OntarioTech
Business & IT

Applied Cryptography

By

Dr. Madani

Winter 2023

Learning Objectives

Upon completion of this material, you should be able to

- Understand the importance of password salting.
- Explain how hashing can be used for secret storage.
- Distinguish between password/passphrases and key derivation functions.
- Explain the importance of cryptographic random number generators.
- Describe the benefits of Certificate Pinning.

Random Number Generators (RNGs)

- Three main categories

1. True Random Number Generator (TRNG)
    - Based on a physical random process: coin flipping, radioactive decay, mouse movement
    - Given a sequence of n output, cannot predict $n+1$ with certainty

2. Pseudorandom Number Generator (PRNG)
    - Generates the sequence from an initial seed value
    - typically, the output stream has good statistical properties (e.g., uniformity)

$$s_0 = seed$$
$$s_{i+1} = f(s_i, s_{i-1}, \dots, s_{i-t})$$

    - Most PRNGs have **bad** cryptographic properties

3. Cryptographically Secure Random Number Generator (SRNG)
    - Special PRNG when output is unpredictable, i.e., given n consecutive bits of output s_i

OntarioTech
Business & IT

Random Number Generators (RNGs) – cont'd

- Using SRNG (when random numbers are needed) is imperative to security of your cryptographic scheme used in your application.
  - Use audited SRNG libraries – DO NOT IMPLEMENT YOUR OWN VERSION!

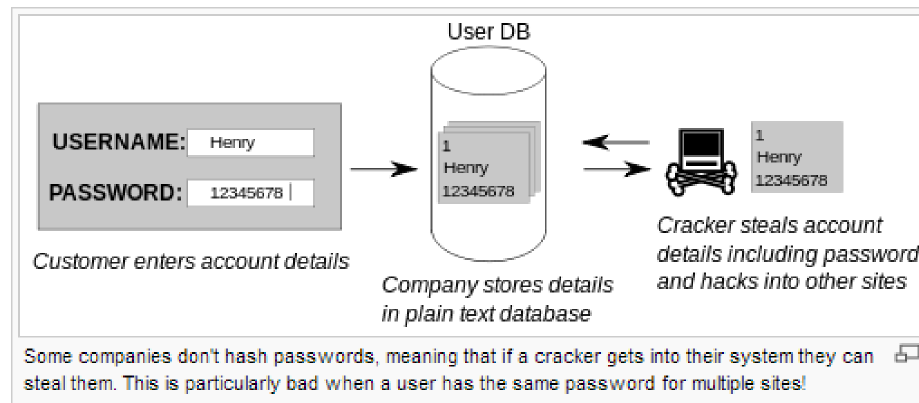
```
>>> from random import SystemRandom
>>> cryptogen = SystemRandom()
>>> [cryptogen.randrange(3) for i in range(20)] # random ints in range(3)
[2, 2, 2, 2, 1, 2, 1, 2, 1, 0, 0, 1, 1, 0, 0, 2, 0, 0, 0, 0]
>>> [cryptogen.random() for i in range(3)]  # random floats in [0., 1.)
[0.2710009745425236, 0.016722063038868695, 0.8207742461236148]
```

```
>>> import secrets
>>> secrets.token_urlsafe(16)
'zs9XYCbTPKvux46UJckflw'
>>> secrets.token_hex(16)
'6bef18936ac12a9096e9fe7a8fe1f777'
```

OntarioTech Business & IT

Password Hashing

- Password Hashing
  - In most systems passwords are stored in a protected (i.e., hash) form.
  - Snooper that gains internal access to the system cannot easily retrieve/steal passwords.
- Login Process
  1. Every time a user logs in, password handling software runs the hash algorithm against the submitted password value.
  2. If (computed hash value === stored hash value) ➔ access is granted.



