

PYTHON SECURITY



LOGISTICS



Class Hours:

- Instructor will provide class start and end times.
- There will be regular breaks in class.



Telecommunication:

- Turn off or set electronic devices to silent (not vibrate)
- Reading or attending to devices can be distracting to other students
- Try to delay until breaks or after class

Miscellaneous:

- Courseware
- Bathroom
- Fire drills

ISHE DORO

Software Engineer

Experienced programmer in C++, Python & C#

7+ Years Designing & Developing Applications

Microsoft Certified Trainer

ishe@innovationinsoftware.com



INTRODUCE YOURSELF

Time to introduce yourself:

- Name
- What is your role in the organization
- Indicate python programming experience
- Knowledge of security concepts



ABOUT THIS COURSE



In this course you will learn key security concepts and how to build secure applications with Python.

We will look at the main security pillars, and how to protect applications against some of the most common attacks by minimizing vulnerabilities in both your application's code and architecture using well known python security libraries.

COURSE AGENDA

Security & Compliance Concepts

Cryptographic Foundations

Authentication & Authorization

Attack Resistance

LABS

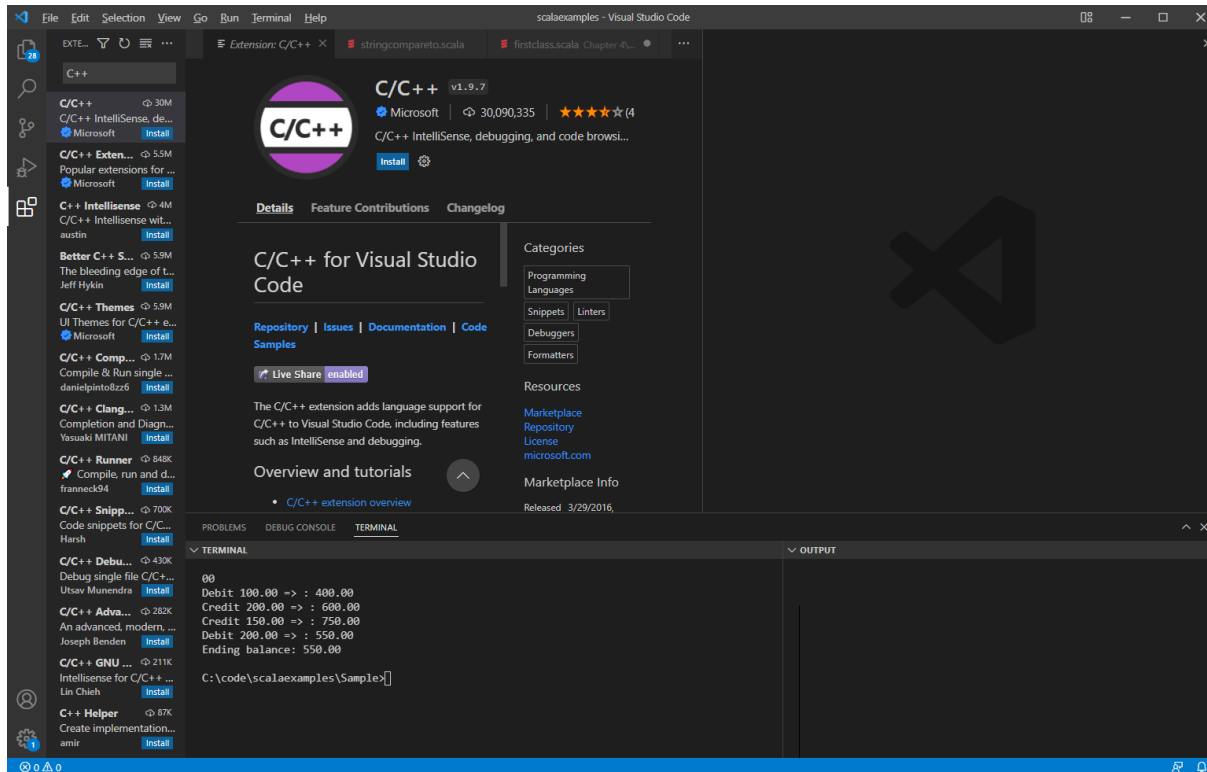


Don't be surprised!

Labs in this course are not overly detailed. Not a check list. General instruction are provided but you must determine the individual steps.

Sometimes new topics are introduced in the labs. But nothing that you cannot research or otherwise find a solution.

VSCODE



VSCode is a free, complete IDE, and available for most environments. VSCode supports a variety of C++ compilers as extensions.

Download VSCode from here:

<https://code.visualstudio.com/download>

RUNTIME ENVIRONMENT

- Python 3.X.X
- Pip or Pipenv
- Visual Studio Code

Lab 1 - Setup



YOUR CLASS!

Yes, this is your class. What does this mean? You define the value.

- What is the most important ingredient of class – your participation!
- Your feedback and questions are always welcomed.
- There is no protocol in class. Speak up anytime!
- We value your comments during and after class.

SECURITY & COMPLIANCE CONCEPTS

THE ORIGIN

As the way we interact & work evolved, information security has become more and more complex.



SHIFTING THE PERIMETER



Front



ATTACK VECTORS



Every attack needs an entry point. This is known as an attack vector.

ATTACK SURFACE



VULNERABILITY



Weaknesses that can be exploited to gain unauthorized access to a computer system.

EXAMPLE ATTACKS

Targets the user

- Reflective cross-site scripting (XSS)
- Social Engineering (e.g., phishing, smishing)
- Cross-site Request Forgery (CSRF)
- Open Redirect Attach

EXAMPLE ATTACKS

Targets the system

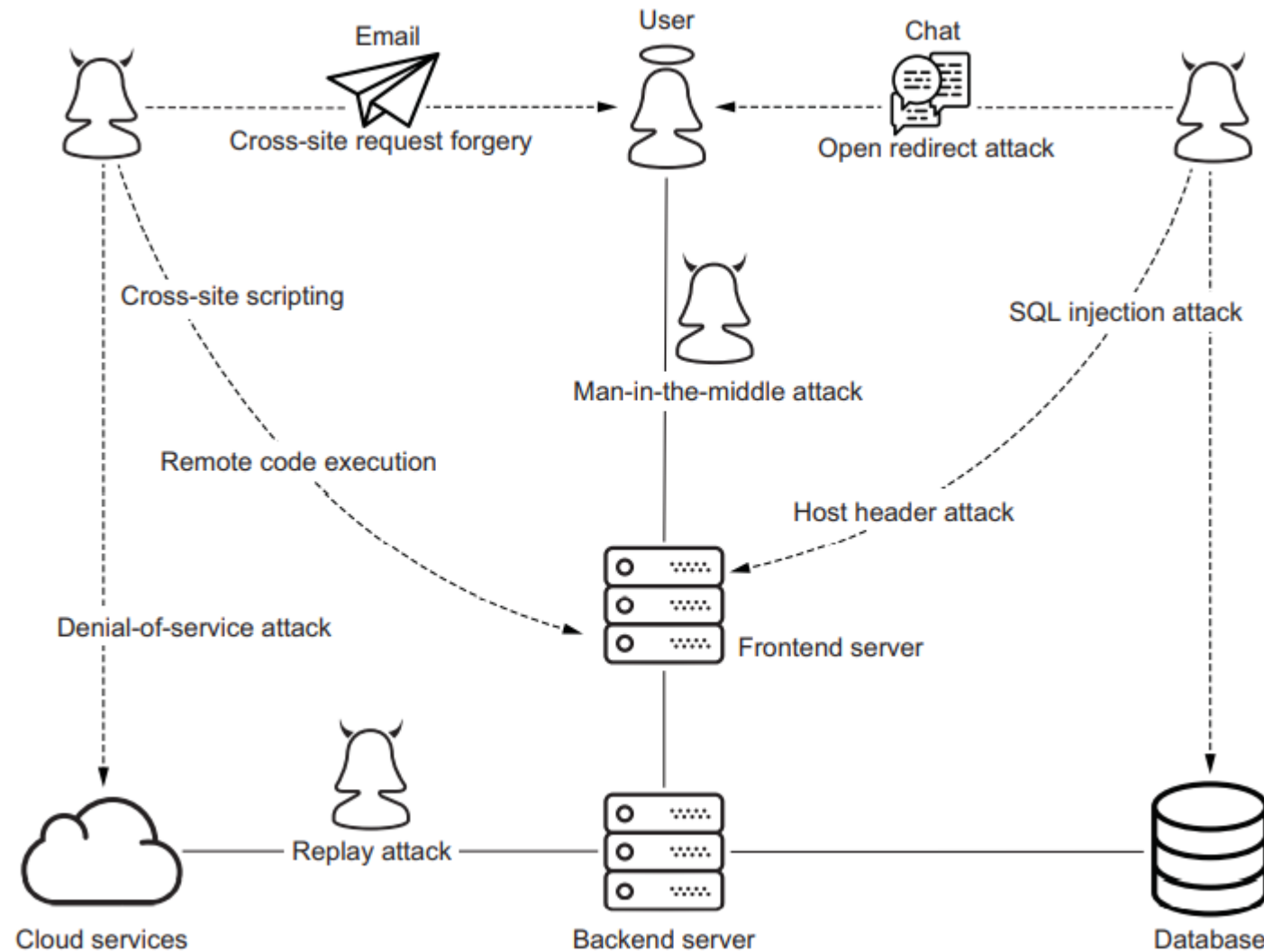
- SQL Injection
- Remote Code Execution
- Host Header Attack
- Denial of Service

EXAMPLE ATTACKS

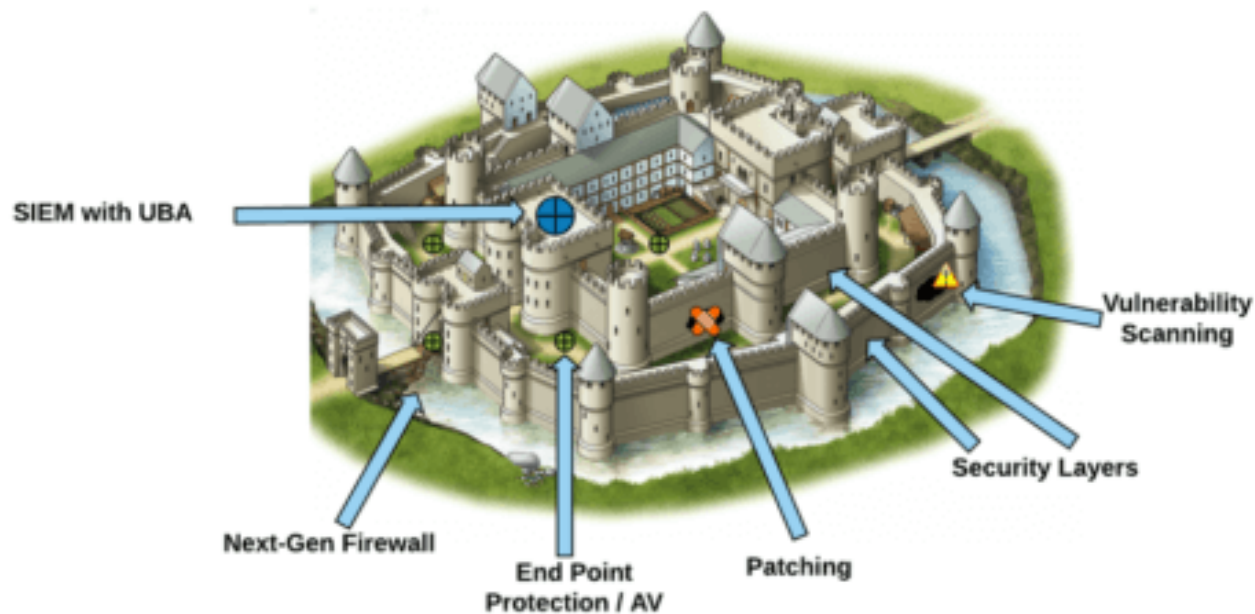
Targets both the
system & the user

- Man-In-The-Middle
- Replay Attack
- Clickjacking

LARGE SCALE ATTACK



DEFENSE IN DEPTH



- Layered approach to security
- Use a series of mechanisms to slow down the advance of an attack.
- Cheaper to add an additional layer than harden a single layer
- Can be used to build security standards

SECURITY STANDARDS

Common
Language/Framework for
building secure applications.

Reusability

- Advanced Encryption Standard (AES)
- Secure Hash Algorithm 2 (SHA-2)
- OAuth 2.0
- Cross Origin Resource Sharing (CORS)

BEST PRACTICES

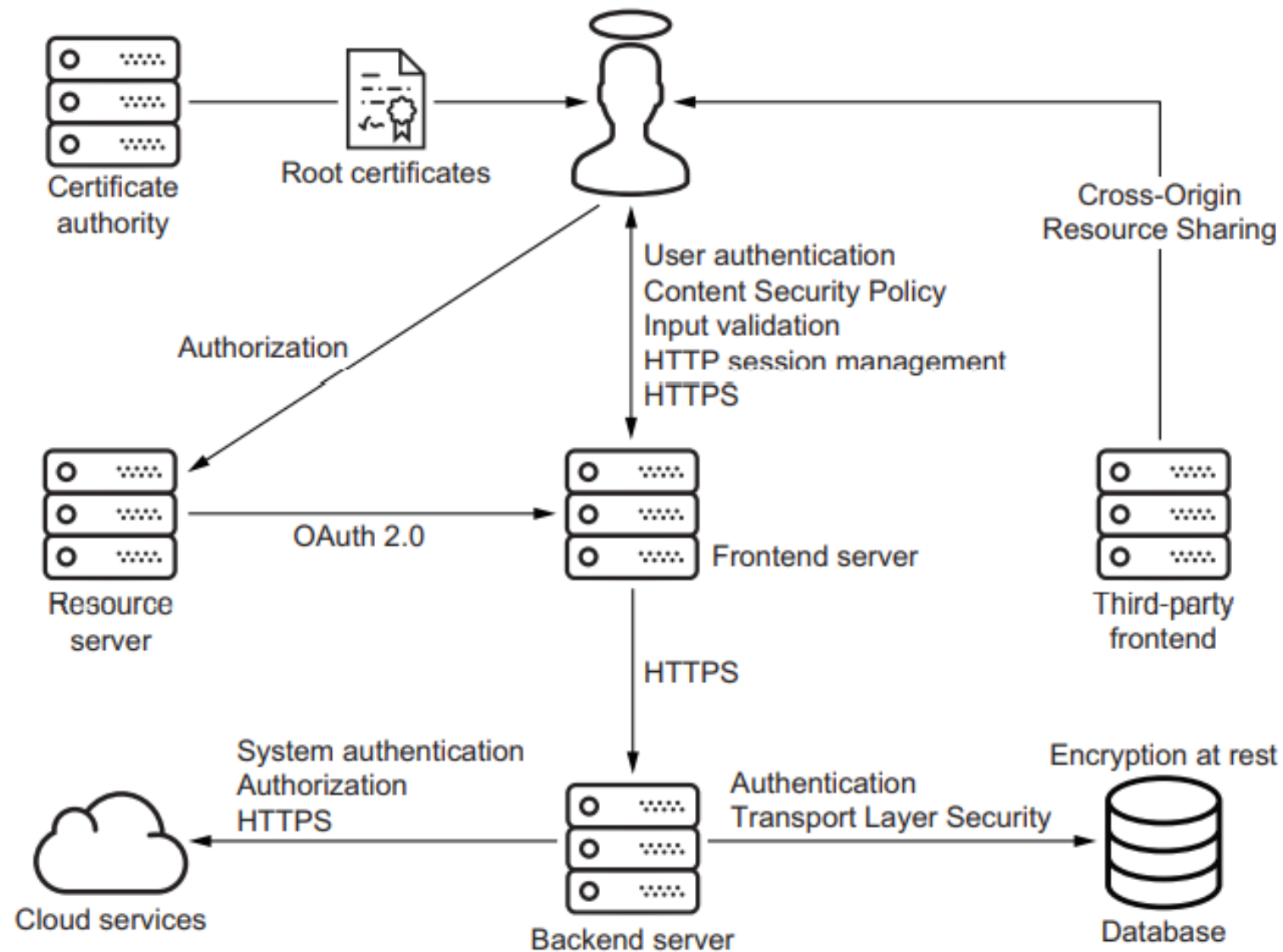
Standardization does not always work

Ways of doing things to complement standards

There is no specification for best-practices

- Defense in depth
- Encryption at rest and in transit
- Don't roll your own crypto
- Principle of Least Privilege
 - OAuth 2.0
 - Cross Origin Resource Sharing (CORS)

DEFENSE IN DEPTH – STANDARDS & BEST PRACTICES



SECURITY FUNDAMENTALS

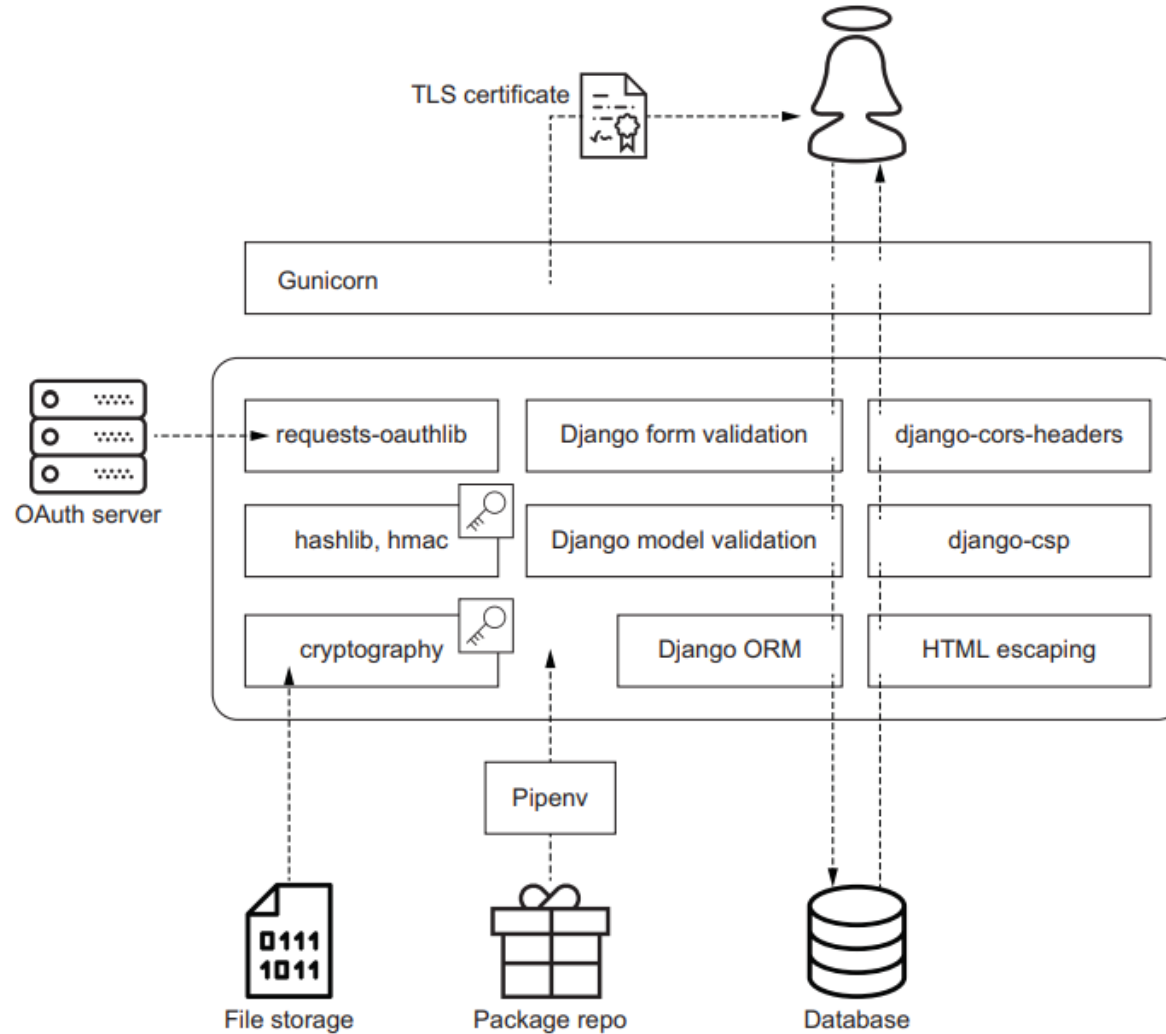
Fundamentals are to best practices and standards

what

Foundations are to Buildings

- Data Integrity
- Authentication
- Authorization
- Non-Repudiation
- Confidentiality

PYTHON TOOLS WE WILL USE



- Is the tool mature
- Is the tool popular

CRYPTOGRAPHIC FOUNDATIONS

OVERVIEW

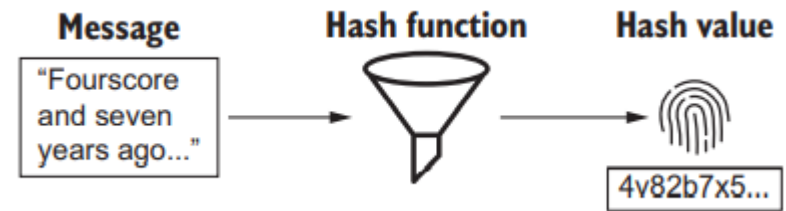
What is cryptography?

