IoT Security and Privacy

Lightweight IoT communication protocol - Message Queuing Telemetry Transport (MQTT)

YIER JIN

UNIVERSITY OF FLORIDA

EMAIL: YIER.JIN@ECE.UFL.EDU

SLIDES ARE ADAPTED FROM PROF. XINWEN FU @ UCF/UMASS

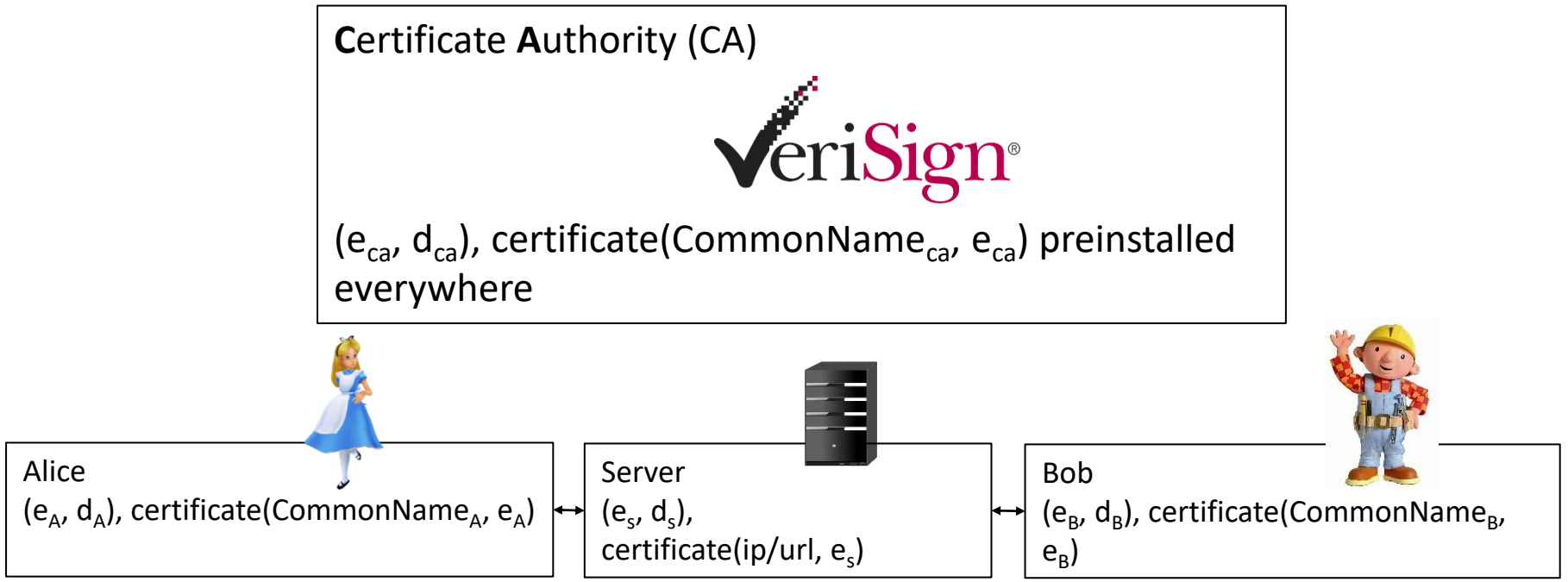
Certificate