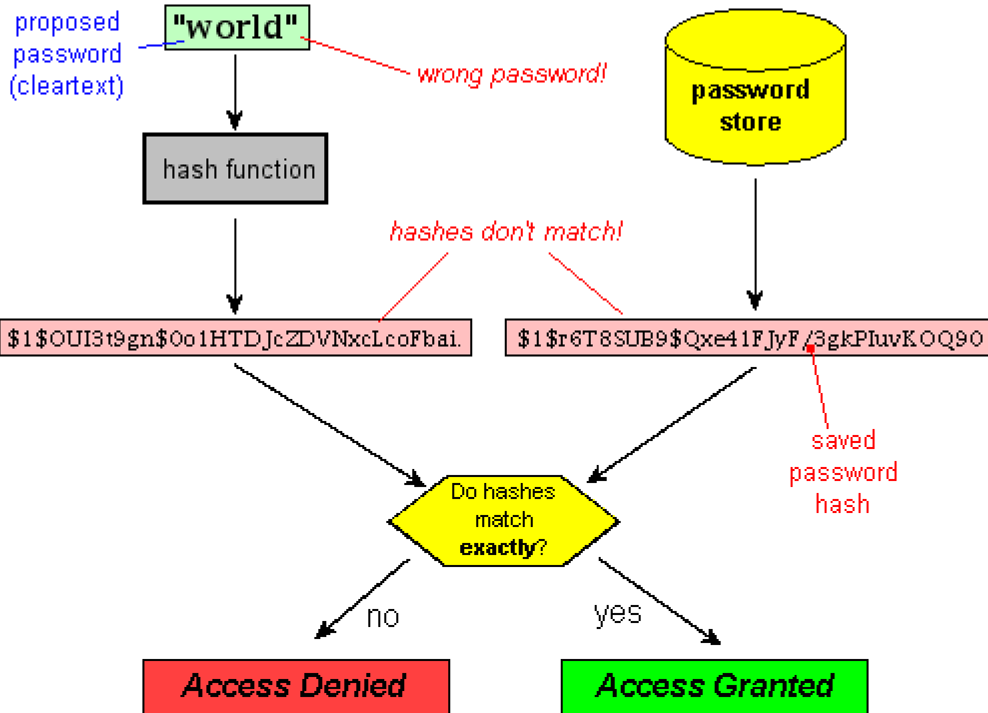
Some companies don't hash passwords, meaning that if a cracker gets into their system they can steal them. This is particularly bad when a user has the same password for multiple sites!