LESSONS

- Hashing
- Encryption
- Digital Signatures

HASHING

A hash function maps an input (message) to a fixed-size output known as a hash value.



THE HASH FUNCTION

Python has a built-in **hash** function

```
#HASHING WITH BUILT-IN hash()
```

```
import json
```

```
message = list(range(10))
```

```
hash_value = hash(json.dumps(message))
```

```
print(hash_value)
```

✓ 0.4s

```
4414228952474833747
```

DETERMINISTIC

Repeatable non-random behavior

```
#HASHING WITH BUILT-IN hash()
import json

message = list(range(10))
hash_value = hash(json.dumps(message))
print(hash_value)
```

✓ 0.4s

4414228952474833747

```
message = [x for x in range(10)]
hash_value = hash(json.dumps(message))
print(hash_value)
```

✓ 0.4s

4414228952474833747

FIXED-LENGTH HASH VALUES

The hash-value will always be a fixed-size integer regardless of the message size

```
import json
import numpy as np

grades = list(range(5))
mean = np.mean(grades)
above_average = ["Yes" if x > mean else "No" for x in grades]
message = {"grades": grades, "above average": above_average}
hash_value = hash(json.dumps(message))
print(hash_value)
```

✓ 0.4s

3825951518510382682

FIXED-LENGTH HASH VALUES

The hash-value will always be a fixed-size integer regardless of the message size

```
import json
import numpy as np

grades = list(range(5))
mean = np.mean(grades)
above_average = ["Yes" if x > mean else "No" for x in grades]
message = {"grades": grades, "above average": above_average}
hash_value = hash(json.dumps(message))
print(hash_value)
```

✓ 0.4s

3825951518510382682

AVALANCHE EFFECT

Small differences in messages result in large differences in hash values.

```
import json
import numpy as np

grades = list(range(5))
mean = np.mean(grades)
above_average = ["Yes" if x > mean else "No" for x in grades]
message = {"grades": grades, "above average": above_average}
hash_value = hash(json.dumps(message))
print(hash_value)
```

✓ 0.4s

3825951518510382682

LAB 1.0: PROPERTIES OF THE PYTHON HASH FUNCTION

The `hash()` method only works for immutable objects such as tuple.

Task 1

Write a simple script to demonstrate this property by comparing the output for a mutable and immutable collection of vowels.

Task 2

Create a python class of your choice and override the `__hash__()` method to create a custom hash value for instances of the class.

CRYPTOGRAPHIC HASH FUNCTIONS

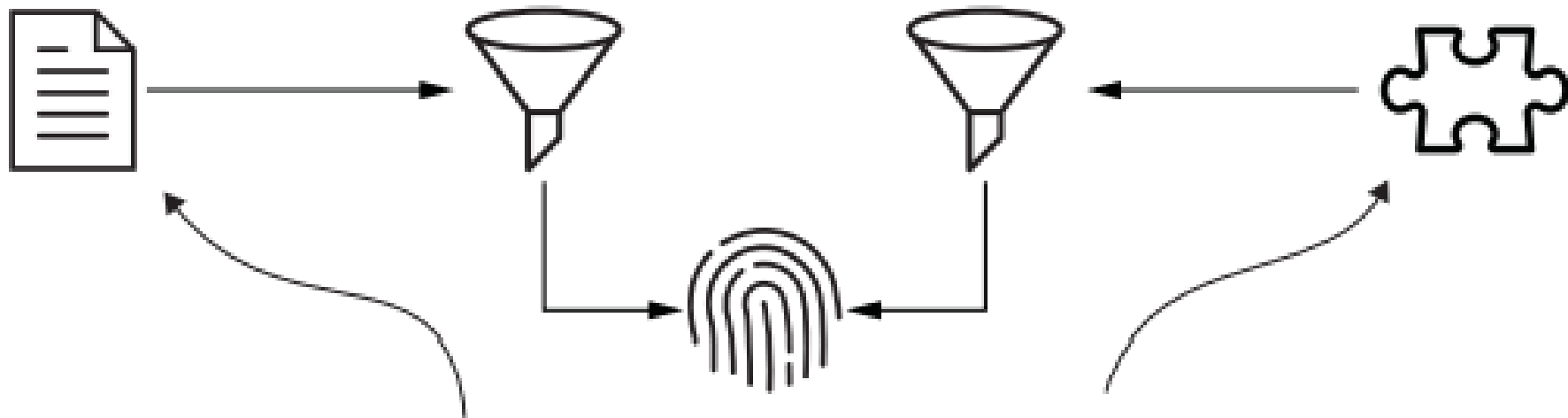
In addition to being deterministic, fixed-length output and avalanche-effect, these functions must be:

- One-way
- Weak collision resistant and
- Strong collision resistant

ONE-WAY FUNCTION

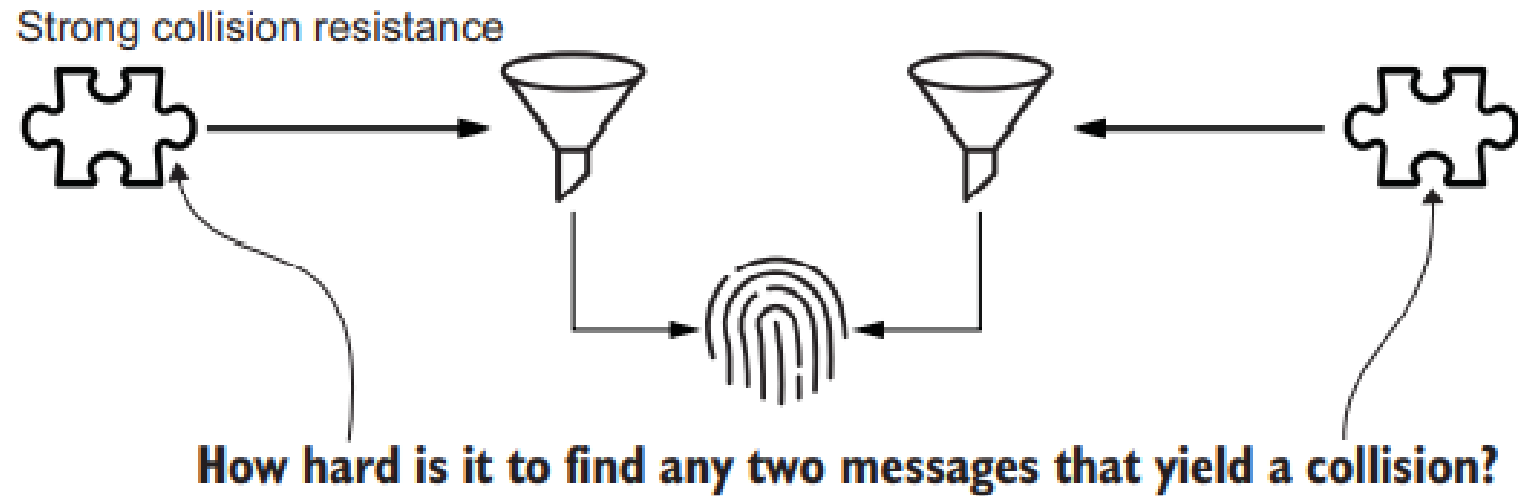
RESERVE-ENGINEERING THE FUNCTION MUST BE INFEASIBLE

WEAK COLLISION RESISTANCE



Given a message, how hard is it to find a different message that yields a collision?

STRONG COLLISION RESISTANCE



HASHLIB MODULE

```
import hashlib

print(hashlib.algorithms_guaranteed)
```

✓ 0.3s Python

{'sha3_256', 'sha256', 'sha3_512', 'shake_128', 'sha224', 'sha512', 'blake2b', 'shake_256', 'md5', 'sha3_224', 'blake2s', 'sha1', 'sha3_384', 'sha384'}

```
print(hashlib.algorithms_available)
```

✓ 0.4s Python

{'sha3_256', 'sha3_512', 'sha224', 'md5-sha1', 'whirlpool', 'sha3_224', 'sha384', 'sha256', 'shake_128', 'sha512', 'ripemd160', 'md5', 'sm3', 'md4', 'blake2b', 'shake_256', 'sha512_224', 'blake2s', 'sha1', 'sha3_384', 'sha512_256', 'mdc2'}

You can view algorithms available to use it:

- `algorithms_guaranteed`
- `algorithms_available`

HASH FUNCTION SAFETY

UNSAFE:

- MD5
- SHA-1

Preserved to maintain backward compatibility.

GUIDELINES:

- Use SHA-256 for general purpose cryptographic hashing
- Use SHA3-256 in high security environments (less support than SHA-256)
- Use BLAKE2 to hash large messages

HASHLIB MODULE

```
import hashlib

print(hashlib.algorithms_guaranteed)
```

✓ 0.3s Python

{'sha3_256', 'sha256', 'sha3_512', 'shake_128', 'sha224', 'sha512', 'blake2b', 'shake_256', 'md5', 'sha3_224', 'blake2s', 'sha1', 'sha3_384', 'sha384'}

```
print(hashlib.algorithms_available)
```

✓ 0.4s Python

{'sha3_256', 'sha3_512', 'sha224', 'md5-sha1', 'whirlpool', 'sha3_224', 'sha384', 'sha256', 'shake_128', 'sha512', 'ripemd160', 'md5', 'sm3', 'md4', 'blake2b', 'shake_256', 'sha512_224', 'blake2s', 'sha1', 'sha3_384', 'sha512_256', 'mdc2'}

You can view algorithms available to use it:

- `algorithms_guaranteed`
- `algorithms_available`

HOW TO CREATE A HASH FUNCTION

```
named = hashlib.sha256()

#OR

from hashlib import sha256

named = sha256()
```

You can create hash functions from hashlib using:

- The named constructor
- The generic constructor

```
generic = hashlib.new("sha256")
```

✓ 0.2s

HOW TO CREATE A HASH FUNCTION

```
named = hashlib.sha256()

#OR

from hashlib import sha256

named = sha256()
```

You can create hash functions from hashlib using:

- The named constructor
- The generic constructor

```
generic = hashlib.new("sha256")
```

✓ 0.2s

DIGEST AND HEXDIGEST METHODS

Each hash function instance has the same API for generating hash values:

- `digest()`
- `hexdigest()`
- `update(message: bytes)`

```
from hashlib import sha256
#Message must be bytes
data = bytes(list(range(5)))
hash_function = sha256(data)
digest = hash_function.digest()
print(digest)
```

✓ 0.4s

```
b'\x08\xbb^]n\xaa\xc1\x04\x9e\xde\x08\x93\xd3\xe0"\xb1\xa4\xd9\xb5\xb4\x8d\xb4\x14\x87\x1fQ\xc9\xcb5(= '
```

CHECKSUM FUNCTIONS

Like hash functions,
checksum functions:

- Accept data but produce checksums
- A checksum, like a hash is also a number
- Can be used to check the integrity of data in transit or at rest

```
import zlib
import pandas as pd
import json

dataset = json.dumps(pd.read_csv('../demo.csv')\
    .to_dict(orient="records"))

data = str.encode(dataset)

checksum = zlib.crc32(data)
compressed = zlib.compress(data)
decompressed = zlib.decompress(compressed)

zlib.crc32(decompressed) == checksum
```

LAB 1.1: USING THE HASHLIB LIBRARY

Task 1

Find two values and show that they hash to the same value using hashlib.md5

Task 2

Given a list of words ['This ', 'is ', 'the ', 'lion ', 'that ', 'leads ', 'the ', 'pride.'] use hash.update() to show how you can iteratively call it and produce a hash value that is exactly the same as hashing the string 'This is the lion that leads the pride.'

LUNCH BREAK: 12:00 TO 13:00

KEYED HASHING

Why keyed hashing:

- Non-keyed hashing can only detect accidental message corruption
- Keyed hashing can detect intentional message modification
- Verifies the integrity and origin of messages (message authenticity)

KEY GENERATION

Keys must be hard to guess, and there are two types of keys:

- Random numbers
- Passphrases

RANDOM NUMBER GENERATION FOR SECURITY

os module

```
import os
os.urandom(16)
✓ 0.4s
b'\x15\xa6\x1c\x15o7~\x8e\x05\x0b\x03=\x03\x1c\xabH'
```

secrets module

```
from secrets import token_bytes, token_hex, token_urlsafe
• print(token_bytes(16))

print(token_hex(16))

print(token_urlsafe(16))
✓ 0.3s
b"\x94- '\x90$[\x9bV\x1bq\x83E<+\x8b\xba"
814c4b96200cff91f5d93b41660d28ce
mff224Bb0NS8rNorIL0hPQ
```


PASSPHRASE GENERATION

Sequence of
random words
from a
dictionary file.

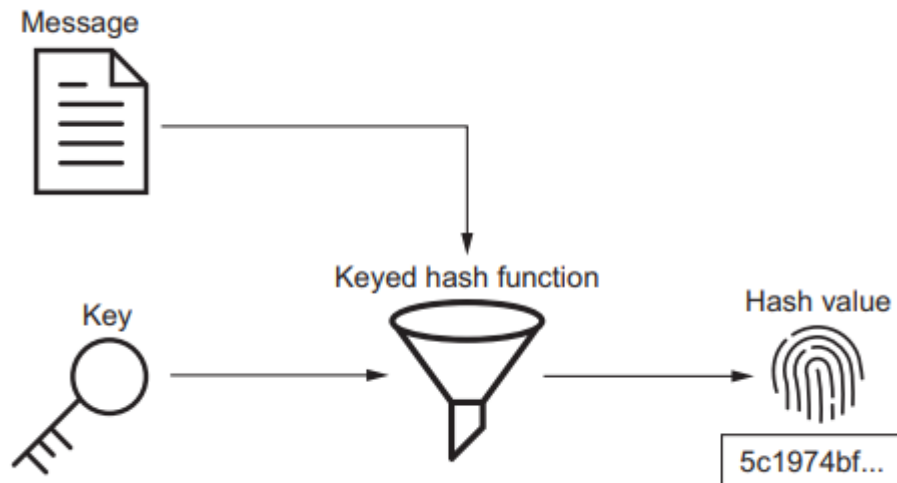
```
import secrets

words = ['data', 'user', 'people', 'california', 'bike', 'tesla']
passphrases = ' '.join(secrets.choice(words) for i in range(3))
print(passphrases)
```

✓ 0.3s

user bike people

BLAKE2



```
• from hashlib import blake2b

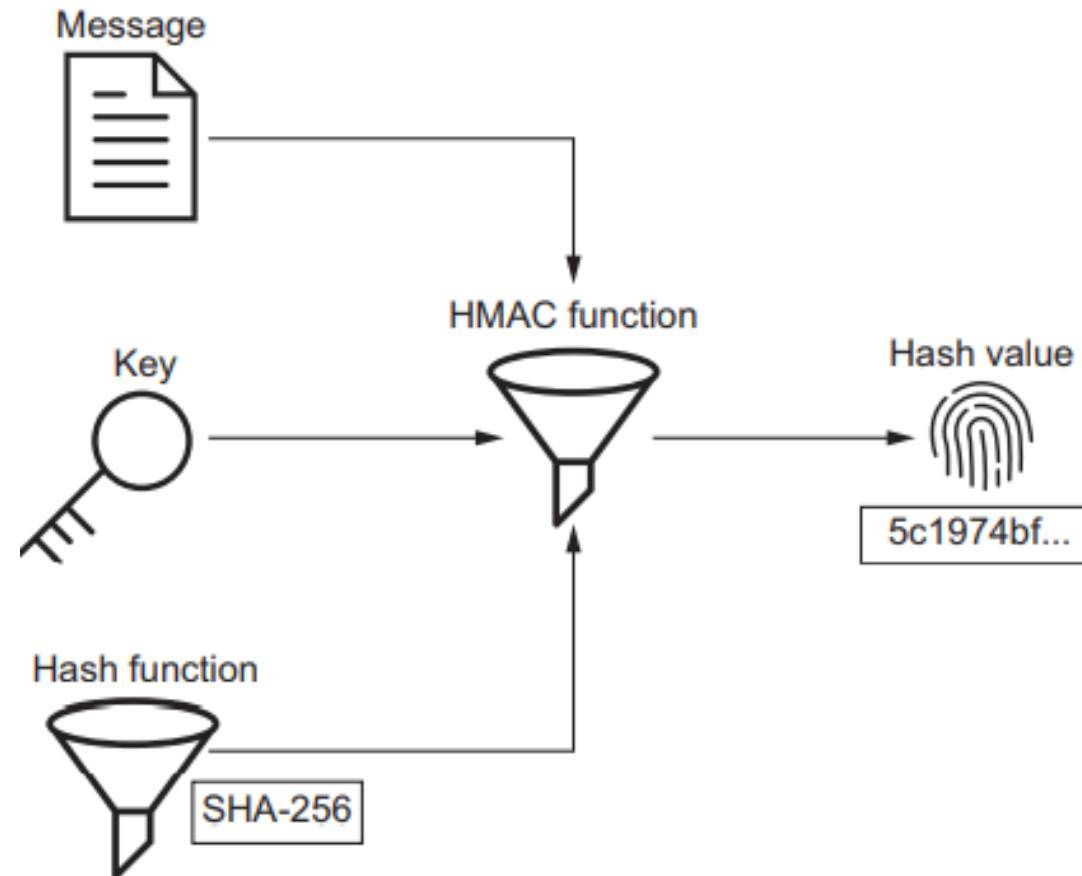
key = os.urandom(16)
message = bytes(list(range(10)))

digest = blake2b(message, key=key).digest()
print(digest)
```

✓ 0.5s

```
b'\x08\xb6\x1a\xa1\x1f\x00\xcb\x17
\xca\xbf\xff\xda\x1f\xc8#\xcc,f}\xeb\xe3\x80=R\
e8\xa3\x04\xdeuq\x15\xd1\x9b'
```