X.509 - the most common format for public key certificates

- Very general

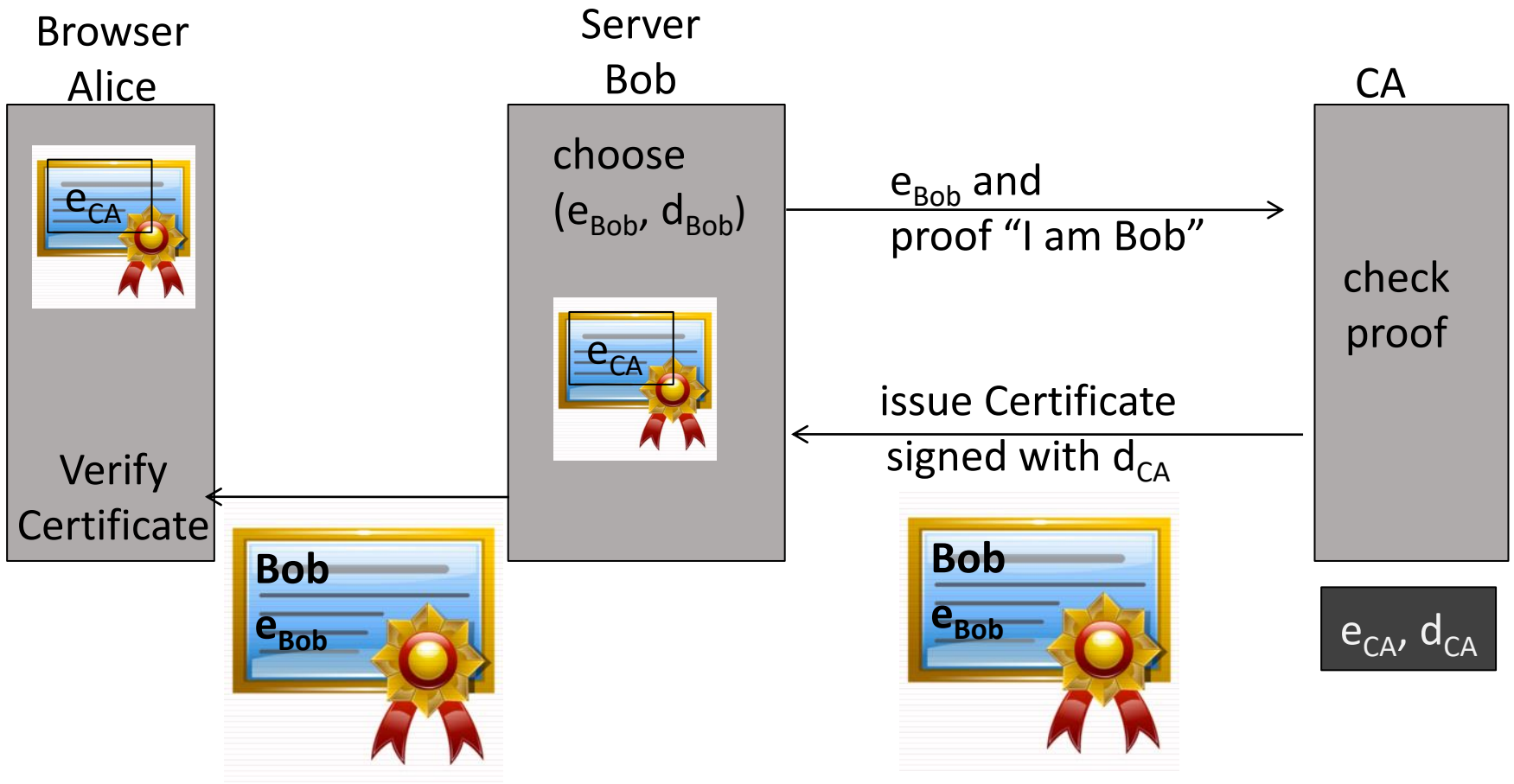
The format is use case oriented

- e.g. Public Key Infrastructure (PKI) X.509 in RFC 5280.



Verifying Certificates

How does Alice (browser) obtain e_{Bob} ?