OntarioTech Business & IT

Password Hashing

- Password Hashing: Storage and Authentication

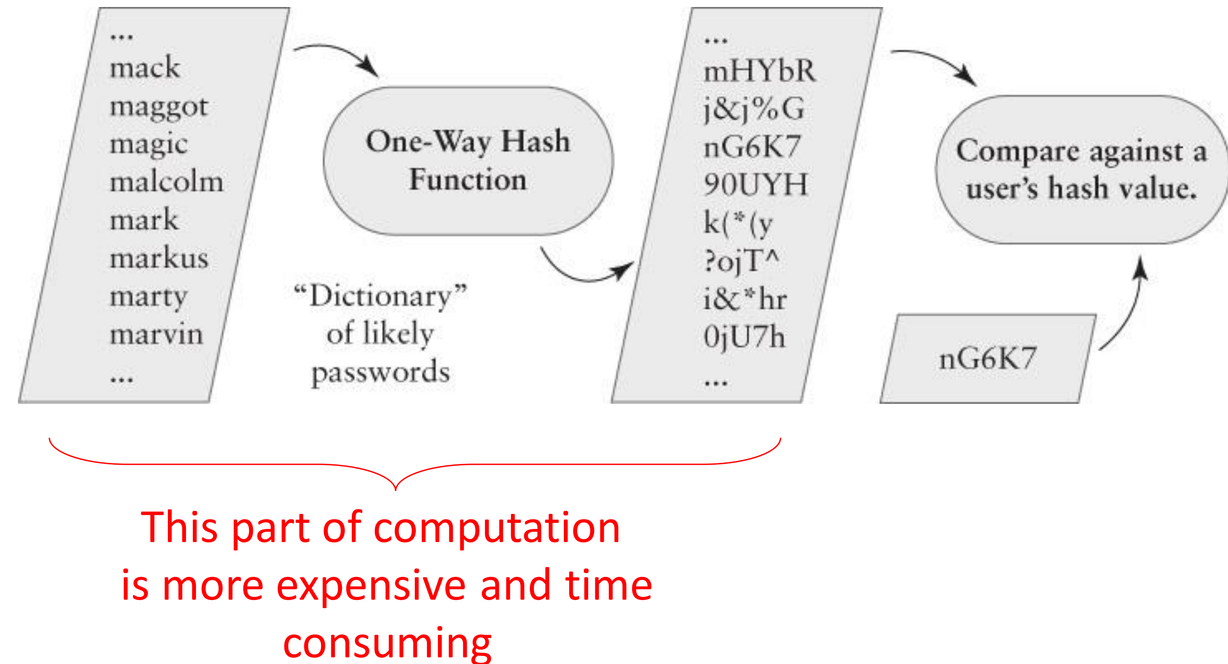
Storing hash instead of password



Testing a password against stored hash

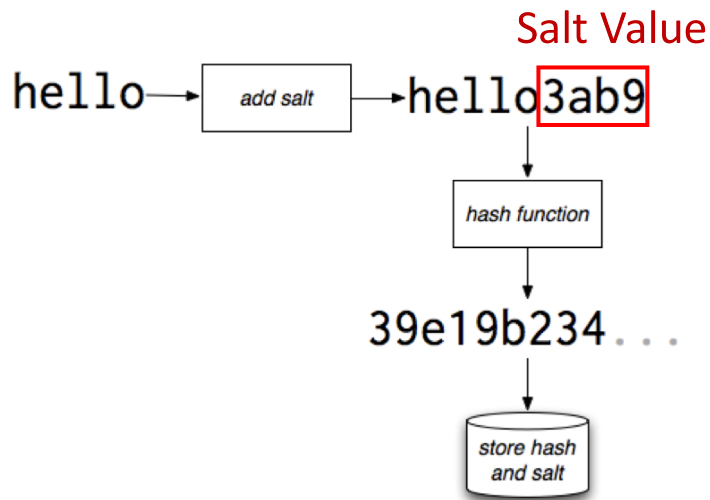
Password Salting

- Dictionary Attack in Password Cracking
  - Assuming: the threat actor got hold of our users' signing credentials stored in the database. Of course, the stored password are all hashed, so she needs to find out what are the corresponding passwords to the stored hashed values [this is known as offline password cracking!]
  - is a type of cyber attack that involves attempting to guess a password or encryption key by systematically trying every word in a dictionary or a list of commonly used passwords. [this is better than brute force in many cases, why? In class discussion]

