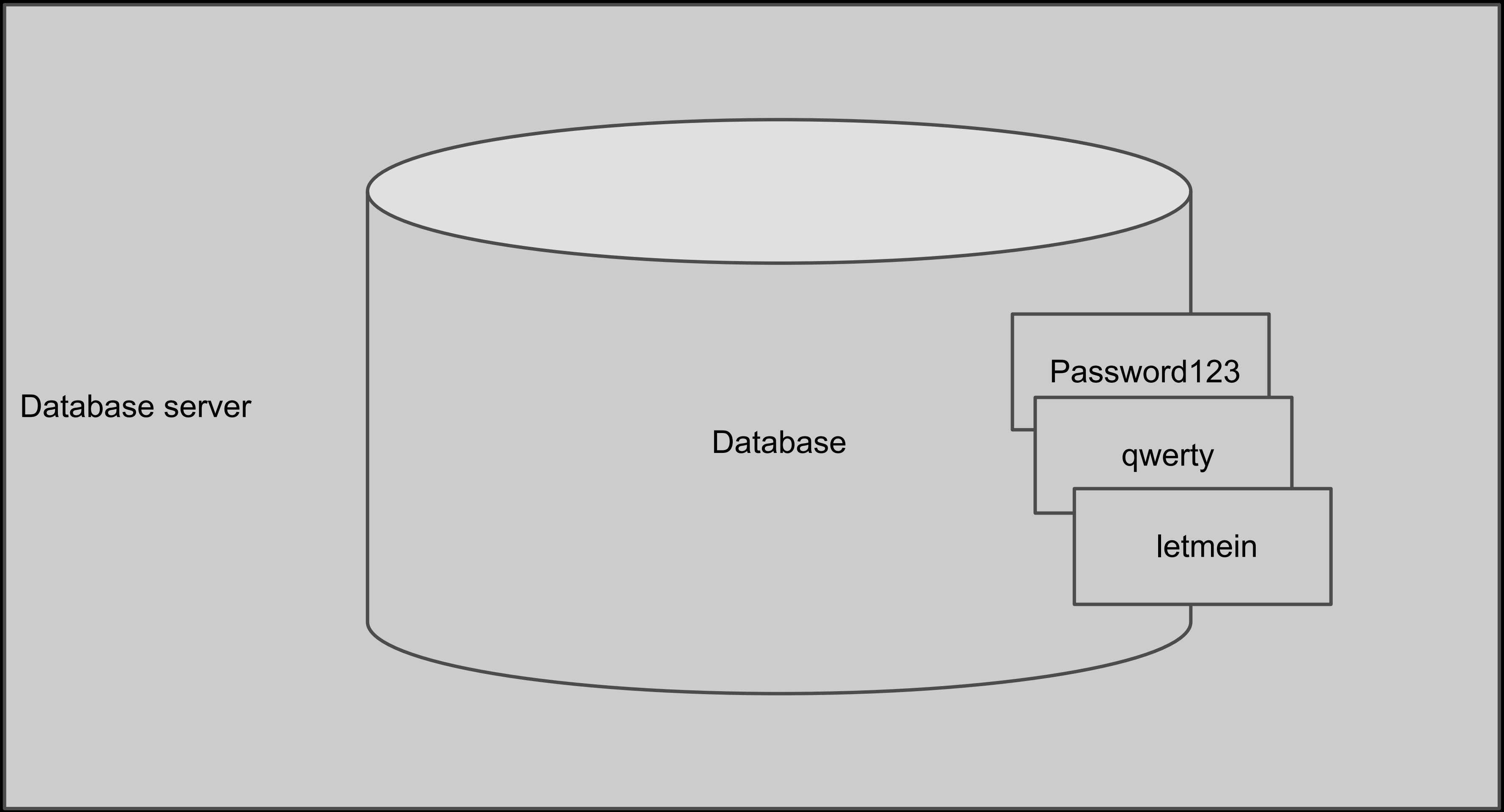
Password Security

Ryan Mulligan

Passwords

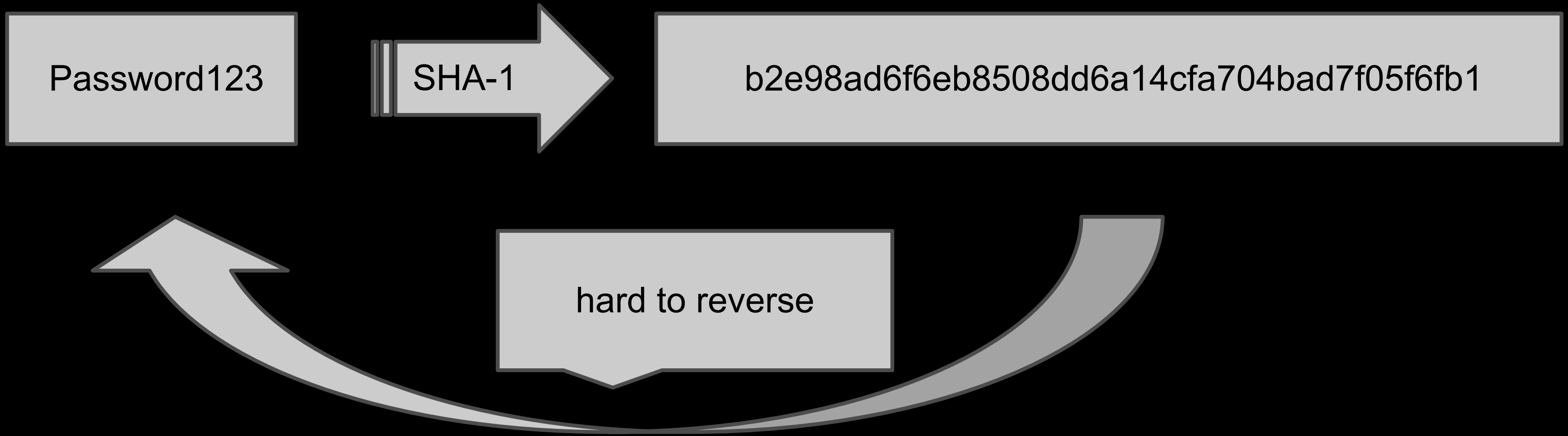
- storing passwords is inherently insecure