Outline

Message Queuing Telemetry Transport (MQTT)

MQTT implementation: mosquitto

MQTT Mosquitto transport security

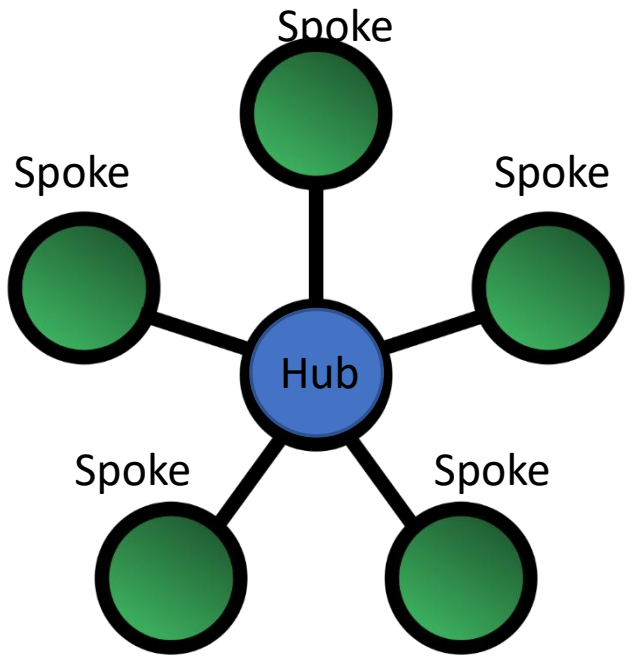
MQTT authentication

Messaging Broker System

A messaging broker system uses a publish/subscribe protocol based on a “hub and spoke” model

- Hub: server/broker
- Spokes: clients

Clients communicate with each other through the hub



Message Brokering Basic Terms

Broker, also called “servers”

- Accepts messages from clients
- Delivers the messages to any interested clients

Client

- Publishes a message to a topic, or
- Subscribes to a topic
- or both.

Topic: A namespace for messages on the broker

- A forward slash / is used to separate the topic hierarchy
- Clients do not need to initialize a topic before subscribing and publishing, and the broker will process the request automatically
- e.g., mysmarthome/groundfloor/familyroom/humidity

Message Brokering Basic Terms (Cont'd)

Publish: a client sends a message to the broker, using a topic name.

Subscribe: a client notifies the broker the topics of interest

- The broker sends messages published to that topic to subscribers
- A client can subscribe to multiple topics.

Unsubscribe:

- Tell the broker not to send the client the messages to a particular topic any more

MQTT Introduction

MQTT is a messaging broker system

Clients can publish (Pub) messages and subscribe (Sub) to topics.

- Clients can both publish and subscribe.

A broker communicates with clients.

Topics can have subtopics.

- Topics starting with $ are reserved for special topics
- Refer to AWS IoT topics

Outline

Message Queuing Telemetry Transport (MQTT)

