used to process the password
  - HMAC is the most common one. This internally uses a cryptographic hash function like SHA256/SH1512
  - The process is numerous times for key stretching, which results in the derived key

It's important to regularly evaluate

- The amount of key stretching
- The key derivation function
- The hashing function

Demo

Safely storing passwords

Activating an Account

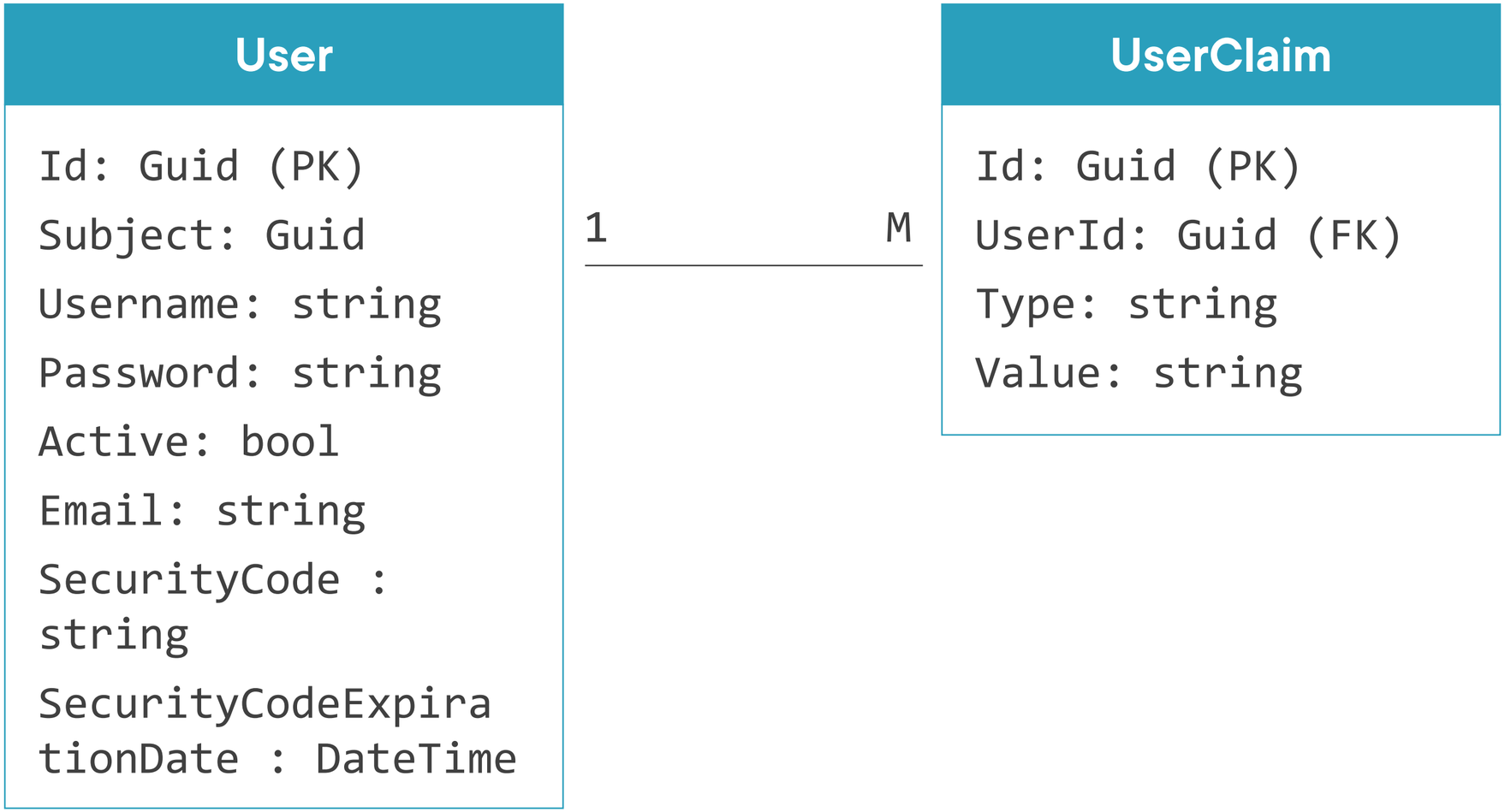
Store an email address to be able to:

- Contact the user
- Implement password resets

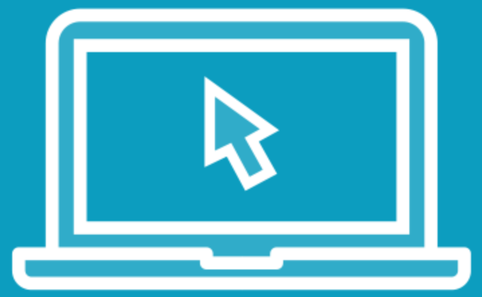
Verify the email address as part of an account activation process

- Ensures the address is real
- Ensures the user owns the address

Activating an Account



Demo



Activating an account

Additional User Management Related Best Practices

Not all IAM systems are created equal

- Functionality depends on your use case

If you allow users to manage their email address, verify it before using it by sending a confirmation link with a token

Implement resend link functionality, as an activation/confirmation/password reset link is only valid for a set amount of time

If you allow password resets, the user identity must be verified to allow them to reset their password

- Avoid common questions
- Instead, send an email with a password reset link to the user's verified email address
- Make sure the email address was verified first!

Locking out users discourages brute force attacks...

... but it's best to avoid locking out users

- Can easily be abused and can cause a DoS attack
- Use key stretching (or a CAPTCHA) to discourage brute force attacks

Password Policy Best Practices

Outdated password policy: force the use of complex characters:

- Remedy: long passwords are better

Outdated password policy: force users to regularly change their passwords:

- This leads to users choosing variations of existing passwords

Password Policy Best Practices

Don’t force users to change passwords regularly

Encourage long passwords or pass phrases

Encourage the use of password manager (& allow copy/pasting in the password field)

Check passwords against a database of often-used passwords

Encourage 2FA/MFA

Summary

User management screens can be implemented at level of the IDP and/or in a separate application

- `IIdentityServerInteractionService`
- Pass through `returnUrl`

Summary

Passwords should be salted, hashed and key stretched

- `IPasswordHasher`

Store the email address, but verify it before using it

Up Next:

Integrating with Active Directory, Azure Active Directory and Social Logins