  - OpenID, Facebook, Twitter, etc.
- password security is a matter of time

Plain text

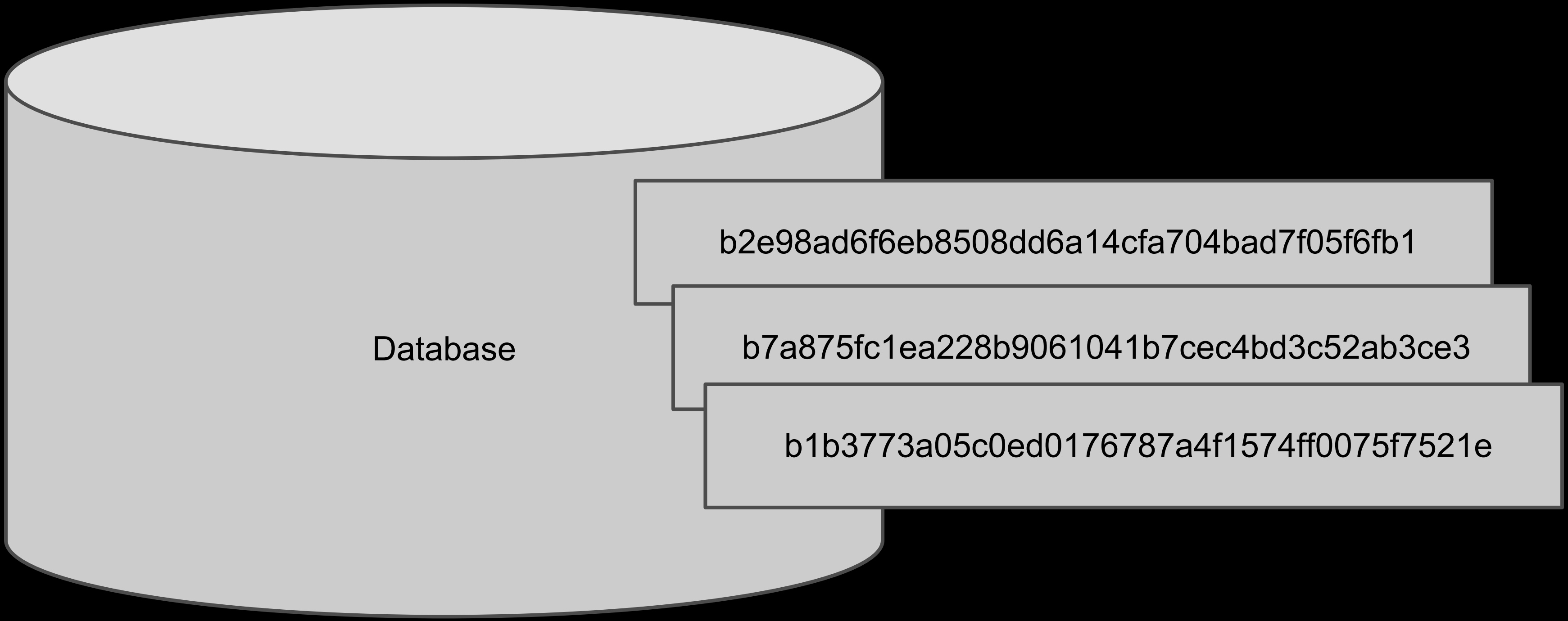


HECK NO!