MQTT implementation: mosquitto

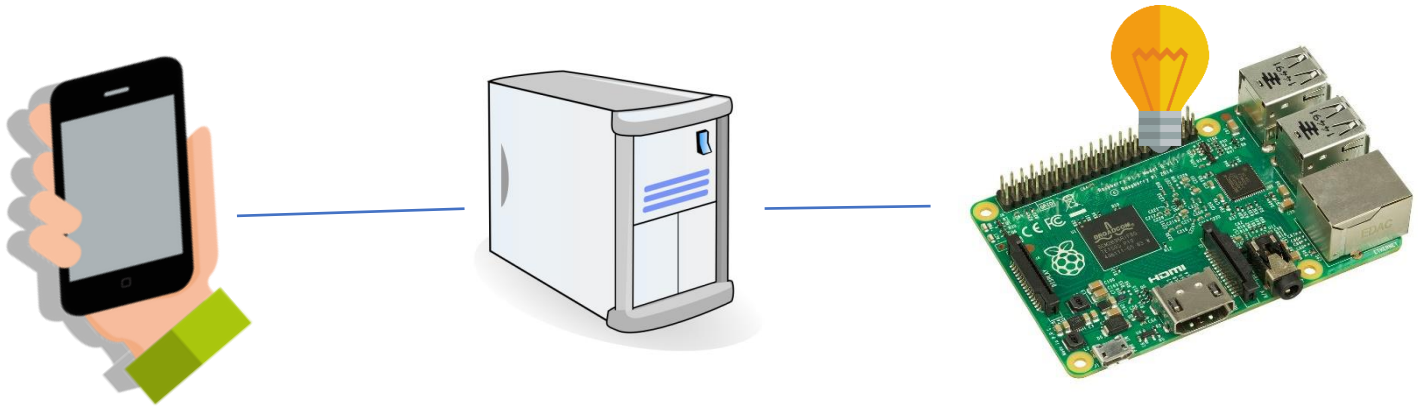
MQTT Mosquitto transport security

MQTT authentication

Example IoT system

Example IoT system components

- Smartphone – controller
- MQTT server – broker
- Client: IoT device



Mosquitto

Open source MQTT Mosquitto is a broker server

- Shipped with publishing and subscribing utilities - use *mosquitto_pub* and *mosquitto_sub*

Windows: binary installers on mosquitto.org

Linux: install “mosquitto” or “mosquitto-mqtt” with a package manager

Mac: use homebrew to install mosquitto.

- Add /usr/local/sbin to PATH by editing /etc/paths if necessary

Running Mosquitto

- Runs on port 1883 with no security by default

Testing Mosquitto

On one computer, we can test the whole MQTT system

1st console (terminal): the server

- mosquitto -v # verbose mode

2nd console: subscribing to MQTT Topic with Mosquitto

- mosquitto_sub -h 127.0.0.1 -i testSub -t debug
- **Host flag (-h)** indicates the mosquitto server
- **Identity flag (-i)** is client id. mosquitto_sub creates one if not provided.
- **Topic flag (-t)** indicates the topic to subscribe

Notice: no topic is pre-created on the server

- The topic is created when the subscriber or publish connect to the server.

Outline

Message Queuing Telemetry Transport (MQTT)

MQTT implementation: mosquito

MQTT authentication

MQTT Mosquitto transport security

Authentication

By default, no authentication

Unauthenticated encrypted support is provided through the use of the certificate based SSL/TLS based options cafile/capath, certfile and keyfile.

- The broker needs to provide the client a certificate

username/password authentication

Through password_file

- Define usernames and passwords.
- C:\Program Files\mosquitto\mosquitto_passwd.exe" -c –b <file_name> <username> <password>

If no encryption used, the username and password will be transmitted in plaintext

- SSL/TLS should be used

Mosquitto configuration

```
# Plain MQTT protocol
listener 1883

# End of plain MQTT configuration

# MQTT over TLS/SSL
listener 8883
cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/hostname.crt
keyfile /etc/mosquitto/certs/hostname.key

# End of MQTT over TLS/SLL configuration
```

```
# Plain WebSockets configuration
listener 9001
protocol websockets

# End of plain Websockets configuration

# WebSockets over TLS/SSL
listener 9883
protocol websockets
cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/hostname.crt
keyfile /etc/mosquitto/certs/hostname.key
```

pre-shared-key based encryption

pre-shared-key based encryption through the password_file option in the configuration file,

- The client must provide a valid identity and key to connect to the broker

If use_identity_as_username is true

- The PSK identity is used for access control purposes.

If use_identity_as_username is false

- The client may still authenticate via username/password so that different users have different passwords

Outline

Message Queuing Telemetry Transport (MQTT)

MQTT implementation: mosquitto

MQTT authentication

MQTT Mosquitto transport security

mosquitto.conf

mosquitto.conf

- Configuration file for mosquitto.
- Can be put anywhere.
- By default, mosquitto does not need a configuration file and will use the default values.
- Refer to the man page mosquitto(8) for information on how to load a configuration file.

Format

- Line with # as the very first character are comments.
- Configuration lines: a variable name and its value separated by a single space.