HASH BASED MESSAGE AUTHENTICATION CODE



HMAC MODULE

```
import hmac
import hashlib

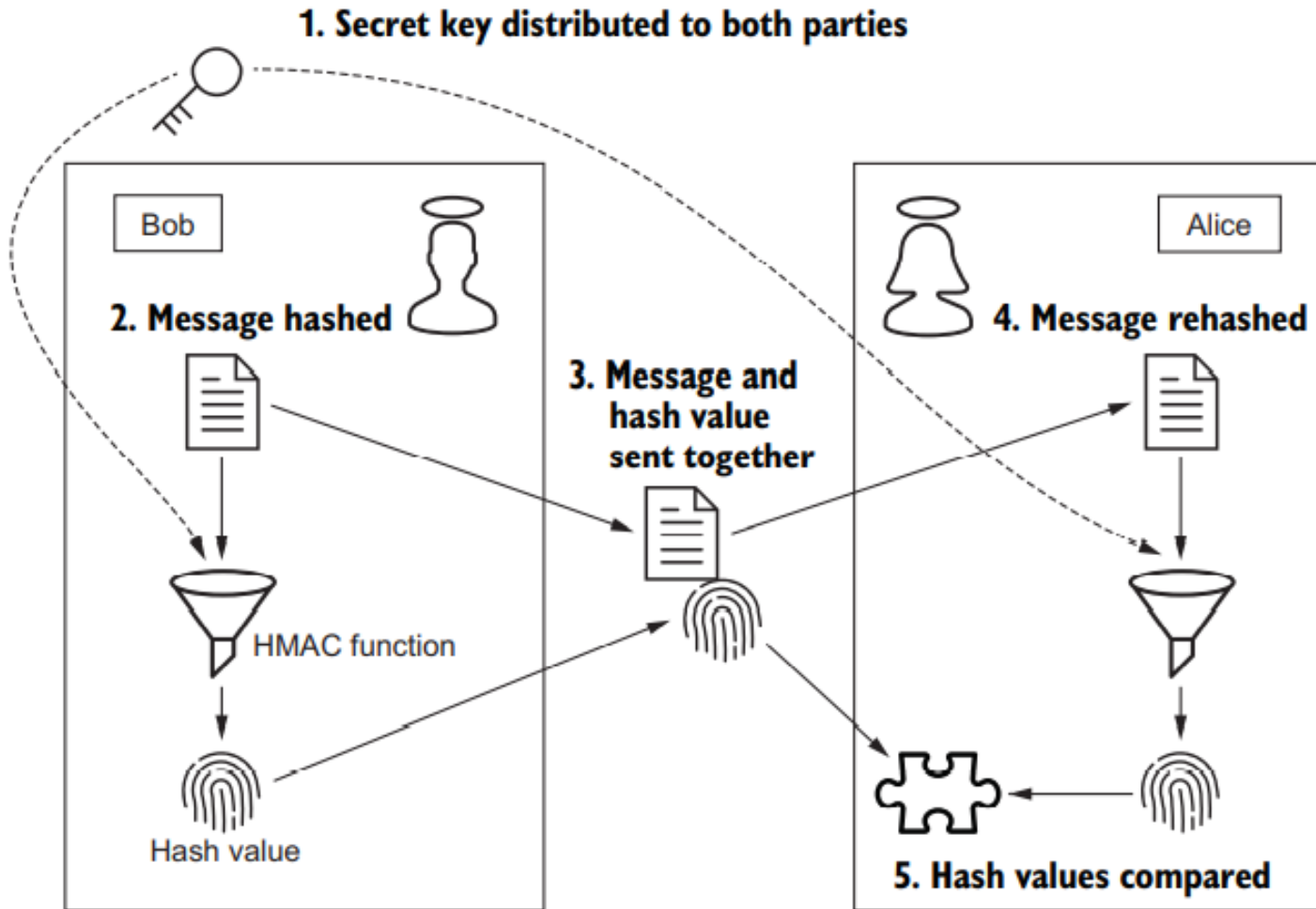
key = os.urandom(16)
message = bytes(list(range(10)))
hmac_func = hmac.new(key, message, hashlib.sha256)

print(hmac_func.digest())

0.4s

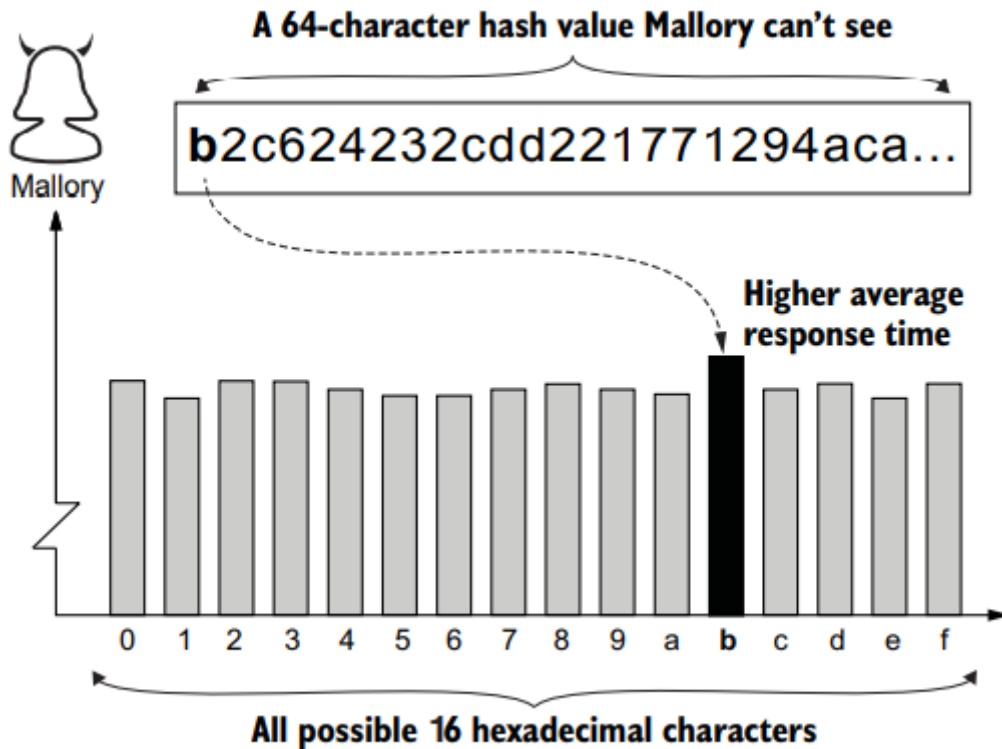
\xab\x10\n\xbe\xf1\xa9\xd9\xe0\x0c/I\xe4\xd2h-\x1fp\x8d\
```

KEY DISTRIBUTION



LAB 2: MESSAGE AUTHENTICATION WITH KEYED HASHING AND HASHLIB

PREVENTING A TIMING ATTACK



```
from hmac import compare_digest
```

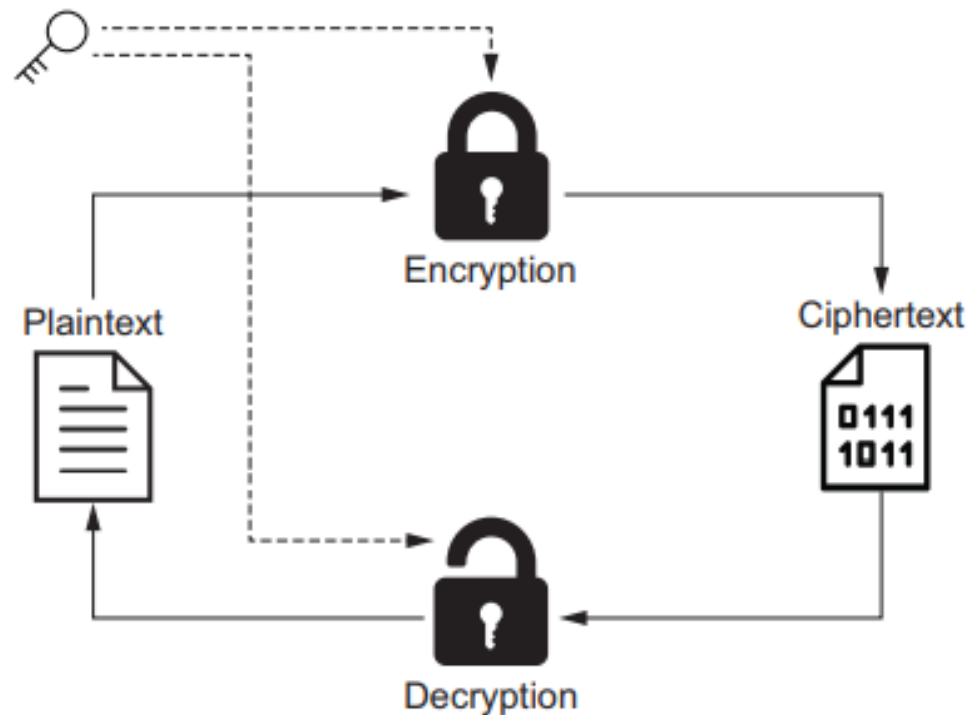
```
compare_digest(b'Digest A',b'Digest B')
```

✓ 0.4s

False

ENCRYPTION

A security method for encoding data from plaintext to ciphertext.



SECURITY FUNDAMENTAL:
Confidentiality

ENCRYPTION & PRIVACY

FORMS OF PRIVACY:

- Individual Privacy – complements the encryption at rest best practice
- Group Privacy – complements the encryption in transit best practice

CRYPTOGRAPHY PACKAGE

Python Encryption Package

- OpenSSL backend
- Hazardous materials layer API
- Recipes layer API

THE FERNET CLASS

```
from cryptography.fernet import Fernet

key = Fernet.generate_key()
print(key)
fernet = Fernet(key)

token = fernet.encrypt(bytes(list(range(10))))

print(token)

list(bytes(fernet.decrypt(token)))
```

✓ 0.4s

```
b'10rrakLBUEbpTt1Nf7N0s7cJv17IAjL9ktBcRBJfK7M='
```

```
b'gAAAAABiYuRGiM41mis0VIx3y9-WXgIT5LF0oHkX-a0-nSdDR-fUoLyd7mMWjb9B_LLonTKSTPa1VPmoL5CINVb1G3IRx2u5EQ=='
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

KEY ROTATION WITH MULTIFERNET

```
from cryptography.fernet import Fernet, MultiFernet

old_key = key #Read from a secure keystore
old_fernet = Fernet(old_key)

new_key = Fernet.generate_key()
new_fernet = Fernet(new_key)

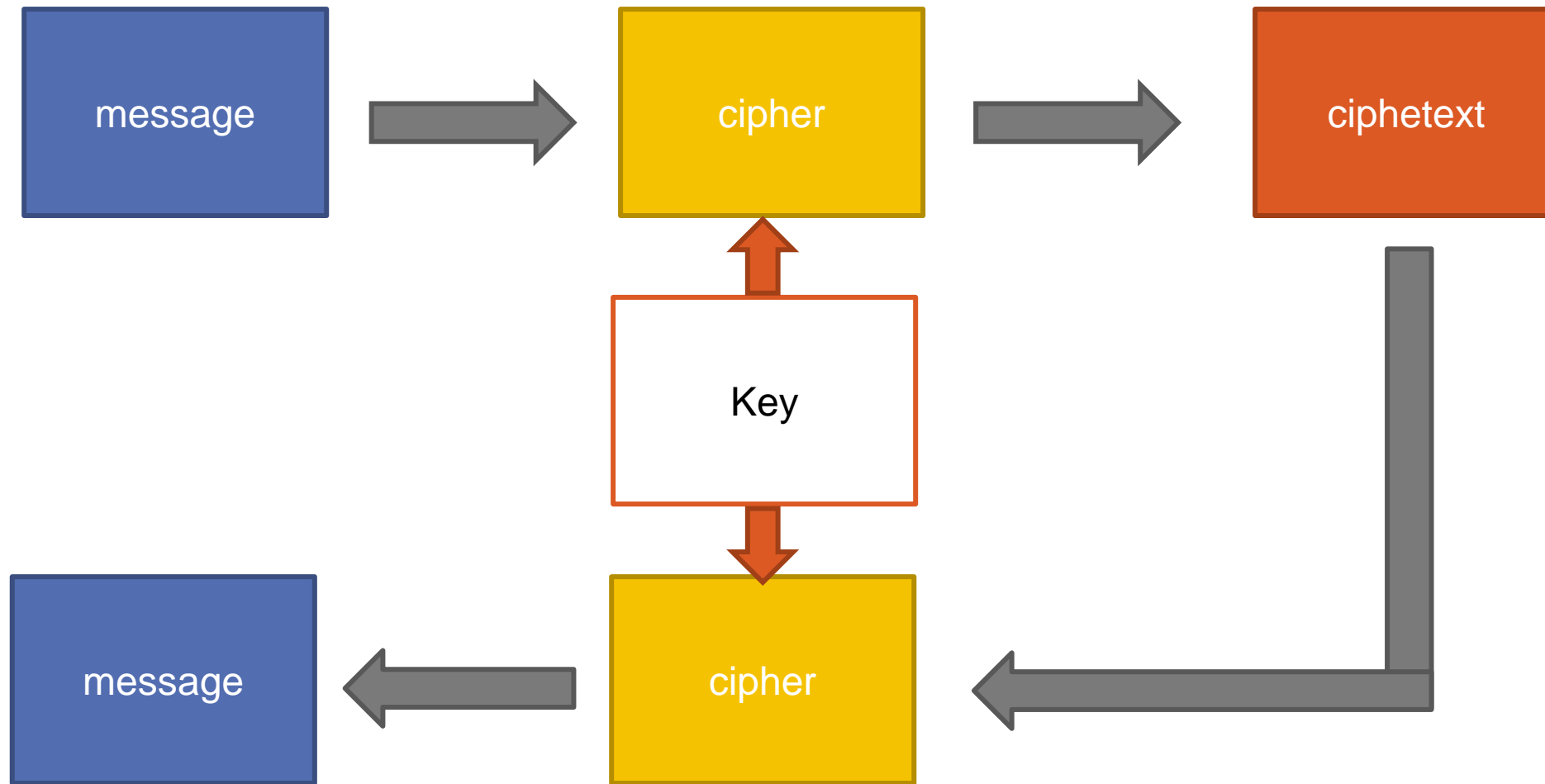
multi_fernet = MultiFernet([new_fernet,old_fernet])
old_token = token
new_token = multi_fernet.rotate(old_token)
```

LAB 2.1: ENCRYPT & DECRYPT FILES WITH FERNET

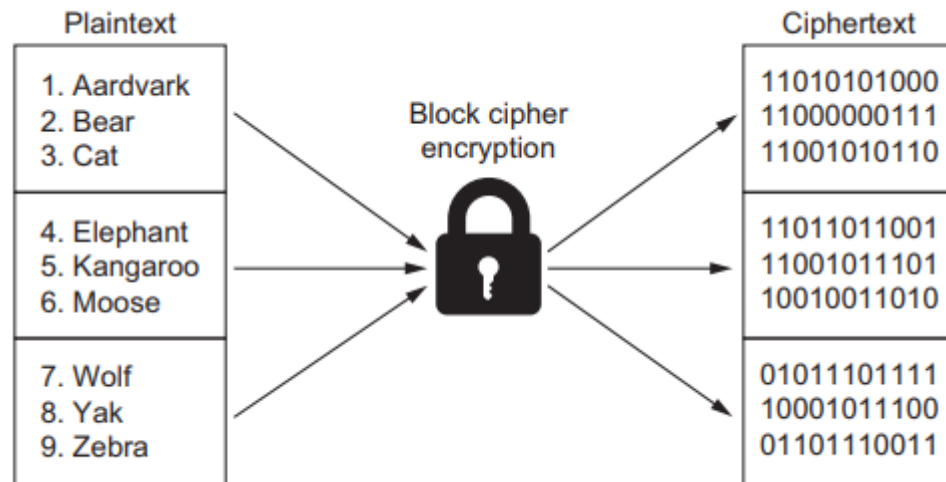
Create a class named Cryptography which has the following methods:

- `create_key(key_path)`: creates a key and saves it to a disk
- `load_key(key_path)`: reads a key from a file
- `encrypt_file(key,file_path)`: encrypts a file and saves the ciphertext to disk.
- `Decrypt_file(key,encrypted_file_path,decrypted_file_path)`: decrypts a file and saves the plain text to disk.

SYMMETRIC ENCRYPTION



BLOCK CIPHERS



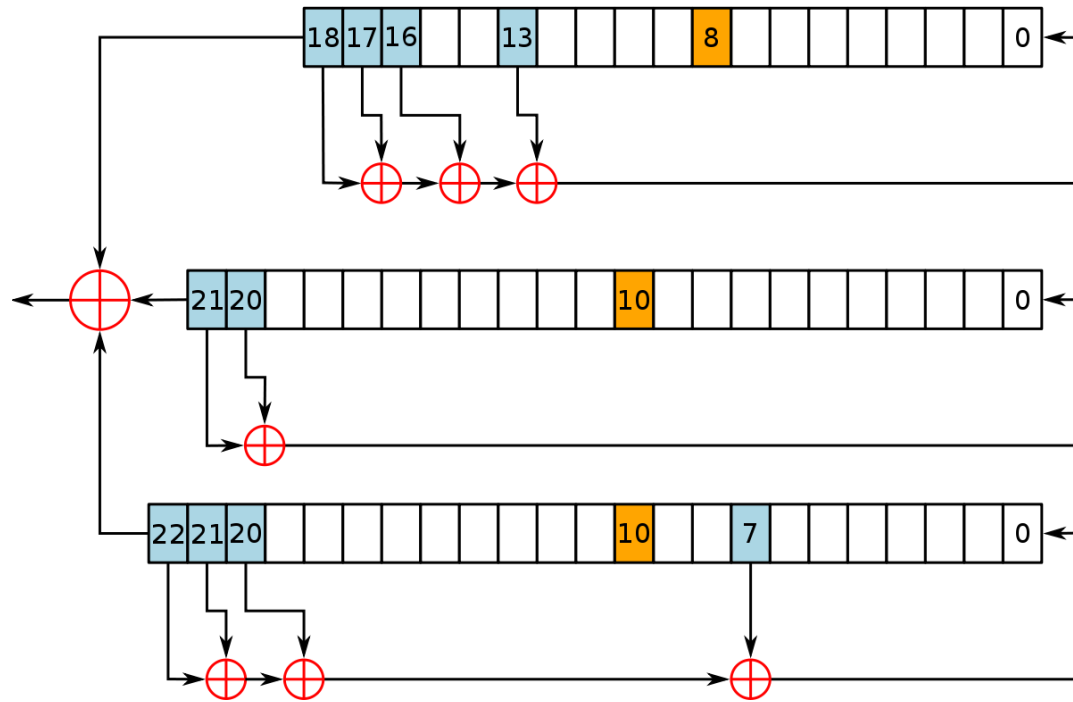
Encrypts N blocks of plaintext and yields N blocks of ciphertext.

Examples of block ciphers include:

- Triple DES
- Blowfish
- Twofish
- Advanced Encryption Standard

Full Stack Python Security – Page 46

STREAM CIPHERS



Processes plaintext as a stream of individual bits.

Examples:

- RC4 (Vulnerable)
- Chacha (Secure & Fast)

Stream ciphertext are more prone to tempering than block ciphertext

ELECTRONIC CODEBOOK MODE (ECM)



Identical Plaintext Blocks

The diagram illustrates the Electronic Codebook (ECB) mode of encryption. It consists of two gray rectangular boxes stacked vertically. The top box is labeled 'Identical Plaintext Blocks' and the bottom box is labeled 'Identical Ciphertext Blocks'. To the right of these boxes, a text block states: 'Never encrypt data with ECB Mode in a production environment.' This visualizes the fundamental flaw of ECB: identical plaintext is always encrypted into the same ciphertext, making patterns in the data visible.

Never encrypt data with ECB Mode in a production environment.

Identical Ciphertext Blocks

CIPHER BLOCK CHAINING MODE

`cryptography.hazmat.primitives.ciphers`