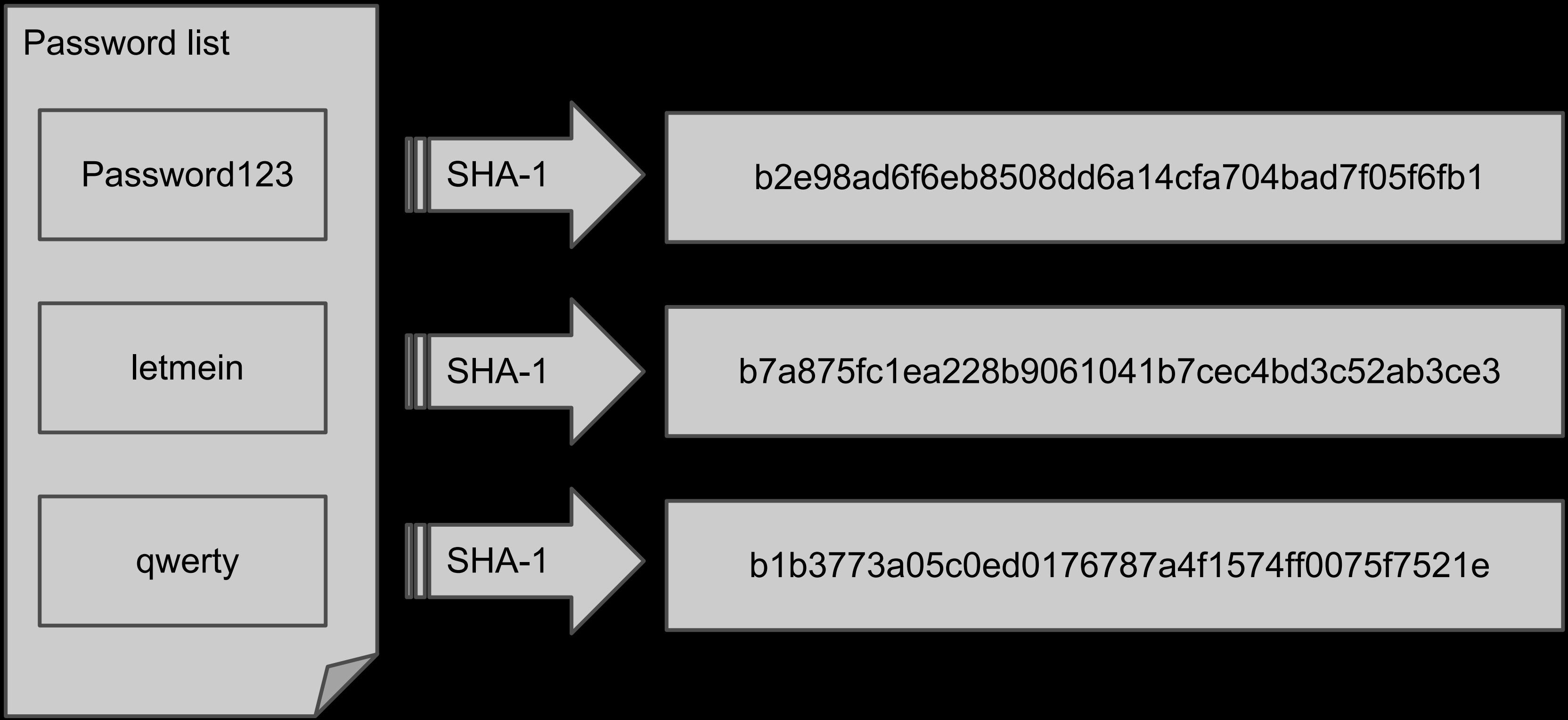
Hash


Password123
SHA-1
b2e98ad6f6eb8508dd6a14cfa704bad7f05f6fb1
hard to reverse

Hash


Database
b2e98ad6f6eb8508dd6a14cfa704bad7f05f6fb1
b7a875fc1ea228b9061041b7cec4bd3c52ab3ce3
b1b3773a05c0ed0176787a4f1574ff0075f7521e

Defeating Hashes



Password list
Password123
letmein
qwerty
SHA-1
SHA-1
SHA-1
b2e98ad6f6eb8508dd6a14cfa704bad7f05f6fb1
b7a875fc1ea228b9061041b7cec4bd3c52ab3ce3
b1b3773a05c0ed0176787a4f1574ff0075f7521e