Mosquitto configuration

```
# Plain MQTT protocol
listener 1883

# End of plain MQTT configuration

# MQTT over TLS/SSL
listener 8883
cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/hostname.crt
keyfile /etc/mosquitto/certs/hostname.key

# End of MQTT over TLS/SLL configuration
```

```
# Plain WebSockets configuration
listener 9001
protocol websockets

# End of plain Websockets configuration

# WebSockets over TLS/SSL
listener 9883
protocol websockets
cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/hostname.crt
keyfile /etc/mosquitto/certs/hostname.key
```

Mosquitto broker with SSL/TLS

Generating the server certificates

- Navigate to openssl directory
- Run openssl executable
- Pass **req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -keyout dir\CA.key -out dir\CA.crt**
- When asked for common name, enter server ip address

The following files are generated:

- **CA.crt** – The CA (Certificate Authority, who published the host certificate) public certificate.
- **CA.key** – The certificate authority private key.

Mosquitto broker with SSL/TLS

- Next we must generate a key for the server
  - openssl genrsa -out dir\server.key 2048
- Generate a certificate signing request (CSR)
  - openssl req -out dir\server.csr -key dir\ server.key –new
- And send it to the CA
  - openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days <duration>
- Repeat this for the client
  - Means that the client must be given its certificate/should not be able to access the CA directly
- https://mosquitto.org/man/mosquitto-tls-7.html

Privacy Enhanced Mail

- Not quite done!
- How to establish chain of trust?
- Privacy Enhanced Mail more common
  - For x509 based
  - Other approaches in industry, a bit more complicated
- PEM has us simply concatenate all certificates in the chain
  - In order!
  - This way each can easily be checked
- CertN->Cert(N-1)->…-> CA

Testing MQTT TLS/SSL configuration

"C:\Program Files\mosquitto\mosquitto.exe" -c mosquitto.conf -v

"C:\Program Files\mosquitto\mosquitto_pub.exe" -t example_secure -h localhost -m 'test' -p 8883 --cafile root.crt -u "class_pub" -d --key pub.key --cert pub.crt -i "class_pub" -P "class_pub“

"C:\Program Files\mosquitto\mosquitto_sub.exe" --cafile root.crt --key sub.key --cert sub.crt -t example_secure -h localhost -p 8883 -i "class_sub" -d -u "class_sub" -P "class_sub"

Authentication via certificate based encryption

Through require_certificate, which can be true or false

If require_certificate false, no certificate based authentication for clients

- Clients can verify server's certificate
- Authentication of clients will have to rely on username/password if needed

If require_certificate true, the client must provide a valid certificate to the server before further communication

- use_identity_as_username can affect the authentication.
  - If true, the Common Name (CN) from the client certificate is used as the identity
  - "If false, the client **must** (?) authenticate as normal (if required by password_file) through the MQTT options."

Notes

Certificate and PSK based encryption are configured for each listener.

Authentication plugins can be created to replace the password authentication (password_file) and psk authentication psk_file

- For example, database lookups.

Multiple authentication schemes can be simultaneously supported

```
# Plain MQTT protocol
listener 1883

# End of plain MQTT configuration

# MQTT over TLS/SSL
listener 8883
cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/hostname.crt
keyfile /etc/mosquitto/certs/hostname.key

# End of MQTT over TLS/SLL configuration
```

References

[1] James Lewis, MQTT Introduction and Tutorial Part One - Message Brokers and why your IoT device should use them, February 17, 2016.
[2] James Lewis, MQTT Tutorial for Raspberry Pi, Arduino, and ESP8266 - Send MQTT messages between 3 different platforms, February 24, 2016
[3] Python Software Foundation, paho-mqtt 1.2, 2016
[4] mosquitto.conf — the configuration file for mosquitto, 2016
[5] Primal Cortex, MQTT Mosquitto broker with SSL/TLS transport security, March 31, 2016
[6] J. Dunmire, SSL/TLS Client Certs to Secure MQTT, 2016
[7] MosquittoAn Open Source MQTT v3.1/v3.1.1 Broker, Documentation, 2016
[8] HuyITF, Configure SSL/TLS for MQTT broker mosquitto, Jun 2, 2016
[9] Roger Light, libmosquitto — MQTT version 3.1 c client library, 2016
[10] mosquitto.h, 2016
[11] https://www.meridianoutpost.com/resources/articles/IP-classes.php