```
✓ import secrets
  import random
  • from cryptography.hazmat.backends import default_backend
    from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

    key = b'1288398383812883'

    iv = secrets.token_bytes(16)
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend = default_backend()).encryptor()

    data = b'Sample plaintext'
    ciphertext = cipher.update(data) + cipher.finalize()
    print(ciphertext)

✓ 0.3s

b''\xfb?P\x103\x89T\xd8R\x96-\x15p,\xf6"
```

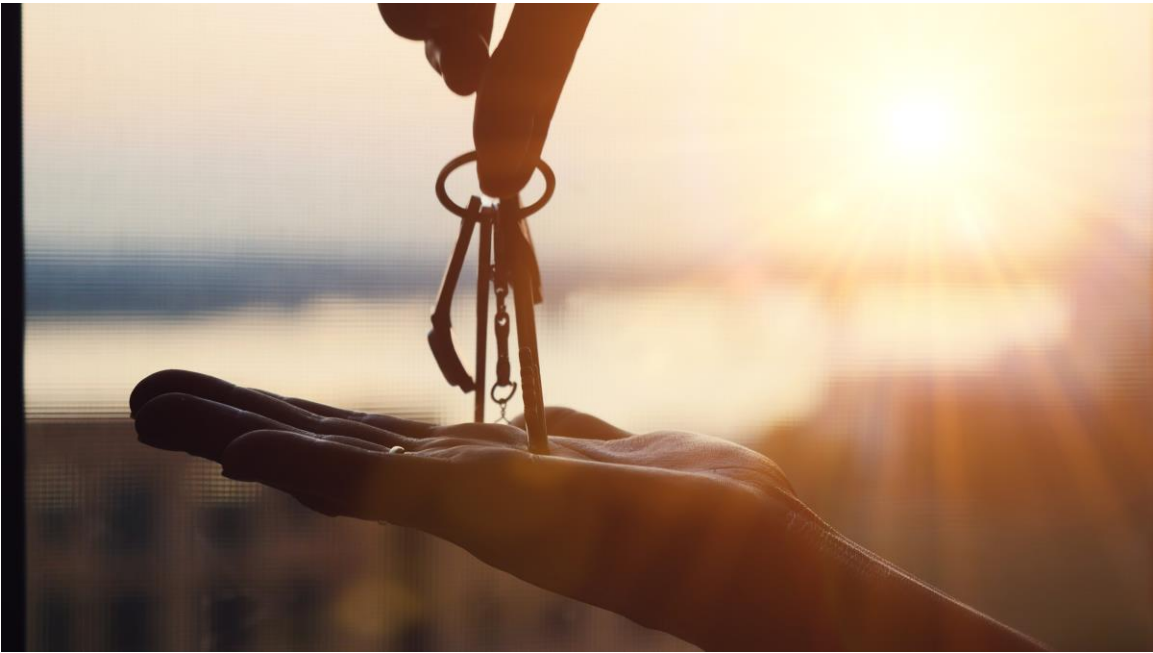
Produces different ciphertexts when encrypting identical plaintexts with the same key.

Example:

Fernet encrypts data with AES in CBC mode.

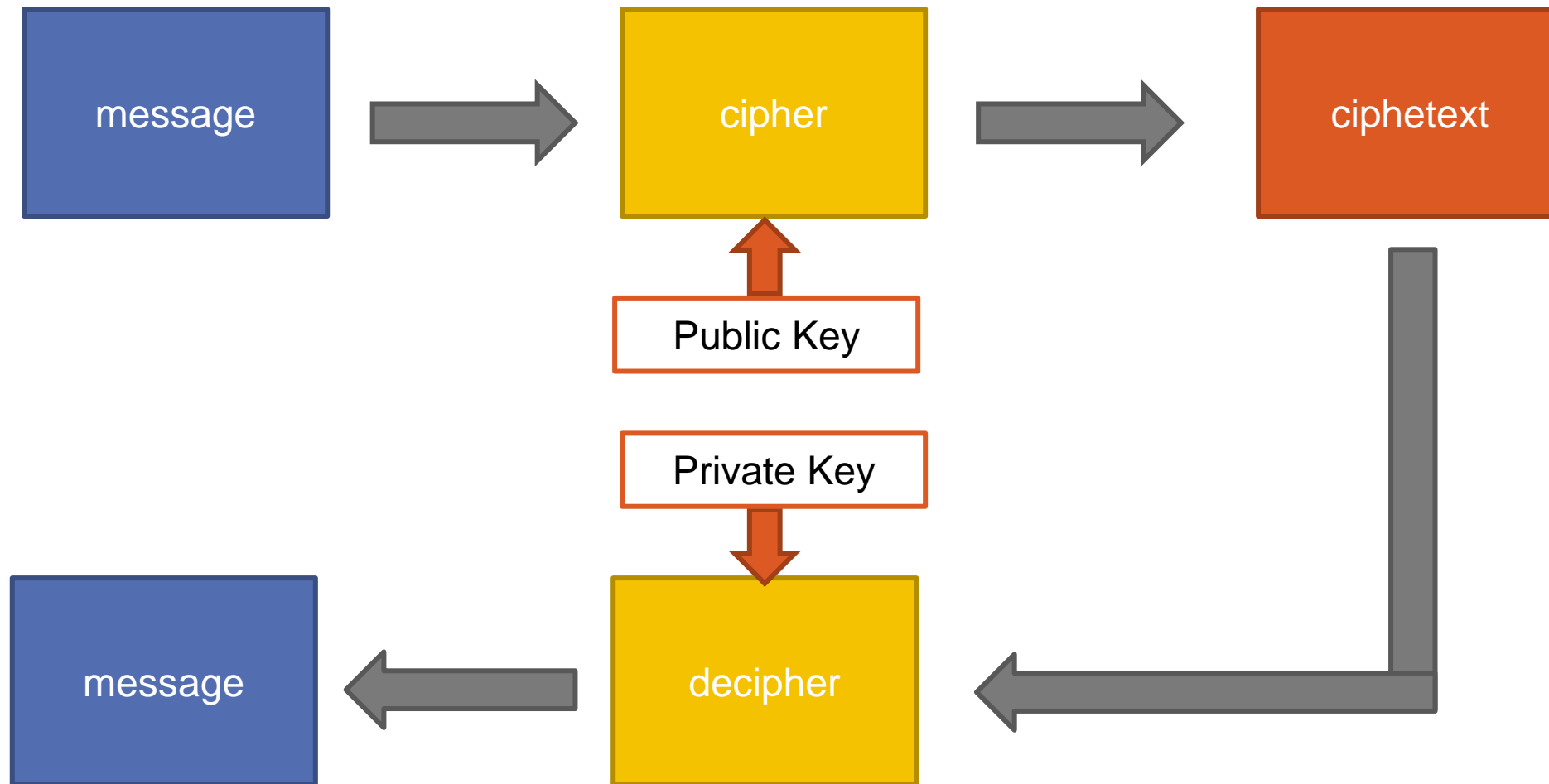
LAB 2.2: SECURE SYMMETRIC KEY ENCRYPTION

KEY-DISTRIBUTION PROBLEM



Exchanging keys becomes a problem when there are more parties involved.

ASYMMETRIC ENCRYPTION



RSA PUBLIC-KEY ENCRYPTION

- Use `rsa` from `cryptography.hazmat.primitives.asymmetric` to generate key-pairs
- Keys should be at least 2048 bits
- Save keys as pem files
- Use public key to encrypt plaintext with OAEP
- Use private key to decrypt ciphertext with OAEP
- The private key file should only be read/written by the owner
- Keys must be stored in a secure key store

DEMO: ASYMMETRIC KEY ENCRYPTION WITH RSA

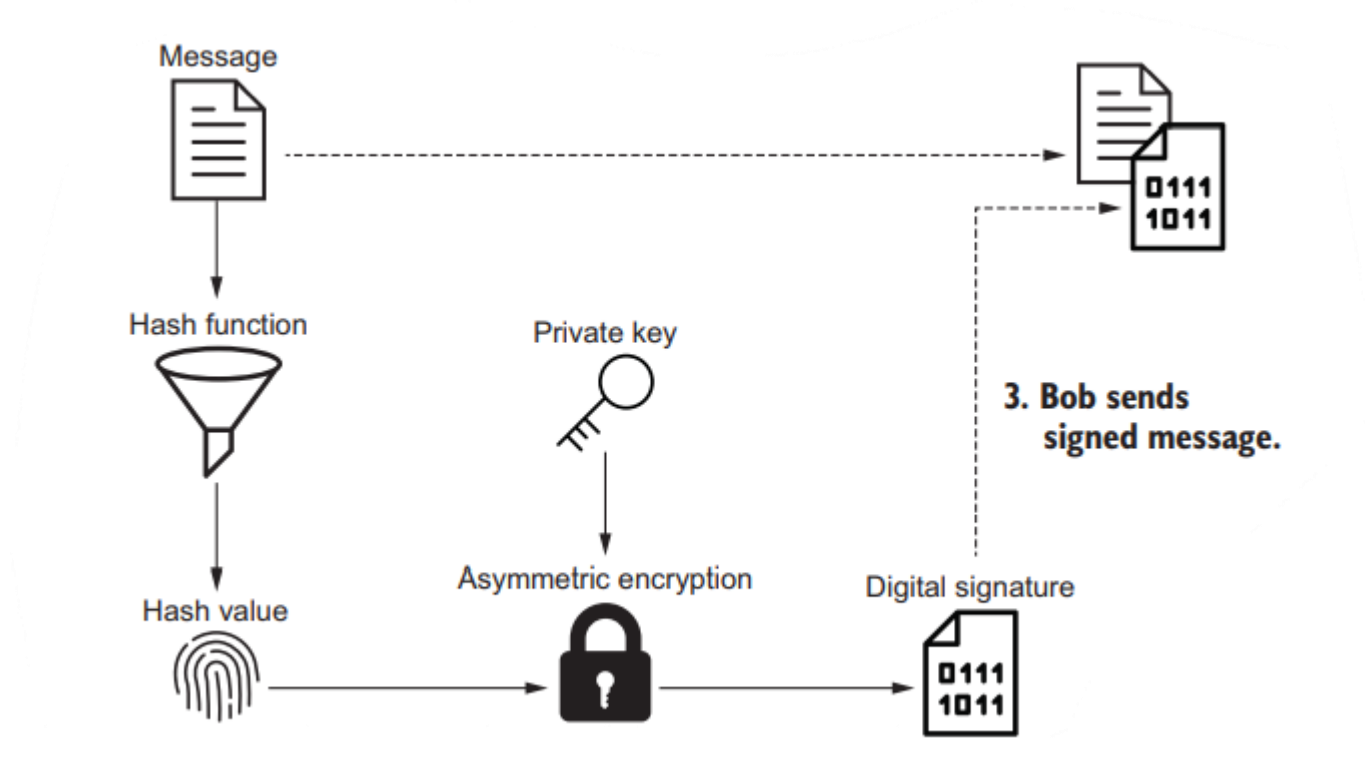
LAB 2.3: ASYMMETRIC KEY ENCRYPTION WITH RSA

In this lab exercise, you will create an API that allows a client to get a public-key from a server and use it to encrypt a message.

The client then sends the ciphertext as a hexademical string to the server which decrypts the message and saves it to a file.

DIGITAL SIGNATURES

- Prevents non-repudiation
- Identifies the sender
- Verifies if message integrity
- Is unique to the signer
- Can be used to legally bind the signer to a contract
- Difficult to forge



RSA DIGITAL SIGNATURES

Sign message with:

- RSA private key
- SHA256
- Probabilistic Signature Scheme
- RSAPrivateKey.sign method

```
import json
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes

message = b'Test message'
padding_config = padding.PSS(mgf=padding.MGF1(hashes.SHA256()),
                              salt_length=padding.PSS.MAX_LENGTH)

#Sign with RSA private key and hash with SHA256
private_key = load_pem_private_key()

signature = private_key.sign(message,
                              padding_config, hashes.SHA256())

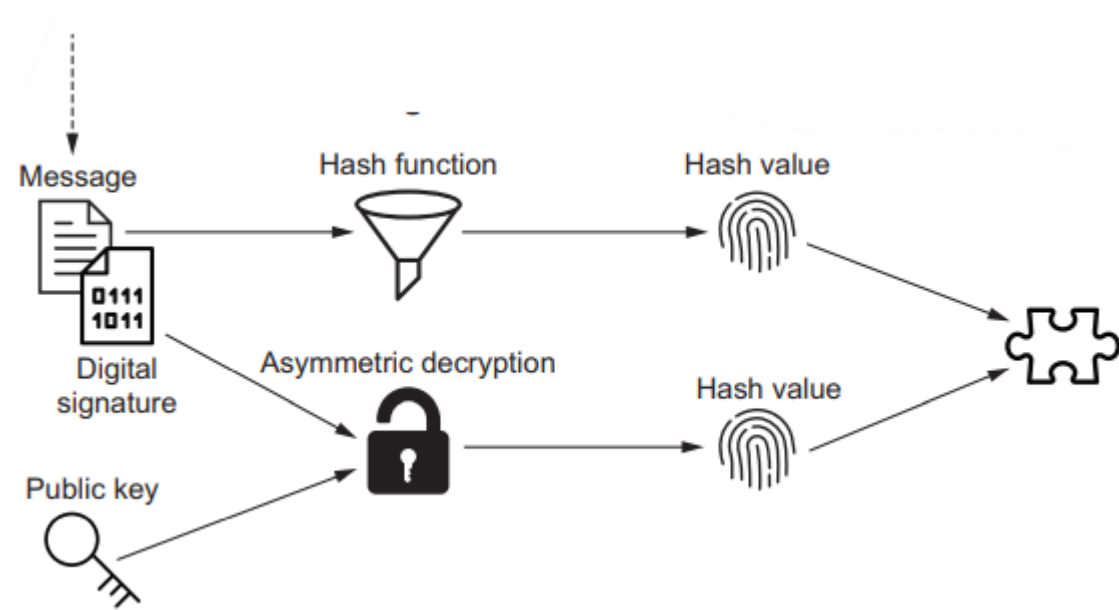
signed_message = {'message':list(message),
                  'signature':list(signature)}