hash-password lookup table

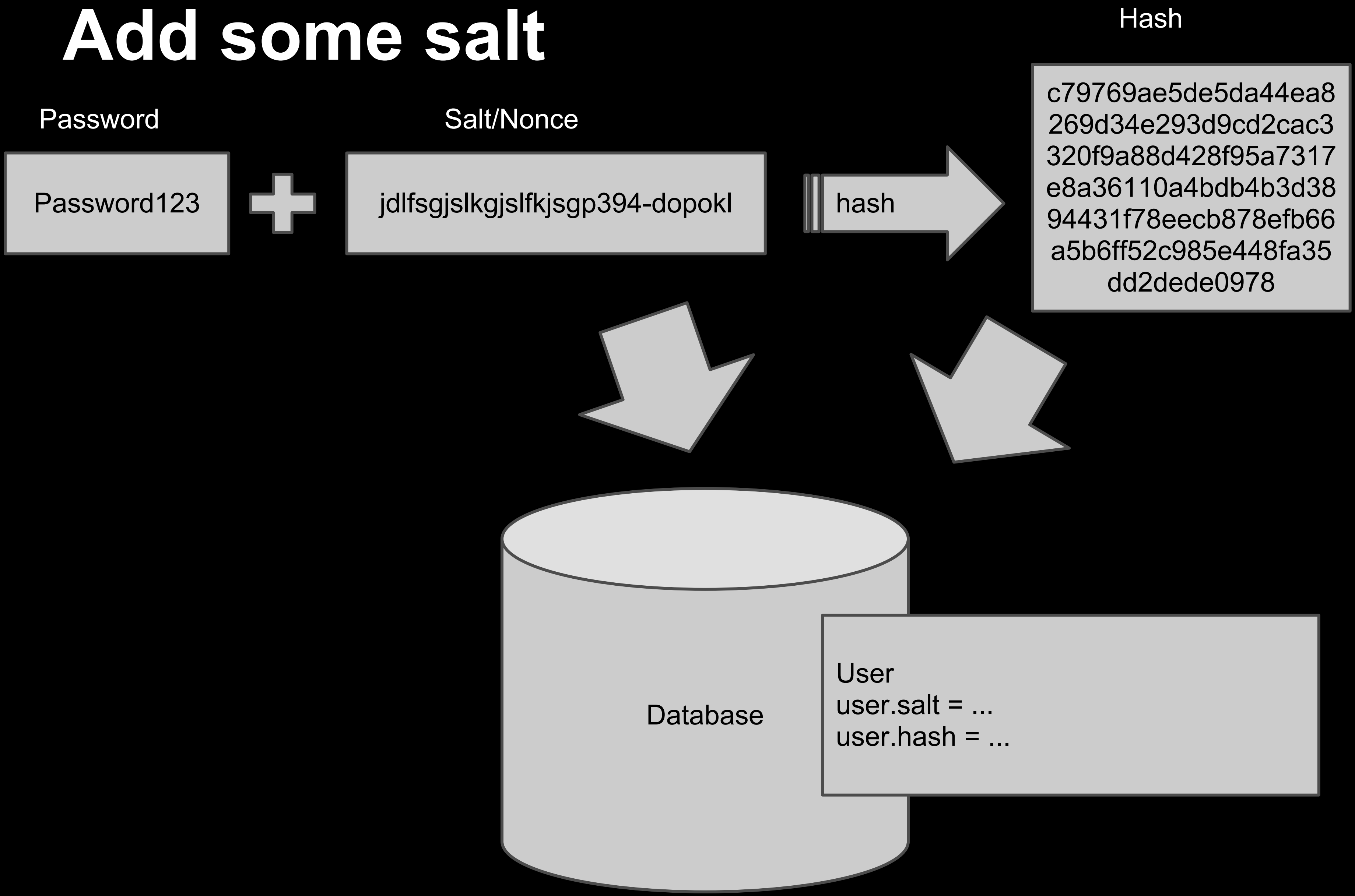
We can do better!

Slow down

Hashes like MD5 and SHA-1 are designed to be fast!

- bcrypt
  - cost function lets you set how slow you want to go
  - bcrypt-ruby gem

Add some salt



Salts kill rainbow tables

- unique salt for each user
- new lookup table for every user

Using bcrypt-ruby

```ruby
include BCrypt

# hash a user's password
@password = Password.create("my grand secret")
@password #=>
"$2a$10$GtKs1Kbsig8ULHZzO1h2TetZfhO4Fmlxphp8bVKnUlZCBYYClPohG"

# store it safely
@user.update_attribute(:password, @password)

# read it back
@user.reload!
@db_password = Password.new(@user.password)

# compare it after retrieval
@db_password == "my grand secret"  #=> true
@db_password == "a paltry guess"   #=> false
```

Using bcrypt-ruby

```
"$2a$10$GtKs1Kbsig8ULHZzO1h2TetZfhO4Fmlxphp8bVKnUlZCBYYClPohG"
```

- salt is part of hash
- version (2a) and cost (10) are also part of it
- you can up the cost later without changing your database

Conclusion

- no password security scheme is uncrackable
- you can easily make the cracking time long
  - use a slow hash function
  - make sure it uses salts