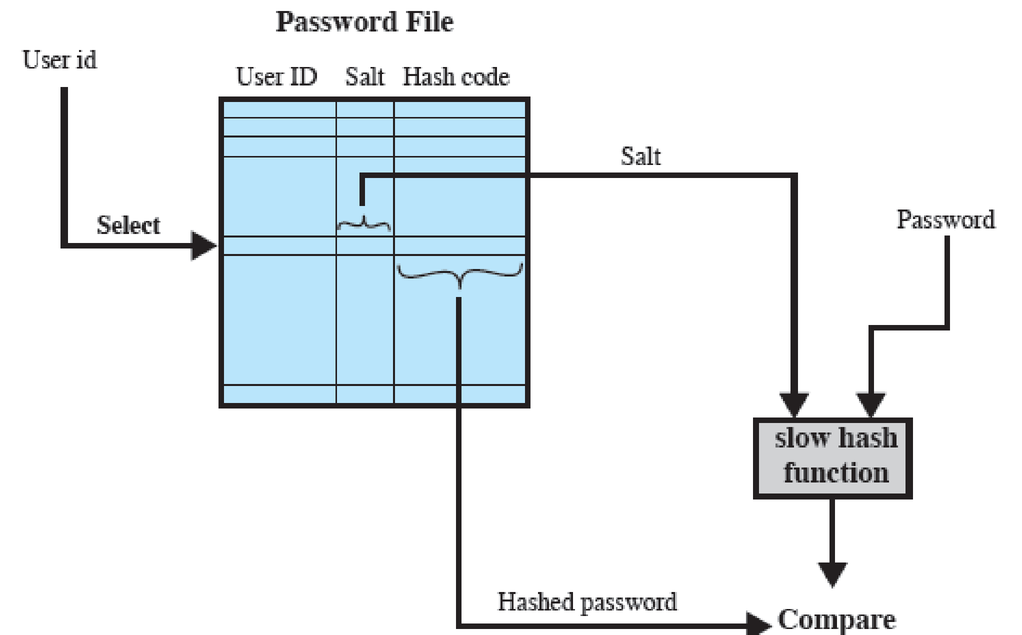


Password Salting – cont'd

- To make a dictionary-based password cracking to go slower – we can use salting.
- Adding a unique random value to each password before hashing:
  - Both the hash and salt are stored.
  - Does not fully prevent against password cracking, but render pre-computed password hash lists useless! **[Why? In-class discussion]**



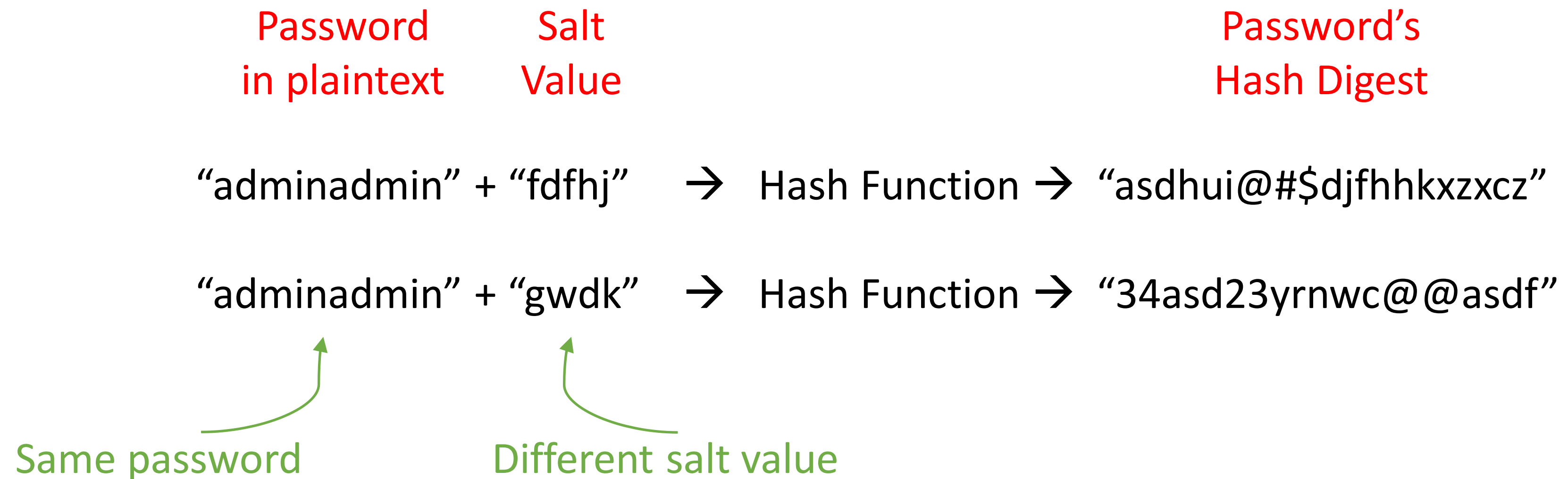

```
import hashlib, uuid
salt = uuid.uuid4().hex
hashed_password = hashlib.sha512(password + salt).hexdigest()
```



(b) Verifying a password

Password Salting

- Benefits – in case of a compromised Password File/Database
  - Render precomputed dictionaries useless!
  - Prevents duplicate passwords from being visible in password file! [Why? In-class discussion]
  - Becomes impossible to find out whether a person has used the same password on multiple systems.