print(json.dumps(signed_message))
```

VERIFYING DIGITAL SIGNATURES

Verify message with:

- RSAPublicKey.verify method
- SHA256
- Probabilistic Signature Scheme



ELLIPTIC-CURVE DIGITAL SIGNATURES

Differences from RSA Digital Signatures

- High performance
- Uses elliptic-curve key pairs to sign data and verify signatures
- Does not encrypt data asymmetrically
- 256-bit key as strong as RSA 3072-bit key

LAB 2.4: NONREPUDIATION WITH DIGITAL SIGNATURES

In this lab exercise, you will make improvements to the API you created in Lab 2.3 to cater for nonrepudiation.

SUMMARY

- Hashing ensures data integrity and data authentication (hash(), hashlib module – sha256())
- Encryption ensures confidentiality (Fernet, RSA)
- Digital signatures ensure nonrepudiation (RSA, Elliptic-curve)

TRANSPORT LAYER SECURITY (TLS)

- Secure networking protocol.
- Applies data integrity, authentication, confidentiality and nonrepudiation.

SSL VS TLS VS HTTPS



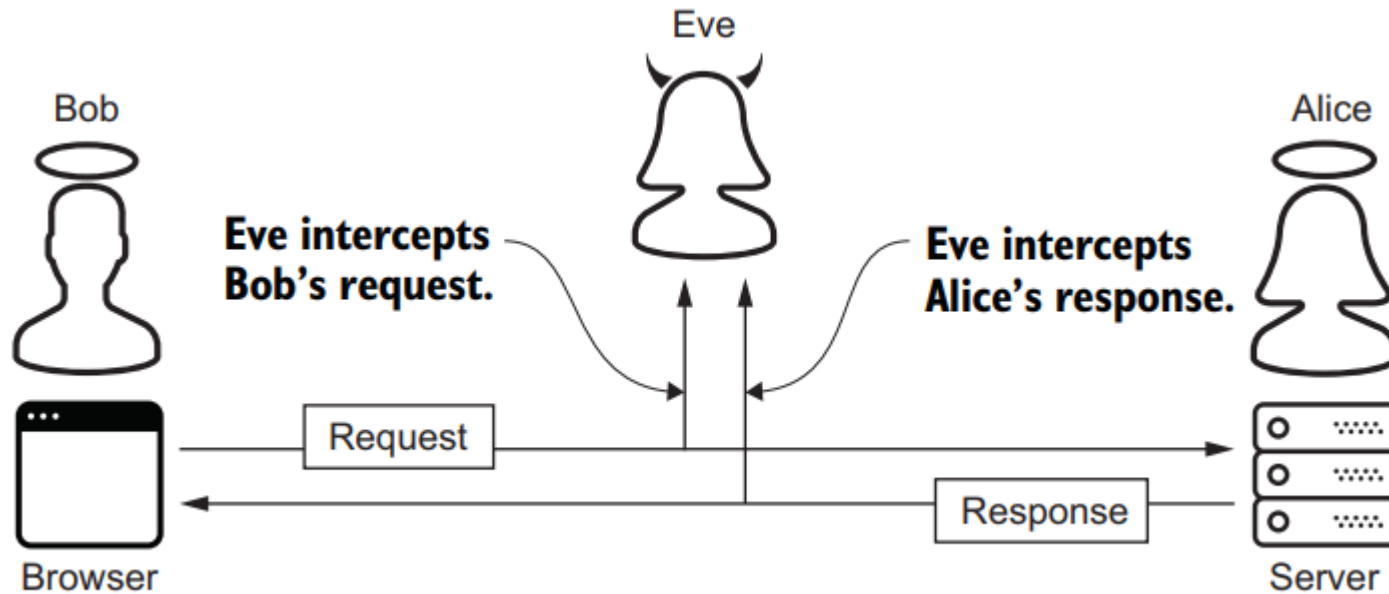
- SSL is a retired older version of TLS
- HTTPS is HTTP over TLS

MAN IN THE MIDDLE ATTACK



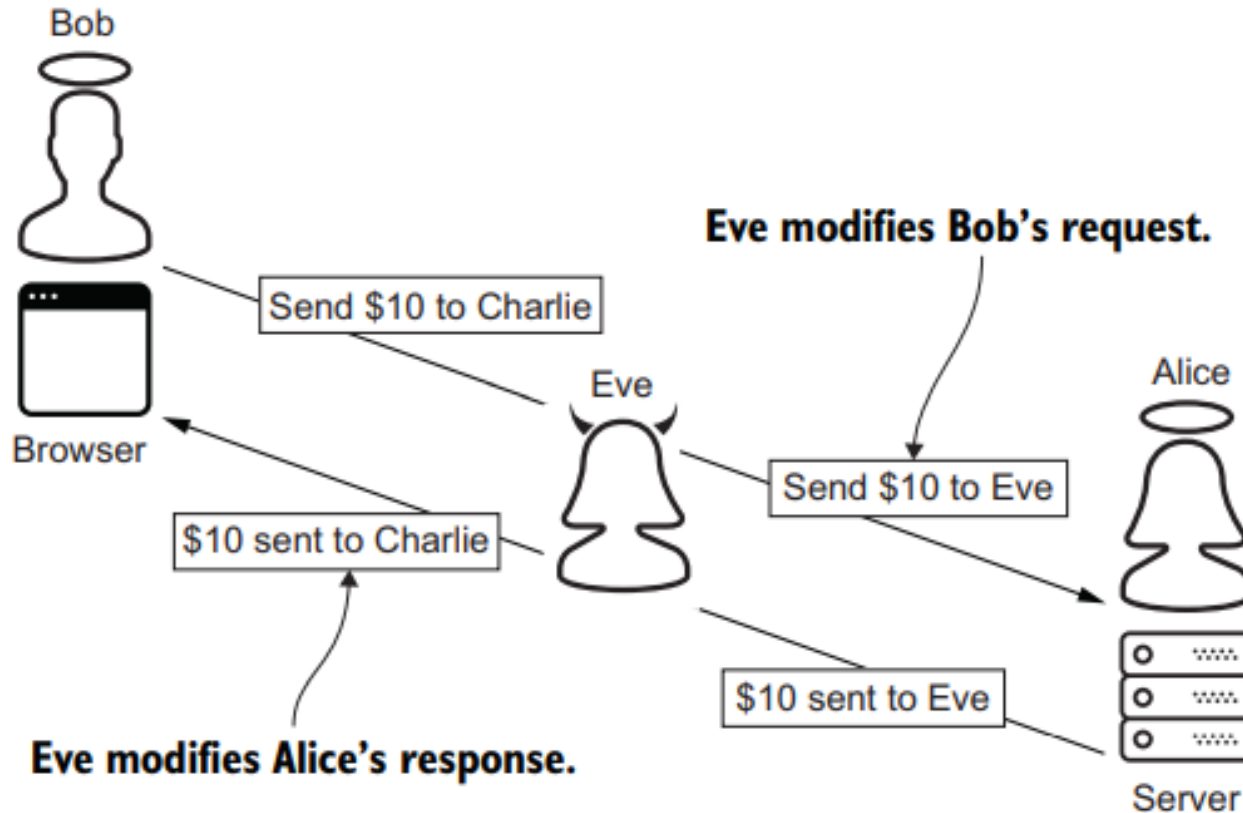
- Attacker takes a position between two vulnerable parties.
- Passive MITM
- Active MITM
- TLS provides confidentiality for data in transit.

PASSIVE MITM



- An attacker intercepts messages whilst in transit.
- TLS provides confidentiality

ACTIVE MITM

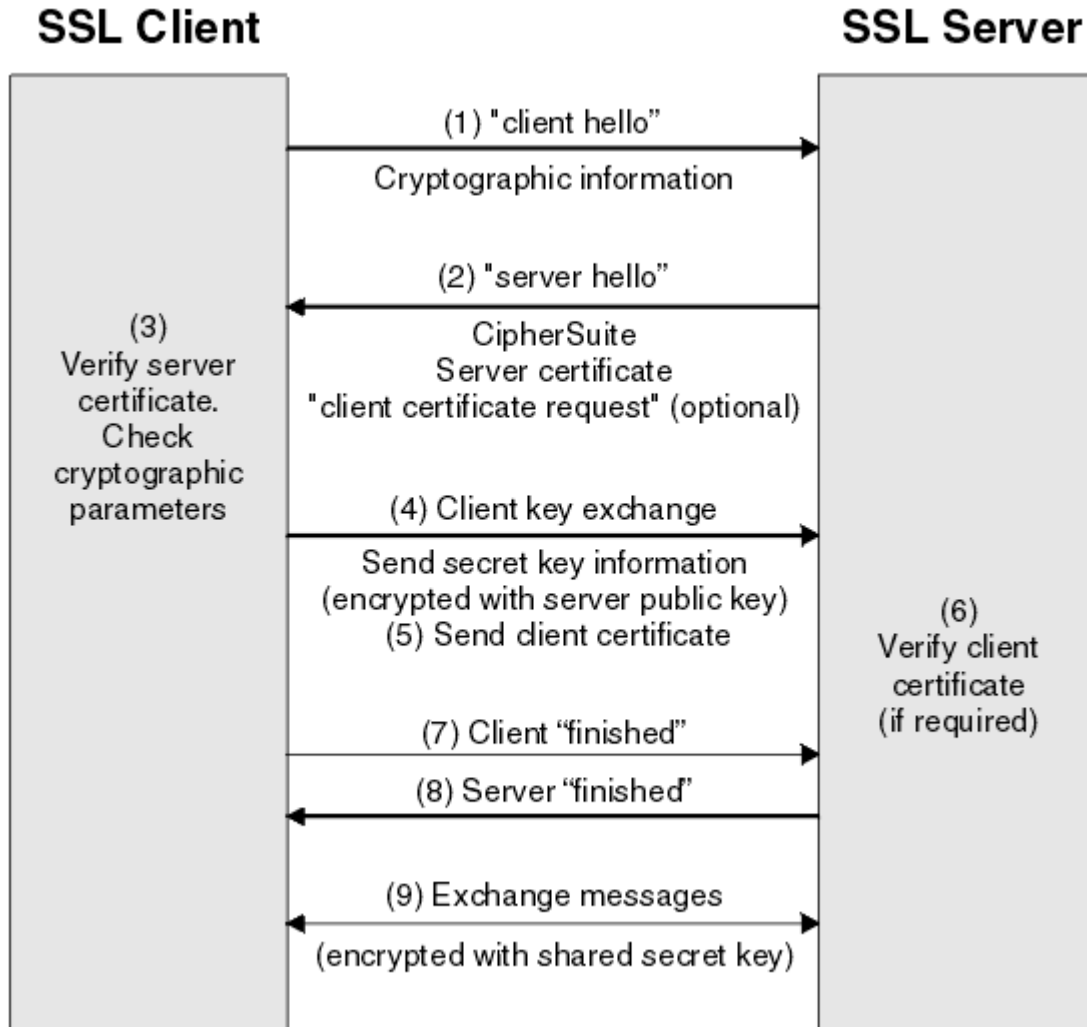


- TLS cannot provide network device security
- TLS provide message authentication

If you are keen, you can learn how a MITM attack is built in python here:

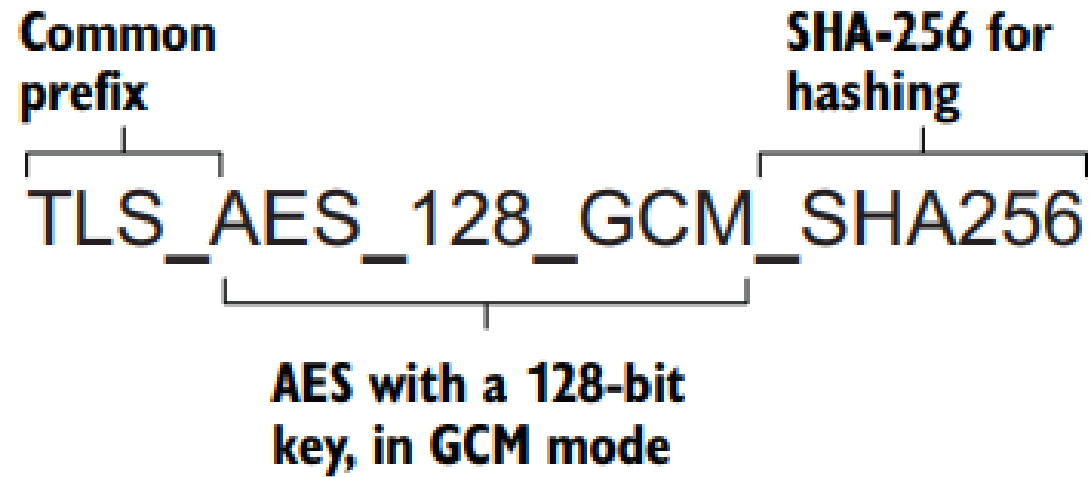
<https://null-byte.wonderhowto.com/how-to/build-man-middle-tool-with-scapy-and-python-0163525/>

TLS HANDSHAKE



- Point-Point client/server protocol
- Cipher suite negotiation
- Key exchange
- Server authentication

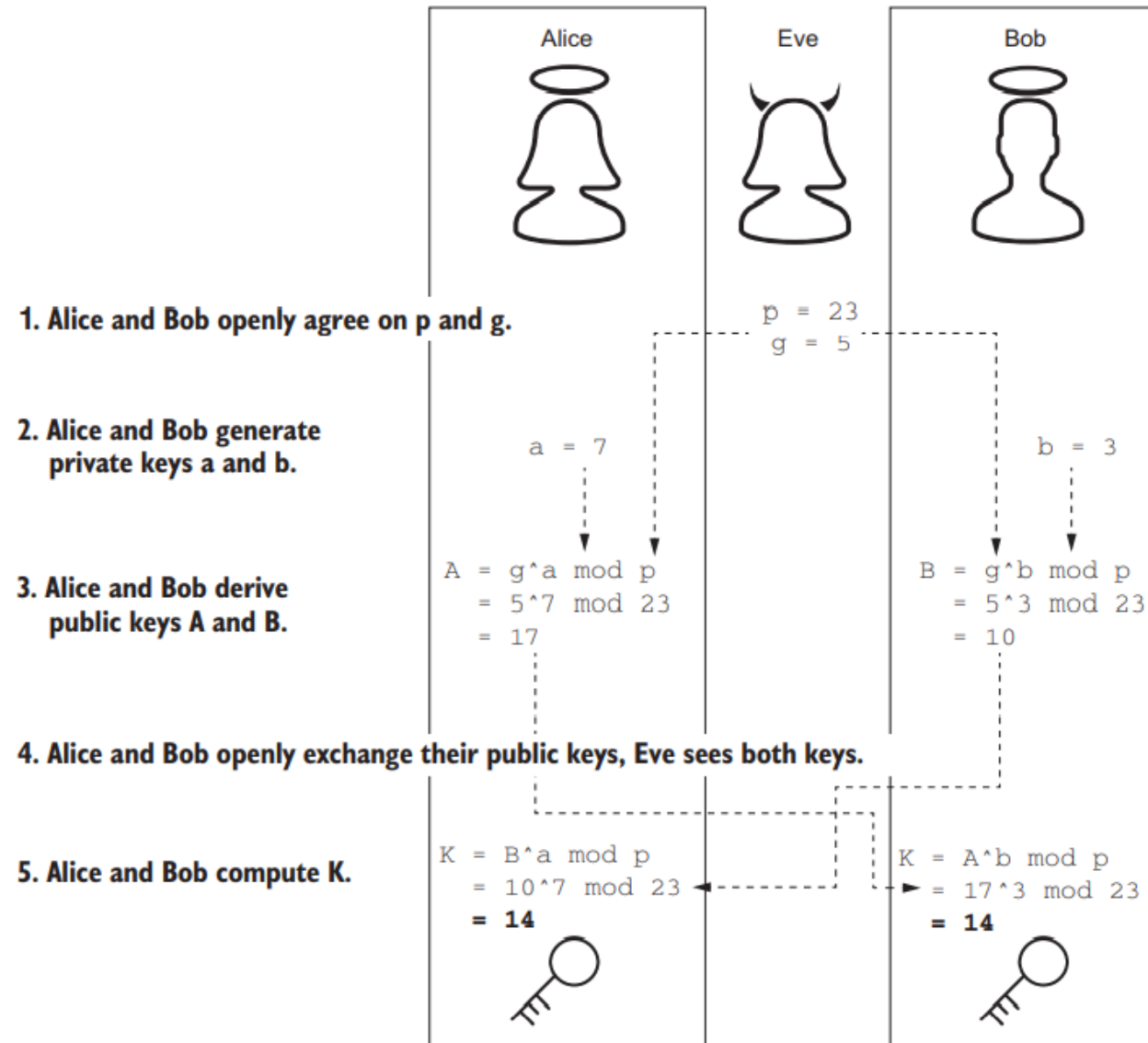
CIPHER SUITE NEGOTIATION



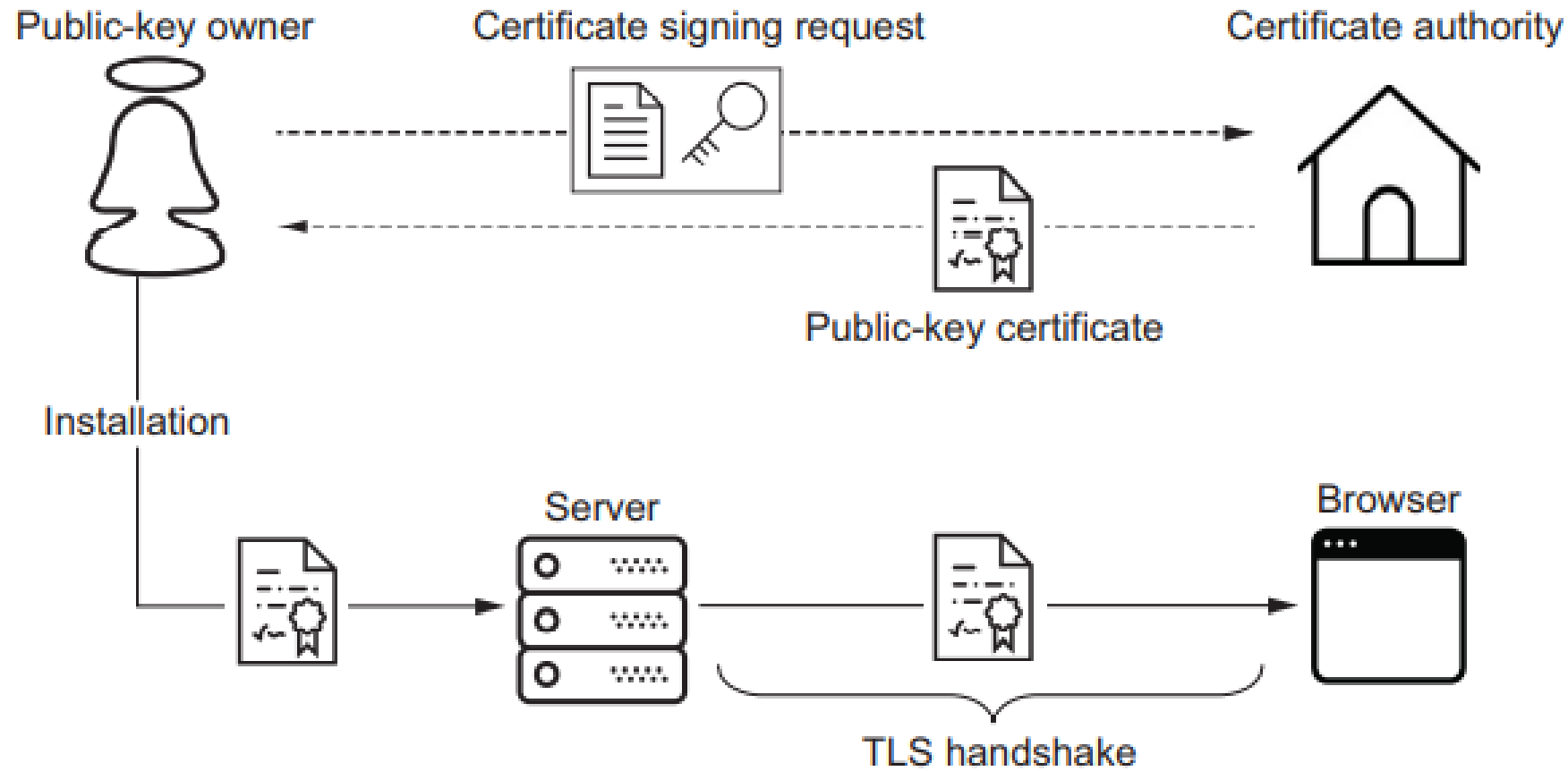
Cipher Suite Segments:

- Common prefix
- Encryption algorithm
- Hashing algorithm

DIFFIE-HELLMAN KEY EXCHANGE IN TLS 1.3



SERVER AUTHENTICATION

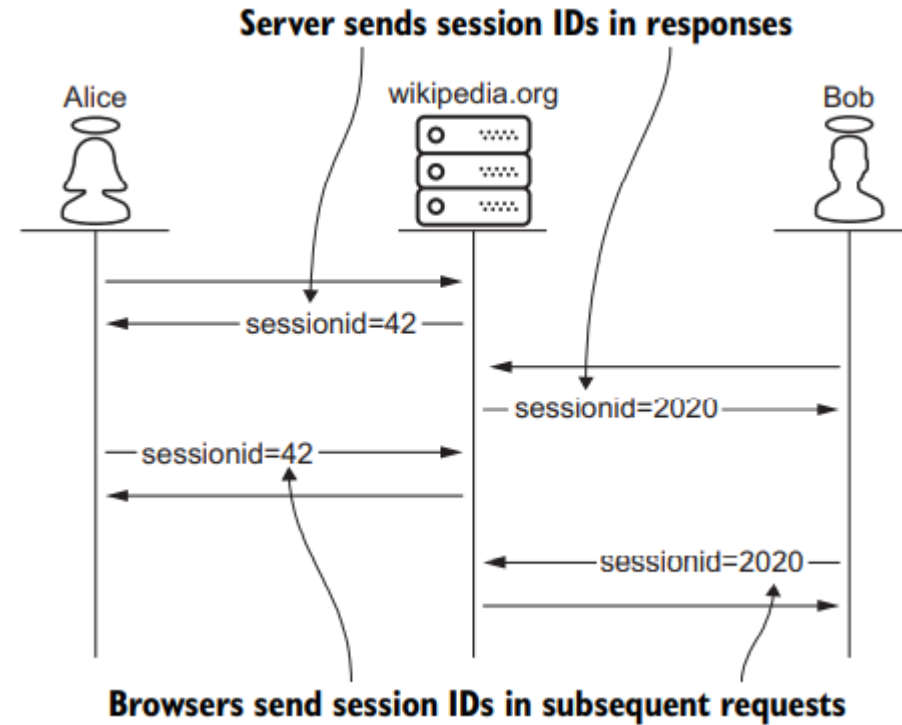


LAB 3.0: CREATE A DJANGO WEB APPLICATION AND USE HTTPS WITH GUNICORN

HTTP SESSION MANAGEMENT

HTTP Sessions are used to:

- Isolate traffic
- Context
- User state



HTTP COOKIES

Server sends sessionId to browser using the Set-Cookie response header.

```
Set-Cookie: sessionId=<cookie-value>
```

Browser echoes back cookies to the server in each request via the Cookie request header.

```
Cookie: sessionId=cgqbyjpxaoc5x5mmm9ymcqtspb7w7cn1; key=value;  
Host: alice.com  
Referer: https://alice.com/admin/login/?next=/admin/
```

SET-COOKIES DIRECTIVES

Secure directive helps servers resist MITM

```
Set-Cookie: sessionid=<session-id-value>; Secure
```

Domain directive is used to control which hosts the browser should send the session ID to.

```
Set-Cookie: sessionid=<session-id-value>; Domain=alice.com
```

Max-age directive is used to set an expiration time for the cookie.