Cryptographic Key Derivation Function (KDF)

- Is a mathematical algorithm that takes an input (such as a password or passphrase) and generates one or more secret cryptographic keys, which can be used to encrypt or decrypt data.
  - The KDF uses the input to generate keys that are difficult for an attacker to guess, even if they know the input.
  - KDFs are commonly used in password-based encryption, where a user's password is used to derive a key for encrypting their data.
  - Also good as a proxy to password hashing for authentication – but not widely exercised.

```
>>> import os
>>> from cryptography.hazmat.primitives import hashes
>>> from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
>>> # Salts should be randomly generated
>>> salt = os.urandom(16)
>>> # derive
>>> kdf = PBKDF2HMAC(
...     algorithm=hashes.SHA256(),
...     length=32,
...     salt=salt,
...     iterations=480000,
... )
>>> key = kdf.derive(b"my great password")
>>> # verify
>>> kdf = PBKDF2HMAC(
...     algorithm=hashes.SHA256(),
...     length=32,
...     salt=salt,
...     iterations=480000,
... )
>>> kdf.verify(b"my great password", key)
```

https://cryptography.io/en/latest/hazmat/primitives/key-derivation-functions/

Encrypted Communication Channel

- Always use TLS to establish network connections
  - TLS provides both encryption and authentication. You need to authenticate TLS certifications to ensure their validity.
    - Threat actors could forge TLS certificates!

```
1   import socket
2   import ssl
3
4   # SET VARIABLES
5   packet, reply = "<packet>SOME_DATA</packet>", ""
6   HOST, PORT = 'XX.XX.XX.XX', 4434
7
8   # CREATE SOCKET
9   sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10  sock.settimeout(10)
11
12  # WRAP SOCKET
13  wrappedSocket = ssl.wrap_socket(sock, ssl_version=ssl.PROTOCOL_TLSv1,
14                                  ciphers="ADH-AES256-SHA")
15
16  # CONNECT AND PRINT REPLY
17  wrappedSocket.connect((HOST, PORT))
18  wrappedSocket.send(packet)
19  print wrappedSocket.recv(1280)
20
21  # CLOSE SOCKET CONNECTION
22  wrappedSocket.close()
```

Encrypted Communication Channel – Cont'd

- Certificate Pinning (CP): is a security technique that involves associating a particular digital certificate with a specific domain or endpoint, and then only accepting that certificate for subsequent connections to that domain or endpoint.
  - This means that even if an attacker were to present a valid certificate from a trusted Certificate Authority (CA), the connection would be rejected if it did not match the pinned certificate.

```
1   import ssl, socket, pprint, os, sys
2
3   # Prepare context, including reference to the server's certificate
4   context = ssl.SSLContext(ssl.PROTOCOL_TLS)
5   context.verify_mode = ssl.CERT_REQUIRED
6   context.check_hostname = False
7   context.load_verify_locations(cafile='server_cert.pem')
8   cnx = context.wrap_socket(socket.socket(socket.AF_INET))
9
10  # Connect and evaluate
11  cnx.connect(('www.hostnametoconnect.com', 8888))
```

  - Certificate pinning is a good security practice because it provides an additional layer of protection against man-in-the-middle (MITM) attacks, where an attacker intercepts and alters communications between two parties. Without certificate pinning, an attacker with a valid certificate could intercept the connection and decrypt the data being transmitted without being detected.
  - certificate pinning requires careful management and updating of pinned certificates, as any changes to the certificate (such as a renewal or revocation) may break the pinned connection. Additionally, it may create some operational challenges when managing a large number of endpoints or domains, as each one requires specific certificates to be pinned.

Be Aware …

- Not all cryptographic schemes in-use today are secure!
  - Symmetric Ciphers
    - DES is not secure, use 3DES instead if you must! But 3DES will not be secure in the near future!
    - Use AES when you can!
  - Asymmetric Ciphers
    - All our asymmetric ciphers are secure for the time being. But, they can be easily broken with a large scale Quantum Computer. Post quantum security is something important to consider in your organization!
  - Cryptographic Hashing Functions
    - Not all cryptographic hash functions are secure: MD2, MD4, MD5, SHA-1
    - Use SHA-256 or SHA-512 instead.