```
Set-Cookie: sessionid=<session-id-value>; Max-Age=1209600
```

SETTING COOKIES WITH DJANGO

```
from django.http import HttpResponseRedirect

response = HttpResponseRedirect()
response.set_cookie('my-cookie', 'my-cookie-value',
                    secure=True, domain='mydomain.com', max_age=3000)
```

SESSION-STATE PERSISTENCE

```
from django.http import HttpRequest

def get_set_data(request: HttpRequest):
    request.session['username'] = 'My Name'
    username = request.session['username']
    del request.session['username']
```

DJANGO SESSION SERIALIZER

```
>>> from django.contrib.sessions.serializers import JSONSerializer
>>>
>>> json_serializer = JSONSerializer()
>>> serialized = json_serializer.dumps({'name': 'Bob'})
>>> serialized
b'{"name": "Bob"}'
>>> json_serializer.loads(serialized)
{'name': 'Bob'}
```

The diagram illustrates the process of serializing and deserializing session data using Django's JSONSerializer. It shows a sequence of operations: creating a serializer, dumping a Python dictionary into a JSON string, and then loading that string back into a Python dictionary. Arrows and labels indicate the direction and nature of the data transformation.

- Serialized Python dict**: Points to the output of the `dumps()` method, which is a JSON string.
- Serialized JSON**: Points to the JSON string output.
- Deserializes JSON**: Points to the output of the `loads()` method, which is a Python dictionary.
- Deserialized Python dict**: Points to the final output, a Python dictionary.

Django delegates the serialization and deserialization of session state to a configurable component. This component is configured by the `SESSION_SERIALIZER` setting. Django natively supports two session serializer components:

- `JSONSerializer`, the default session serializer
- `PickleSerializer`

CACHE-BASED SESSIONS

Store session state in a cache service:

- Redis
- Memcached

```
#Cache based session  
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
```

MEMCACHED BACKENDS

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
    },
    'cache': {
        'BACKEND': 'django.core.cache.backends.memcached.PyLibMCCache',
        'LOCATION': '/tmp/memcached.sock',
    }
}
```

← **Local loopback address**

← **UNIX socket address**

ADDITIONAL CACHE BACKENDS SUPPORTED BY DJANGO

```
CACHES = {  
    'default': {  
        'BACKEND': 'django.core.cache.backends.db.DatabaseCache',  
        'LOCATION': 'database_table_name',  
    }  
}
```

Database Backends

```
CACHES = {  
    'default': {  
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',  
    },  
    'dummy': {  
        'BACKEND': 'django.core.cache.backends.dummy.DummyCache',  
    }  
}
```

Local memory,
Dummy Backends

```
CACHES = {  
    'default': {  
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',  
        'LOCATION': '/var/tmp/file_based_cache',  
    }  
}
```

FileSystem Backend

SECRET_KEY SETTING

The SECRET_KEY setting is used by Django to perform keyed hashing.

Django uses an HMAC function for authentication and verification of the session state.

Using an HMAC function is vulnerable to HTTP Session attacks.

EXAMPLE ATTACKS ON HTTP SESSIONS

Unauthorized access to session state via MITM:

Cookie-based session engine does not encrypt session state. There is no guarantee for confidentiality.

Replay attack:

Browser can replay an older version of a request. A server cannot tell if the payload it receives is the latest.

Remote Code Execution:

Combining cookie-based sessions with PickleSerializer is risky if an attacker has access to the SECRET_KEY setting.

LAB 3.1: CREATE A DJANGO WEB APPLICATION WITH SECURE SESSION COOKIES

USER AUTHENTICATION

Authentication is the term that refers to the process of proving a user's identity. Users prove their identity by proving their credentials.

The most popular authentication mechanism is the username and password authentication. Before a user can access their account, they must prove that they own the correct credentials.

A user must first be registered and their account activated in order to access an application or service.

DJANGO USER REGISTRATION

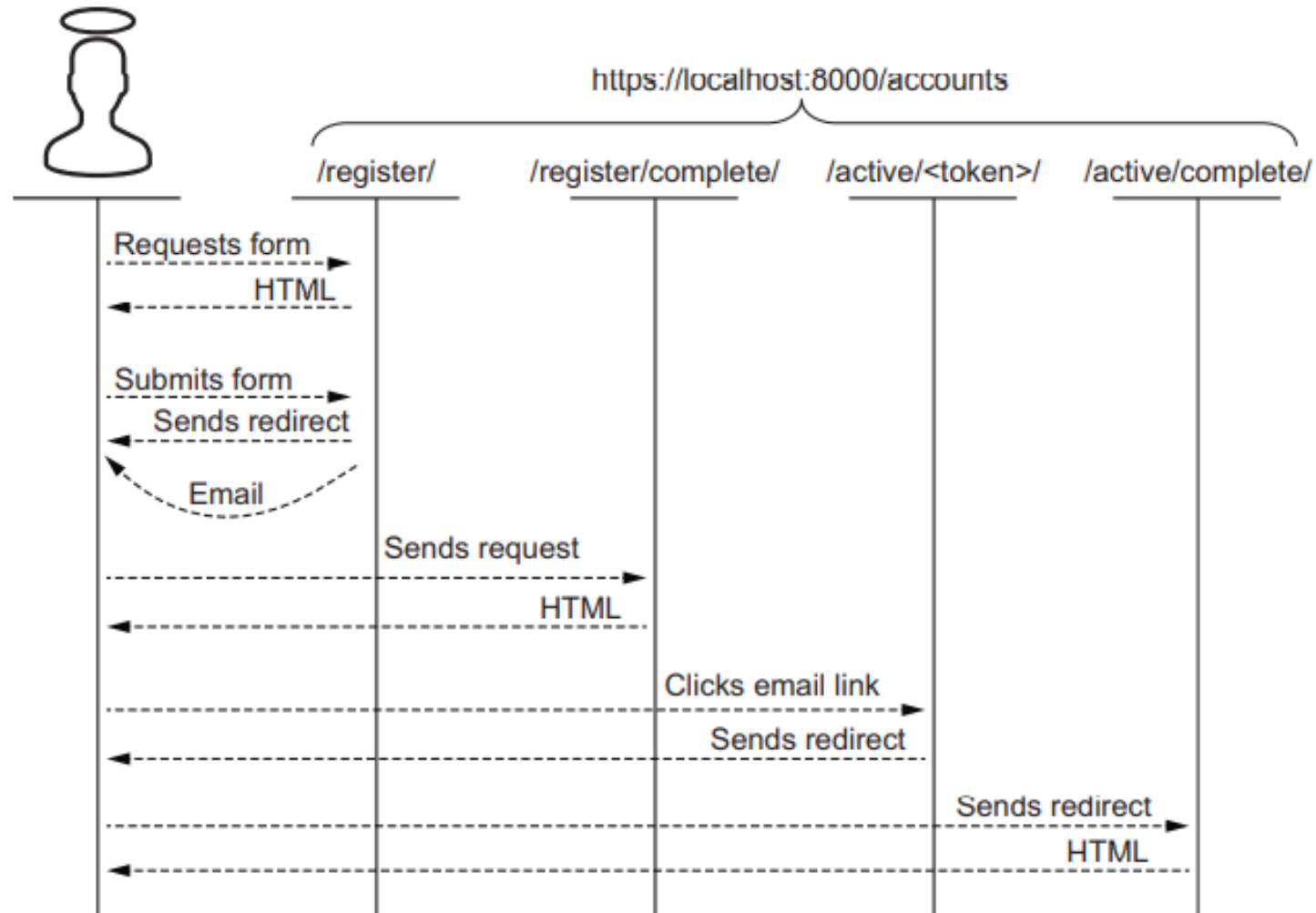
Use django-registration library to create a user-registration workflow.

Django web development building blocks

The user registration workflow consists of:

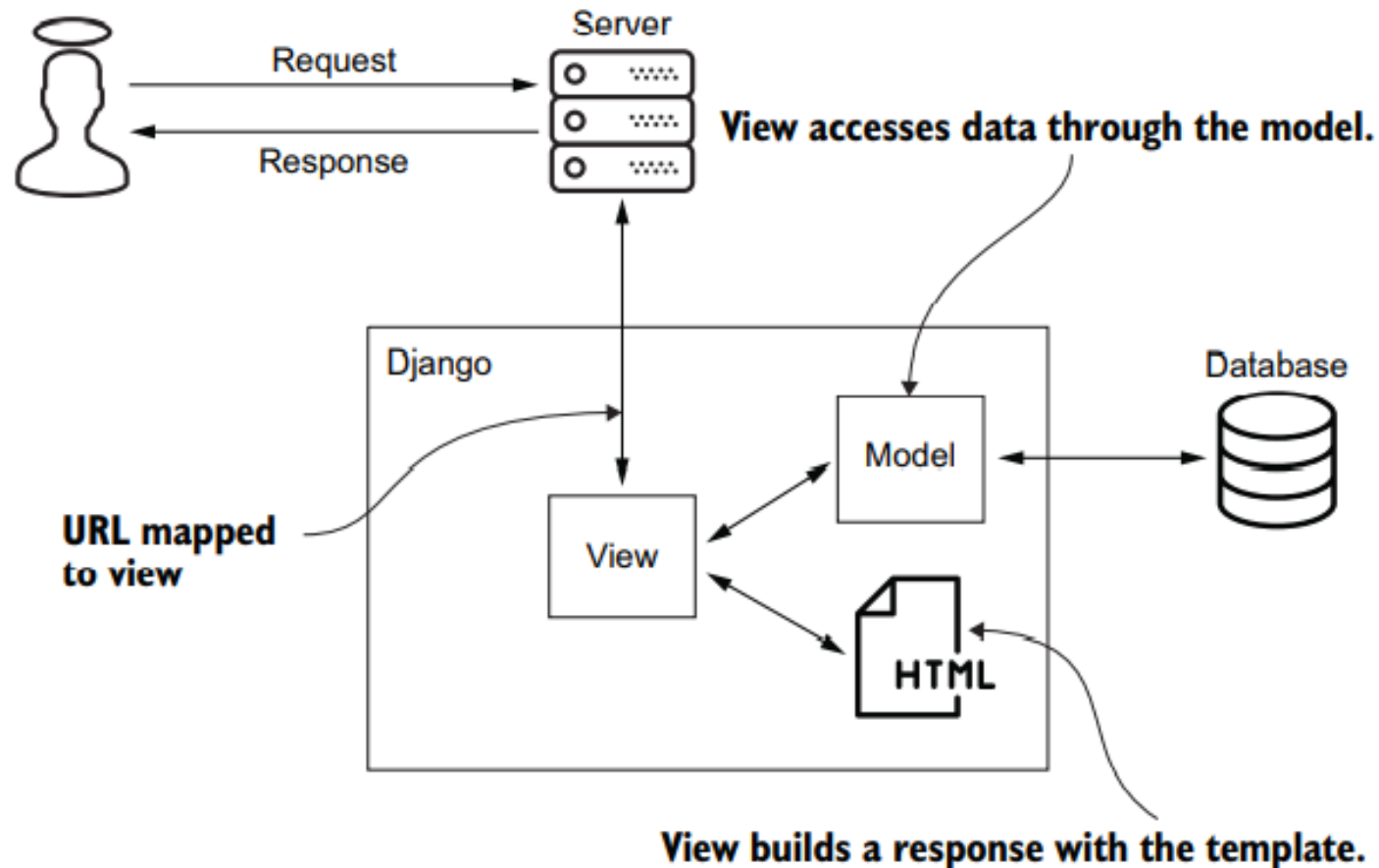
- Models
 - Views
 - Templates
-
- Each inbound HTTP request is an object.
 - A view is a python request handler
 - Models is a class that represent application data

DJANGO USER REGISTRATION WORKFLOW



DJANGO MODEL-VIEW-TEMPLATE ARCHITECTURE

MVT term	MVC term	Description
Model	Model	Object-relational mapping layer
View	Controller	Request handler responsible for logic and orchestration
Template	View	Response content production



DJANGO USER REGISTRATION APP

You can install the Django user registration app by running the following command:

```
pip install django-registration
```

Then add `django_registration` to the list of `INSTALLED_APPS` in the `settings.py` file.

Make modifications to include user registration in the Django database by running:

```
python manage.py migrate
```

DJANGO BUILT-IN AUTHENTICATION VIEWS

The django-registration app takes advantage of secure built-in views that ensures that you adhere to best practices and ship your application to production faster. You can add the views by adding their urls using the `django.contrib.auth.urls` module.

```
urlpatterns = [  
    #Built-In Django URLs  
    path('accounts/',include('django.contrib.auth.urls')),  
]
```

DJANGO-REGISTRATION VIEWS

Before being able to login, a user's account must be registered and activated. Django provides a secure and easy way to enable this using django-registration views.

```
#Built-In Django URLs
path('accounts/', include('django.contrib.auth.urls')),
#Django registration views
path('accounts/', include('django_registration.backends.activation.urls')),
]
```

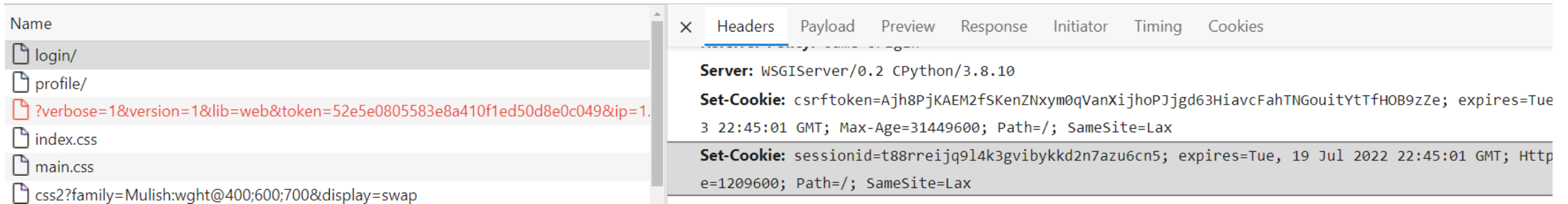
After you create an account, Django generates an activation url like the one below:

https://localhost:8000/accounts/activate/ImJvYil:1o8qdj:Nv_sWghtG-rMqUW1TvFLeZJQjgac2YpcN9TrWJ_MPY/

The token contains a URL-encoded timestamp and a keyed hash value. Django uses the SECRET_KEY and an HMAC function to hash the username and account creation time.

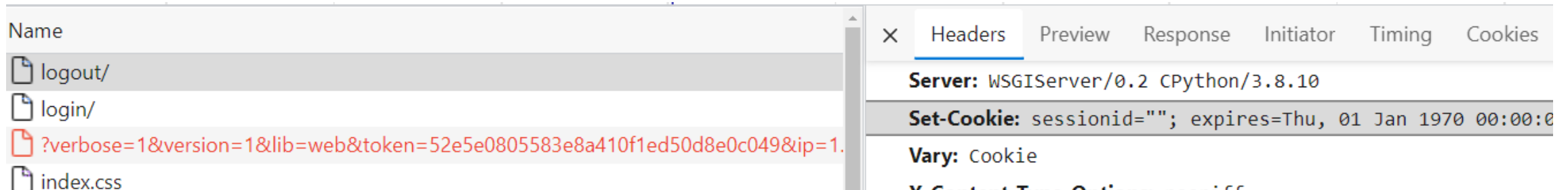
LOGIN AND SESSION ID CREATION

Once a user is logged in, the LoginView will send a response to the browser with a Set-Cookie response header that delivers the session ID to the client.



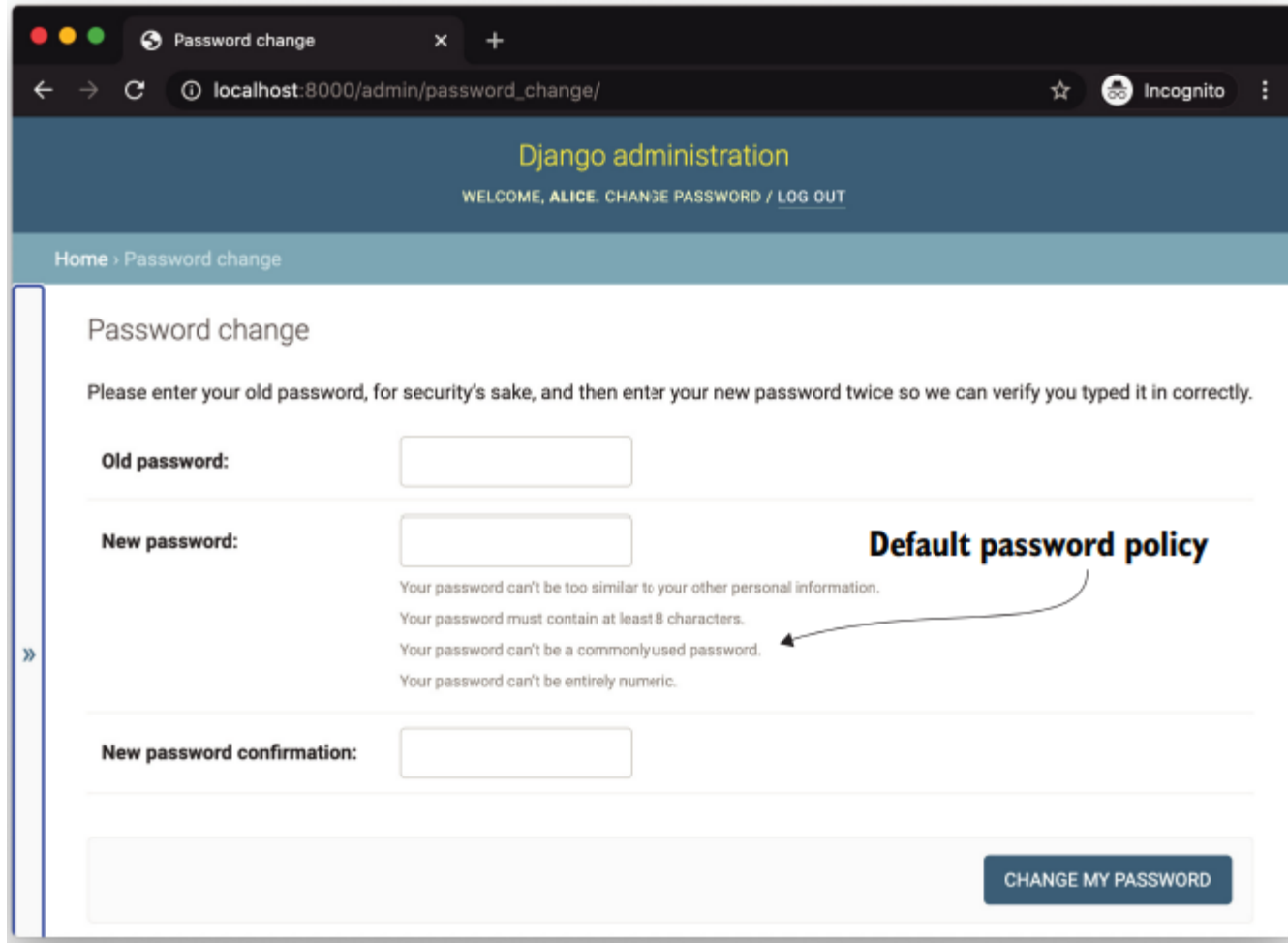
LOGOUT AND SESSION ID DELETION

When you call the LogoutView, it uses Set-Cookie header to set the session ID to an empty string thus invalidating the session.



LAB 3.2: ADDING USER REGISTRATION & AUTHENTICATION TO A DJANGO APP

MANAGING PASSWORD IN DJANGO



The screenshot shows a web browser window with the title 'Password change' and the URL 'localhost:8000/admin/password_change/'. The page is part of the 'Django administration' interface, with a header bar showing 'WELCOME, ALICE. CHANGE PASSWORD / LOG OUT'. The main content area is titled 'Password change' and contains the following text: 'Please enter your old password, for security's sake, and then enter your new password twice so we can verify you typed it in correctly.' Below this text are three input fields: 'Old password:', 'New password:', and 'New password confirmation:'. To the right of the 'New password:' field, there is a section titled 'Default password policy' with four bullet points: 'Your password can't be too similar to your other personal information.', 'Your password must contain at least 8 characters.', 'Your password can't be a commonly used password.', and 'Your password can't be entirely numeric.' A blue button labeled 'CHANGE MY PASSWORD' is located at the bottom right of the form.

PasswordChangeView and PasswordChangeDoneView enforces the password change workflow

https://localhost:8000/admin/password_change/

PASSWORD POLICY

```
AUTH_PASSWORD_VALIDATORS = [  
    {  
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',  
    },  
    {  
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',  
    },  
    {  
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',  
    },  
    {  
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',  
    },  
]
```

The password policy of a Django project is defined by the `AUTH_PASSWORD_VALIDATORS` Setting

Uses validators to define a password policy.

CUSTOM PASSWORD VALIDATOR

```
from django.core.exceptions import ValidationError
from django.utils.translation import gettext_lazy as _
```

```
class PassphraseValidator:
```

```
    def __init__(self, dictionary_file='/usr/share/dict/words'):
```

```
        self.min_words = 4
```

```
        with open(dictionary_file) as f:
```

```
            self.words = set(word.strip() for word in f)
```

**Loads a dictionary
file into memory**

**Communicates
the constraint
to the user**

```
    def get_help_text(self):
```

```
        return _('Your password must contain %s words' % self.min_words)
```

CUSTOM PASSWORD VALIDATOR

```
def validate(self, password, user=None):  
    tokens = password.split(' ')
```

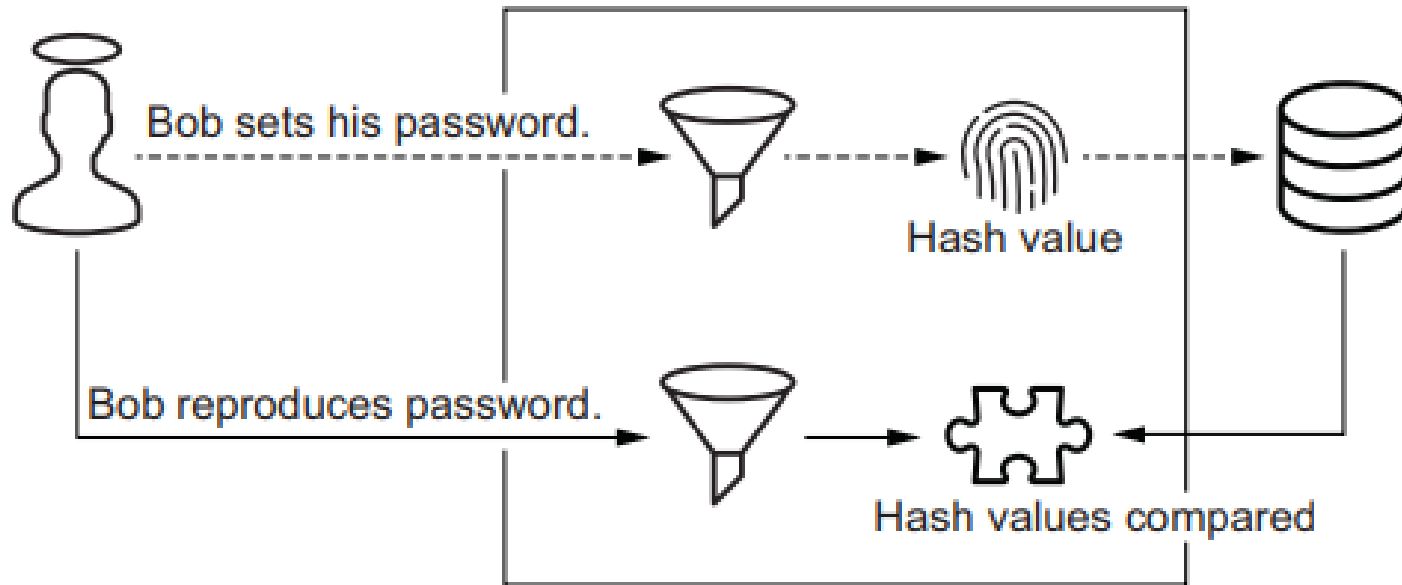
**Ensures each password
is four words**

```
    if len(tokens) < self.min_words:  
        too_short = _('This password needs %s words' % self.min_words)  
        raise ValidationError(too_short, code='too_short')
```

**Ensures each
word is valid**

```
    if not all(token in self.words for token in tokens):  
        not_passphrase = _('This password is not a passphrase')  
        raise ValidationError(not_passphrase, code='not_passphrase')
```

SECURE PASSWORD STORAGE

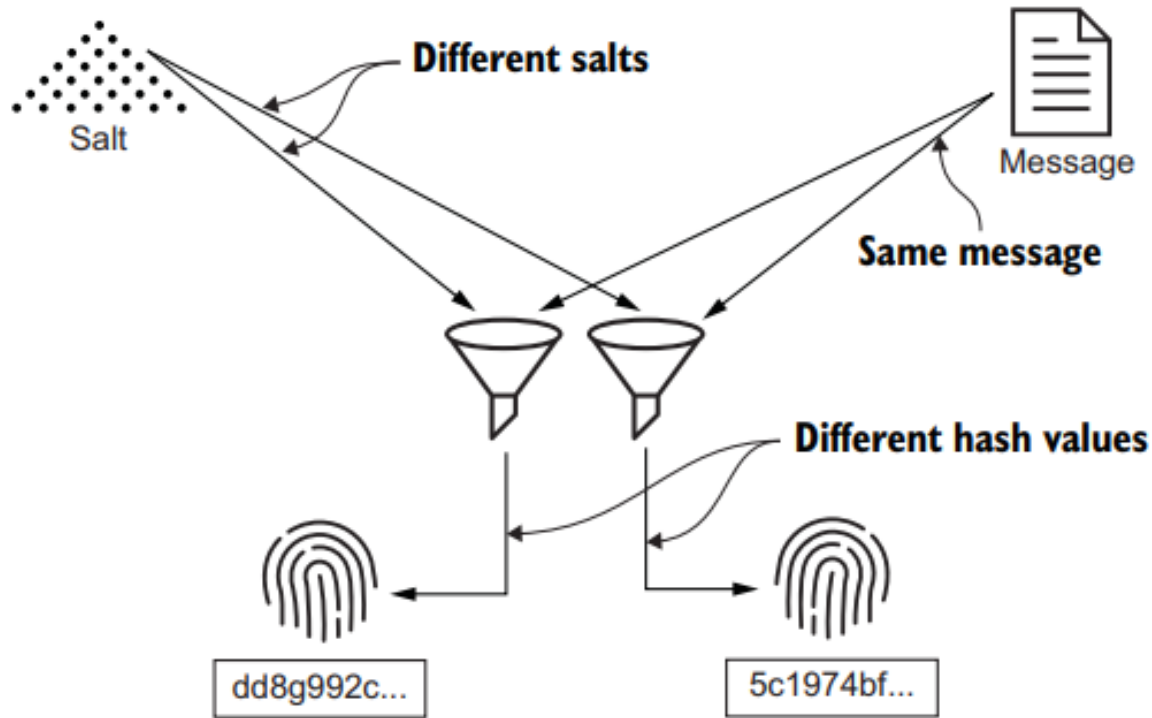


Hash functions are one way; hence a hashed password is not easy to recover unlike with encrypted passwords which can be decrypted and compromised.

Hashing alone is also prone to attackers using:

- Common password lists
- Hash function determinism

SALTED HASHING



```
from hashlib import blake2b
import secrets

message = b"message"

salt_a = secrets.token_bytes(16)
salt_b = secrets.token_bytes(16)

salted_hash_1 = blake2b(message, salt=salt_a)
salted_hash_2 = blake2b(message, salt=salt_b)

print(salted_hash_1.digest() == salted_hash_2.digest())
```

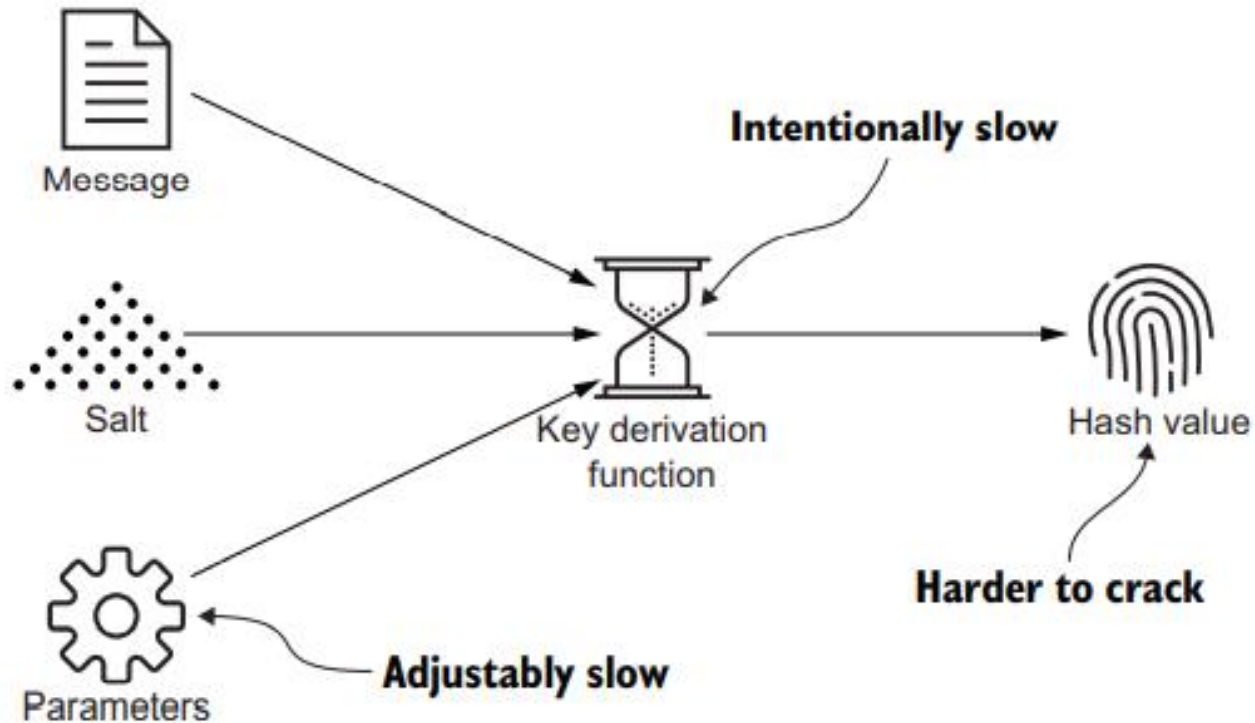
✓ 0.3s

False

A salt is a random string of bytes that accompanies the message as input to a hash function.

Each message is paired with a unique salt and produces a unique hash value.

KEY DERIVATION FUNCTIONS

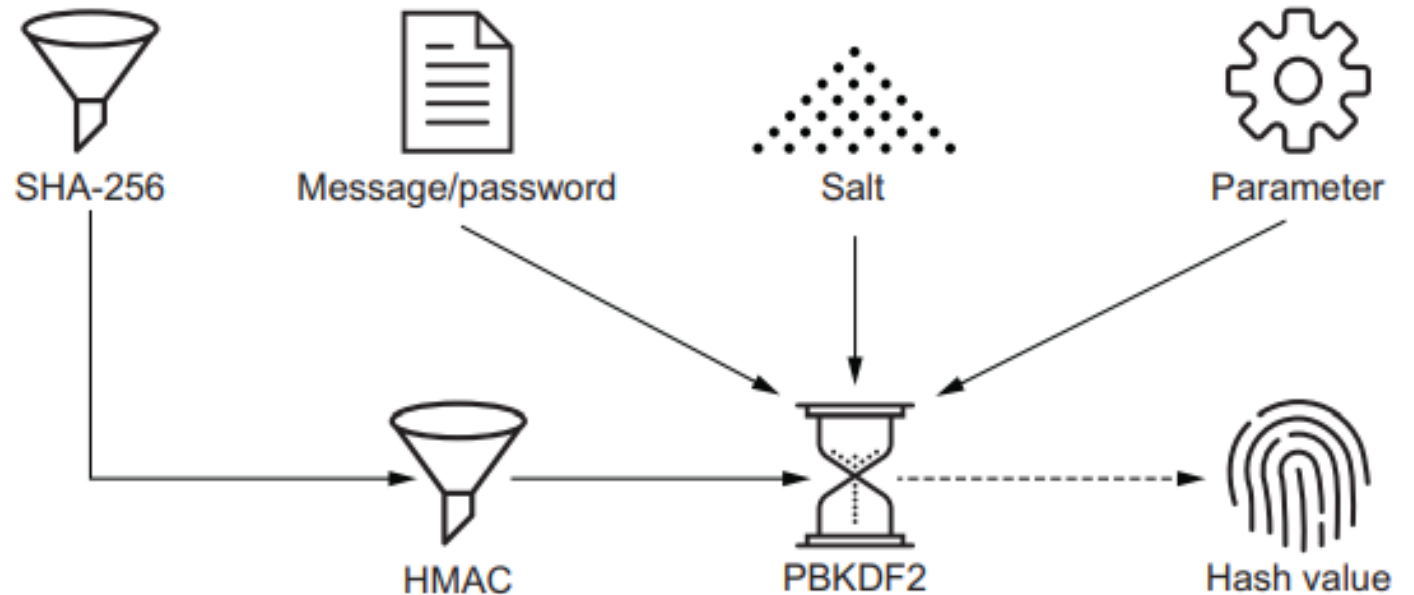


Preferred to hash functions because of their high resource consumption that can be configured using parameters.

The high resource consumption makes it expensive to crack passwords using brute-force.

PASSWORD BASED KEY DERIVATION FUNCTION 2

Django uses PBKDF2 to hash passwords. PBKDF2 wraps HMAC-SHA256.



<https://en.wikipedia.org/wiki/PBKDF2>

CONFIGURE DJANGO PASSWORD HASHING

```
PASSWORD_HASHERS = [  
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',  
    'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',  
    'django.contrib.auth.hashers.Argon2PasswordHasher',  
    'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',  
]
```

You can use the `PASSWORD_HASHERS` setting to define a list of hashers that Django can use.

The first password hasher in the list is used to hash new passwords.

Authentication attempts against stored hash values can be verified by any password hasher in the list.

<https://docs.djangoproject.com/en/4.0/topics/auth/passwords/>

UNSAFE PASSWORD HASHERS

Using MD5 or SHA1 for password hashing is not safe. Django, however, has 4 SHA1 and MD5 password hashers that it maintains for backward compatibility.

- `django.contrib.auth.hashers.SHA1PasswordHasher`
- `django.contrib.auth.hashers.MD5PasswordHasher`
- `django.contrib.auth.hashers.UnsaltedSHA1PasswordHasher`
- `django.contrib.auth.hashers.UnsaltedMD5PasswordHasher`

CUSTOM PASSWORD HASHERS

The default number of iterations (work factor - # of times the algorithm is run over a hash) for PBKDF2PasswordHasher is 260,000 in Django. You can create a custom password hasher and increase this value

```
from django.contrib.auth.hashers import PBKDF2PasswordHasher

class MyPBKDF2PasswordHasher(PBKDF2PasswordHasher):
    """
    A subclass of PBKDF2PasswordHasher that uses 100 times more iterations.
    """
    iterations = PBKDF2PasswordHasher.iterations * 100
```

<https://docs.djangoproject.com/en/4.0/topics/auth/passwords/#pbkdf2-and-bcrypt>

ARGON2 PASSWORD HASHING

Argon2 is the recommended password hashing algorithm for Django. It relies on an external python library (argon2-cffi) that can be installed using: *pip install django[argon2]*

You must then add *django.contrib.auth.hashers.Argon2PasswordHasher* to the PASSWORD_HASHERS setting.

```
PASSWORD_HASHERS = [  
    'django.contrib.auth.hashers.Argon2PasswordHasher',  
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',  
    'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',  
    'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',  
    'django.contrib.auth.hashers.ScryptPasswordHasher',  
]
```

<https://docs.djangoproject.com/en/4.0/topics/auth/passwords/#using-argon2-with-django>

UPGRADING HASHED PASSWORDS

Django automatically upgrade stored passwords to the first hash, `PASSWORD_HASHES[0]` if they are not already hashed with the preferred algorithm.

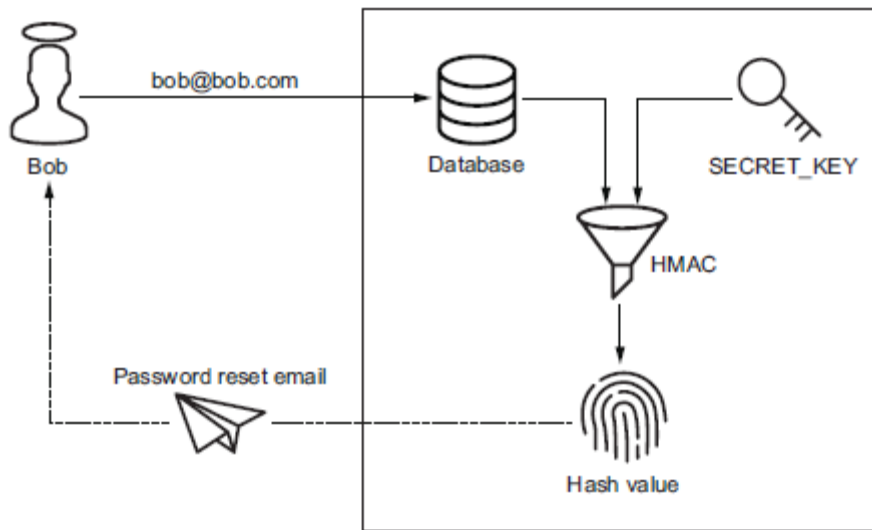
Alternatively, you can use a wrapped password hasher to manually upgrade passwords without waiting for user login.

You can follow the ADD-MIGRATE-DELETE pattern, which works as follows:

1. Add the new hashing algorithm to `PASSWORD_HASHERS` as the first entry
2. Add a custom wrapper for the preferred password hasher and use it to hash the stored hash value
3. Add a custom Django migration that iterates through each user and updates their stored hash value to a new password hashed with the preferred algorithm.
4. Remove the old algorithm from `PASSWORD_HASHERS`

READING MATERIAL: Full Stack Python Security page 133

PASSWORD RESET



During a password reset, a user is sent a password-reset url which includes a token that is a keyed hash value.

Like the activation token, the password reset token is produced with an HMAC function and the key defined in the SECRET_KEY setting.

To prevent malicious users from resetting passwords, the password-reset token has an expiry which can be configured using the PASSWORD_RESET_TIMEOUT

LAB 3: MANAGING PASSWORDS & AUTHORIZATION IN A DJANGO APPLICATION

AUTHORIZATION

After a user has been authenticated, the next step is controlling what they can access.

Authorization is a mechanism used to determine what an application user is able to do.

DEFAULT DJANGO PERMISSIONS

When you create a model and run migrations, Django automatically generates the following permissions:

- add_lowercase_model_name
- change_lowercase_model_name
- delete_lowercase_model_name
- view_lowercase_model_name

You can view the list of permissions on a model by calling `Permission.objects.filter()`

```
>>> from django.contrib.auth.models import Permission
>>> permissions = Permission.objects.all()
>>> [p.codename for p in permissions]
['add_logentry', 'change_logentry', 'delete_logentry', 'view_logentry', 'view_group', 'add_permission', 'change_permission', 'delete_permission', 'change_user', 'delete_user', 'view_user', 'add_contenttype', 'change_contenttype', 'add_members', 'change_members', 'delete_members', 'delete_session', 'view_session']
>>>
```

CUSTOM PERMISSIONS

```
from django.db import models

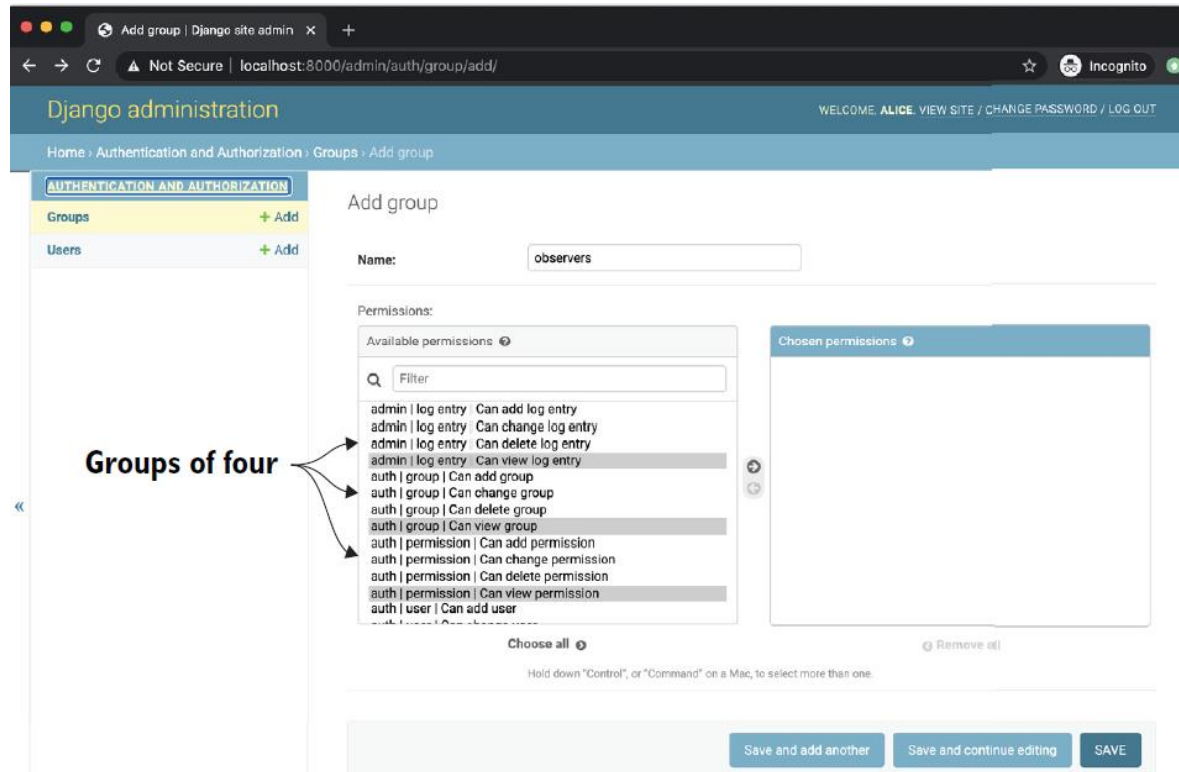
# Create your models here.
class Members(models.Model):
    firstname = models.CharField(max_length=255)
    lastname = models.CharField(max_length=255)

    class Meta:
        permissions = [
            ("approve_members", "Can approve membership")
        ]
```

You can add a list of custom permissions to a model by defining an inner Meta class.

The custom permissions are automatically created during migrations.

DJANGO USER & GROUP ADMINISTRATION



Django has a built-in administration console that allows user and group management.

You can access the console by visiting <https://localhost:8000/admin>.

You can create a superuser with root permissions using the following command:

```
python manage.py createsuperuser --username=abc --email=abc@abc.com
```

GROUP MEMBERSHIP

```
from django.contrib.auth.models import User
from django.contrib.auth.models import Group, Permission

bob = User.objects.get(username='bob')
observers = Group.objects.get(name='observers')
can_send =
Permission.objects.get(codename='send_authenticatedmessage')
bob.groups.add(observers)
bob.user_permissions.add(can_send)
bob.groups.remove(observers)
bob.user_permissions.remove(can_send)
```

Groups provide a generic way for applying permissions for a set of users.

You can manage groups programmatically or by using the Django admin application.

PROGRAMMATIC USER & GROUP ADMINISTRATION

```
from django.contrib.auth.models import User, Group, Permission

def change_permissions():
    user1 = User.objects.get(username='bob')
    administrators = Group.objects.get(name='administrators')

    can_approve_loan = Permission.objects.get(codename='aprove_loan')

    user1.groups.add(administrators)
    user1.user_permissions.add(can_approve_loan)

    user1.groups.remove(administrators)
    user1.user_permissions.remove(can_approve_loan)
```

You can use the User, Group and Permission classes to programmatically manage users, groups and permissions.

AUTHORIZATION USING DECORATORS

The easiest way to enforce authorization in Django views is to use the `@permission_required` decorator.

When an unauthorized user makes a request to an action decorated with `@permission_required` a 403 authorization error is generated and the user is redirected to a login page.

```
@login_required
@permission_required('members.view_members')
def index(request):
    members = Members.objects.all().values()
    template = loader.get_template('index.html')
    context = {
        'mymembers': members
    }
    response = HttpResponse(template.render(context, request))
    return response
```

AUTHORIZATION USING

The easiest way to enforce authorization in Django views is to use the `@permission_required` decorator.

When an unauthorized user makes a request to an action decorated with `@permission_required` a 403 authorization error is generated and the user is redirected to a login page.

```
@login_required
@permission_required('members.view_members')
def index(request):
    members = Members.objects.all().values()
    template = loader.get_template('index.html')
    context = {
        'mymembers': members
    }
    response = HttpResponse(template.render(context, request))
    return response
```

AUTHORIZATION USING CUSTOM LOGIC

```
from django.contrib.auth.decorators import user_passes_test
from django.http import JsonResponse

def test_func(user):
    return user.email.endswith('@alice.com') or user.first_name == 'bob'
@user_passes_test(test_func)
def user_passes_test_view(request):
    data = {}
    return JsonResponse(data)
```

You can guard a resource by defining a custom function that accepts a user model as an argument, then decorate the resource with `@user_passes_test`.

CONDITIONAL RENDERING

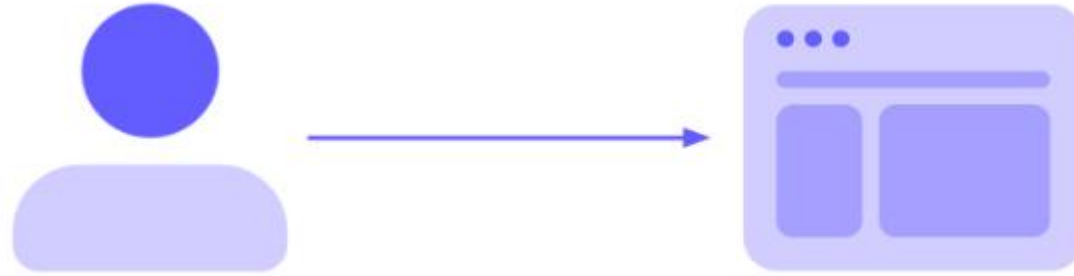
The default Django templating engine has authorization-based conditional rendering built-in by default.

You can access the permissions of the current user using the perms variable.

NB: Conditional rendering must never substitute server-side authorization.

```
{% if perms.microfinance.approve_loan %}  
<input type="submit" value="Approve" >  
{% endif %}
```

IDENTITY AND ACCESS MANAGEMENT

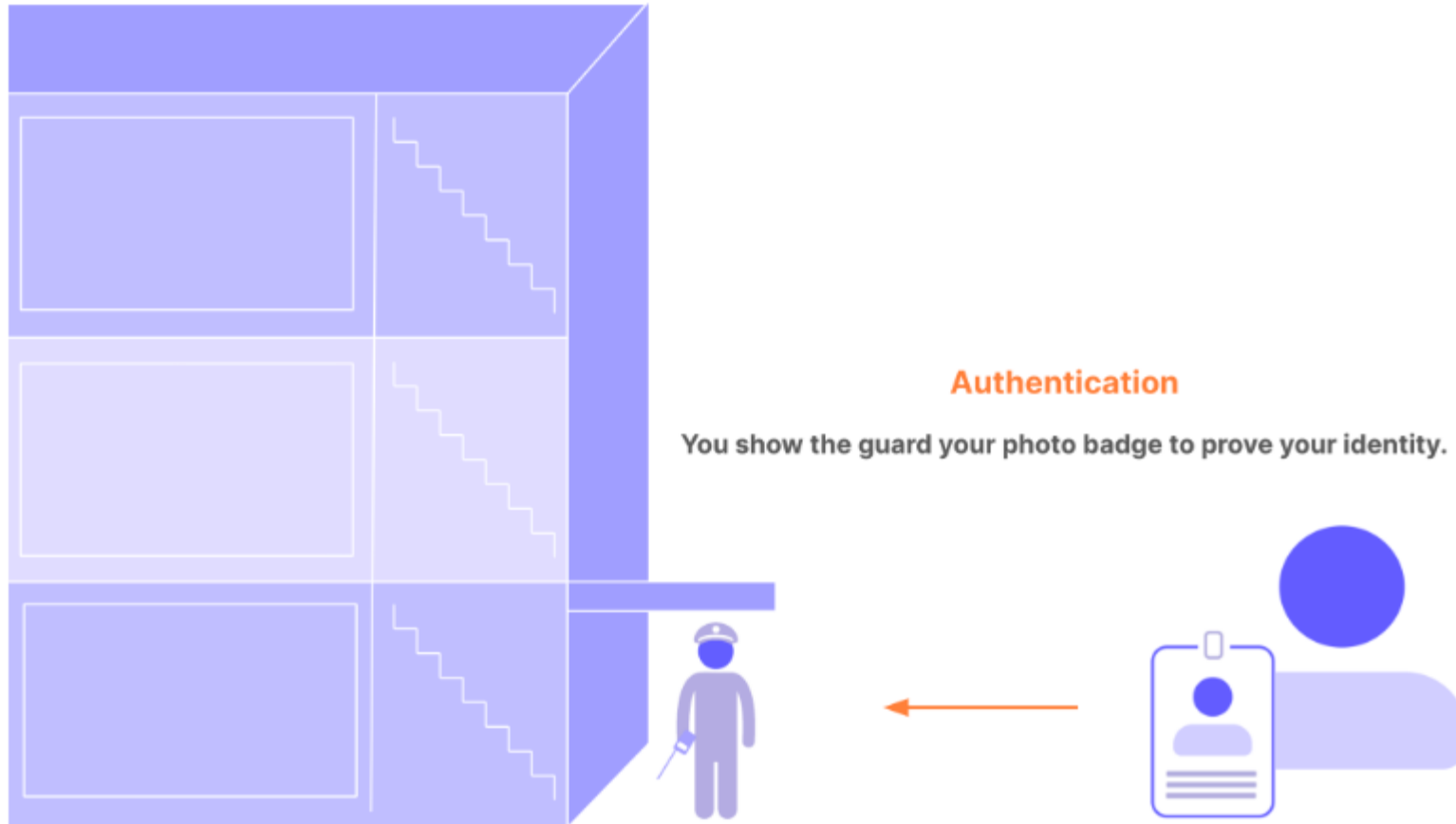


A user wants access to a resource.



Identity and access management verifies the user and controls their access to the resource.

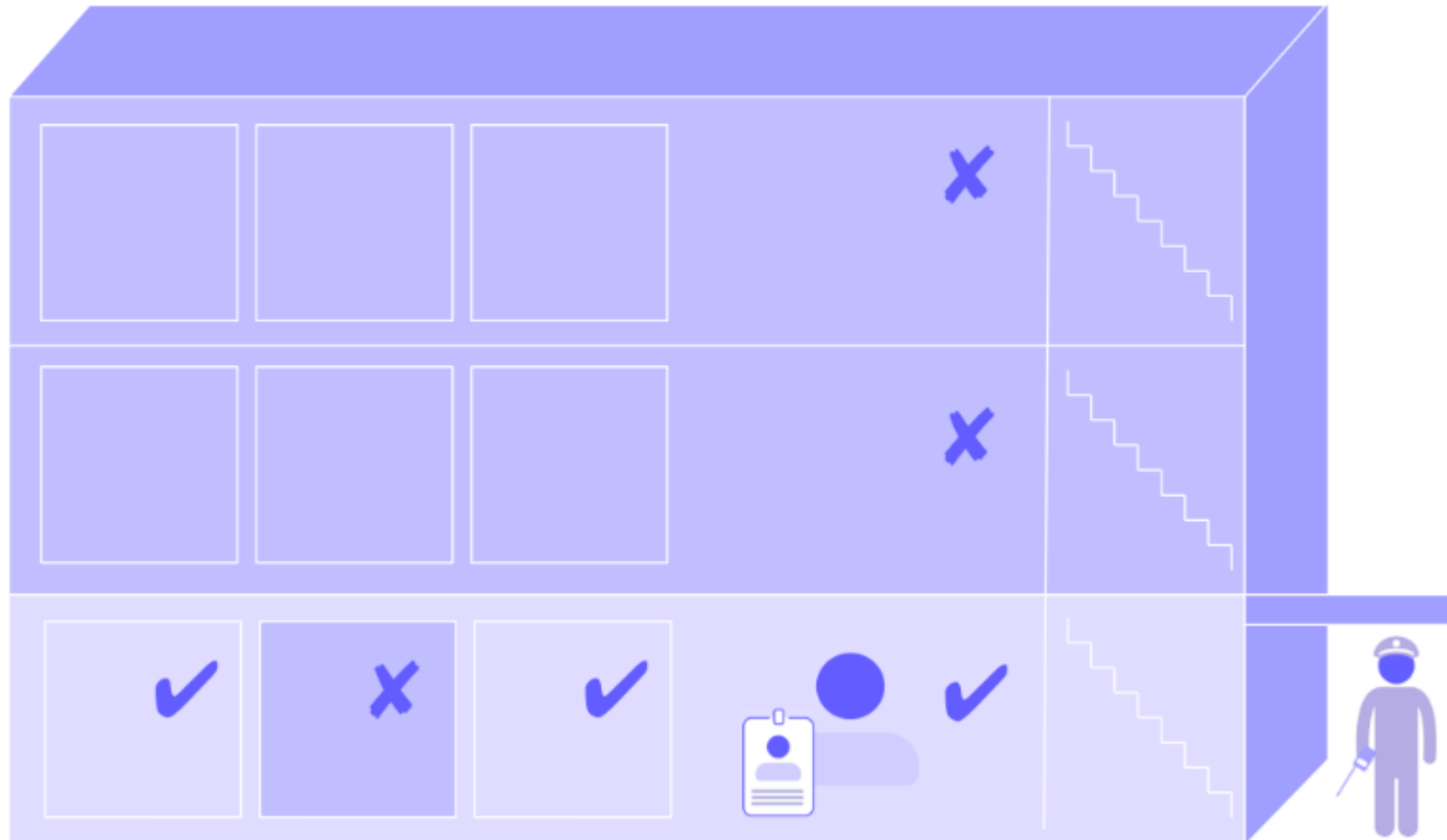
AUTHENTICATION



AUTHENTICATION

Authorization

You use the chip in your badge to enter only the floor and rooms you have permission to access.



WHAT DOES IAM DO?

- User registration
- Storage of user information
- How users can prove their identity
- When and how often users must prove their identity
- The experience of proving identity
- Who can and cannot access different resources

HOW IAM WORKS

IDENTITY PROVIDERS

- Microsoft Azure Active Directory
- AWS IAM
- Google Accounts
- Facebook
- LinkedIn

AUTHENTICATION FACTORS

- Knowledge
- Possession
- Inherence

AUTHENTICATION & AUTHORIZATION STANDARDS

- OAuth 2.0
- OpenID Connect
- JSON Web Tokens
- Security Assertion Mark Language (SAML)
- Web Services Federation (WS-Fed)

OAUTH 2.0

OAuth 2.0 (Open Authorization) is the industry standard protocol for authorization.

It allows websites or applications to access resources hosted by other web apps on behalf of a user.



OAuth 2.0

IDENTITY PROVIDERS

- Microsoft Azure Active Directory
- AWS IAM
- Google Accounts
- Facebook
- LinkedIn

AUTHENTICATION FACTORS

- Knowledge
- Possession
- Inherence

AUTHENTICATION & AUTHORIZATION STANDARDS

- OAuth 2.0
- OpenID Connect
- JSON Web Tokens
- Security Assertion Mark Language (SAML)
- Web Services Federation (WS-Fed)

OAuth 2.0 PRINCIPLES

OAuth is designed primarily as a means of granting access to a set of protected resources such as remote API's or data.

It makes use of Access Tokens in order to grant access. The access token is data that represents authorized access to protected resources on behalf of a user.

The format of the access token is not defined by OAuth so in most cases the JSON Web Token format is used.

OAuth 2.0 ROLES

The OAuth2.0 authorization framework defines the following essential components:

- a. Resource Owner: The user or system that owns the protected resources and can grant access to them.
- b. OAuth Client: The system that requires access to a protected resource and must hold a valid access token.
- c. Authorization Server: Receives requests from the client for access tokens and issues them upon successful authentication and consent by the Resource Owner.
- d. Resource Server: A server that protects the user's resources and receives access requests from the client.

OAUTH 2.0 ROLES EXAMPLE

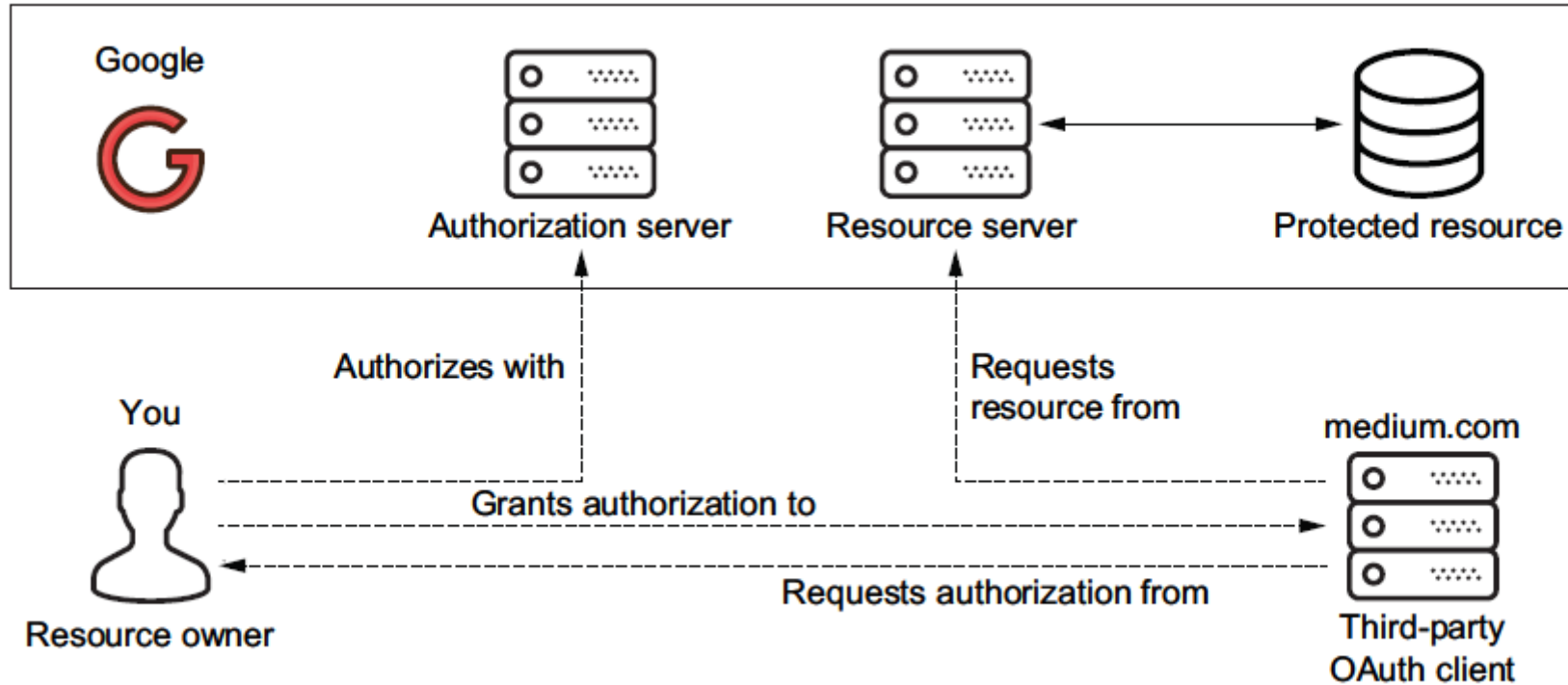


Image Source: Full Stack Python Security p156