IoT Security – Autumn 2024

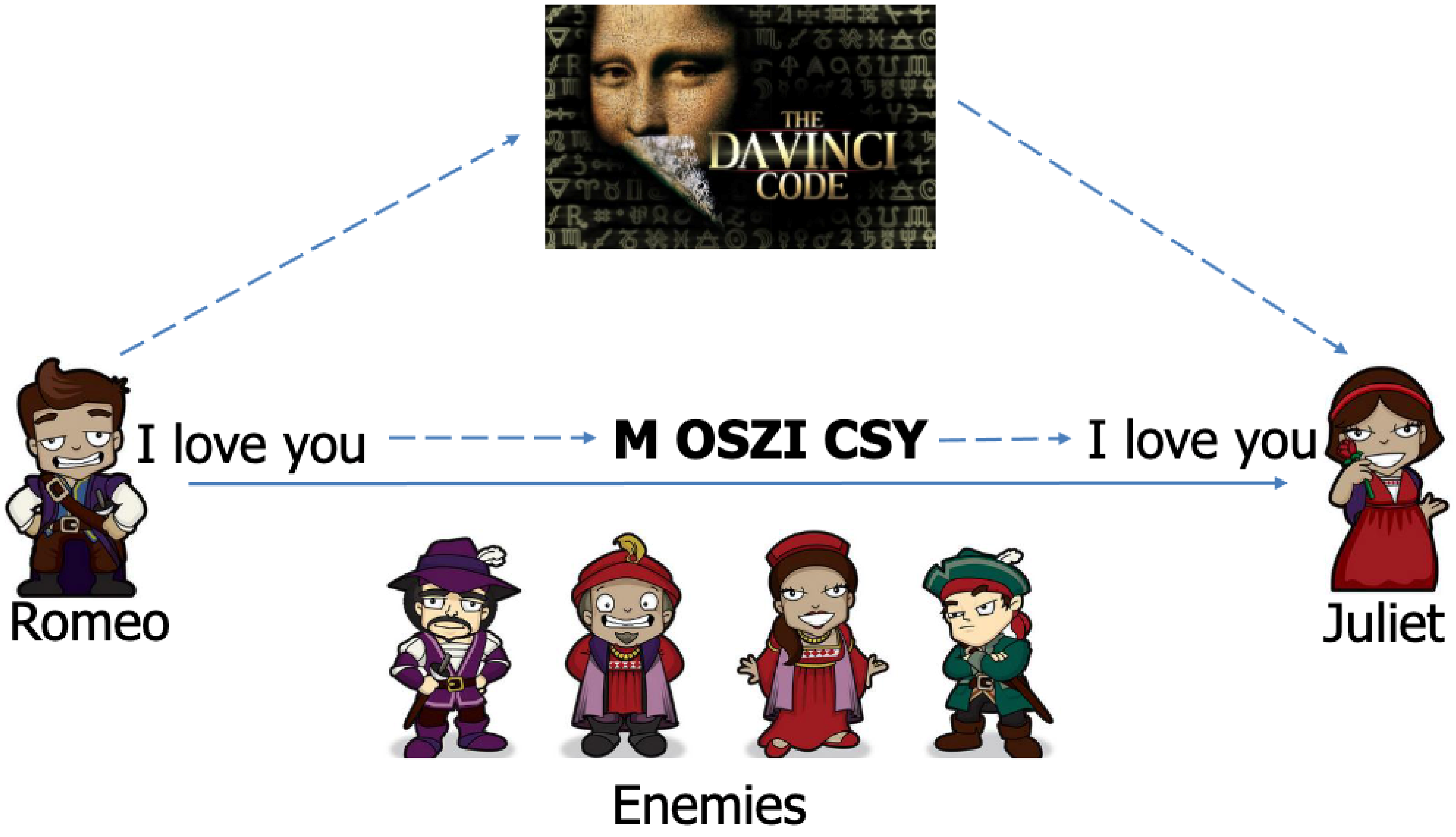
Lab 3: Authentication between sensor nodes with UDP

Manh Bui
School of Electrical and Data Engineering
Email: DucManh.Bui@uts.edu.au

Contact

- Name: Manh Bui
- Email/Teams: DucManh.Bui@uts.edu.au

Cryptography



Authentication between nodes: Caesar cipher

- The Caesar cipher is one of the earliest known and simplest encryption techniques
- Key idea: each letter in the plaintext is **“shifted”** a certain number of positions (key) down the alphabet.
- The method is named after Julius Caesar, who apparently used it to communicate with his generals.

Authentication between nodes: Caesar cipher

Plaintext

IoT Security

Key: 4 →

Ciphertext

MsX Wigyvmxc

A	B	C	D	E	F	G
H	I	~~J~~	~~K~~	~~L~~	M	N
O	P	Q	R	S	T	U
	V	W	X	Y	Z	

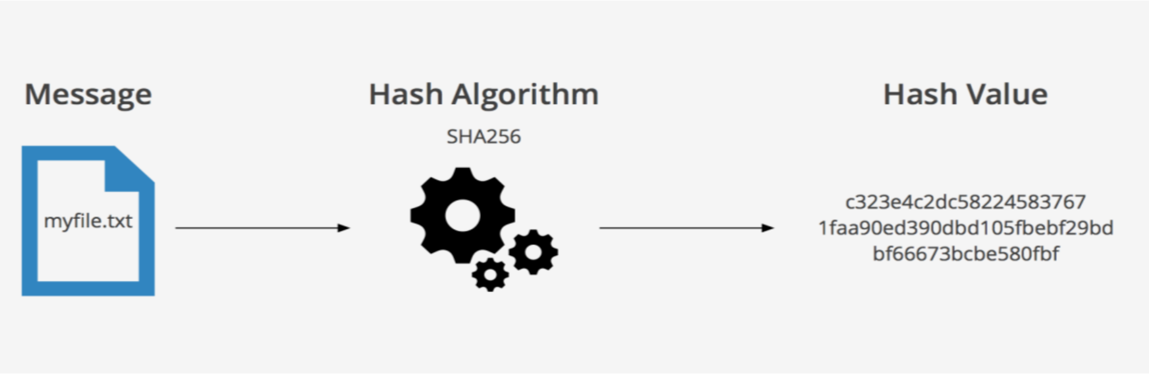
IoT node authentication

Scenario:

- An IoT network with **a UDP server** and **a UDP client**.
- The client's identity **"client1"** is registered with the server.
- The client uses **SHA-256 hashing algorithm** to calculate the hash value of its identity and send to the server.
- Upon receiving the hash value from the client, the server checks the received hash value by calculating the same hash.
- After successfully validate the client's identity, the server sends an **ACK message** to the client.
- Upon receiving the ACK message, the client encrypts its message **"confidential"** with **Caesar cipher** and sends to the server.
- The server decrypts the message with the same encryption algorithm and send message **"Msg Received"** to the client.
- Both the client and server are using the same key value **(key = 5)** for encryption and decryption.

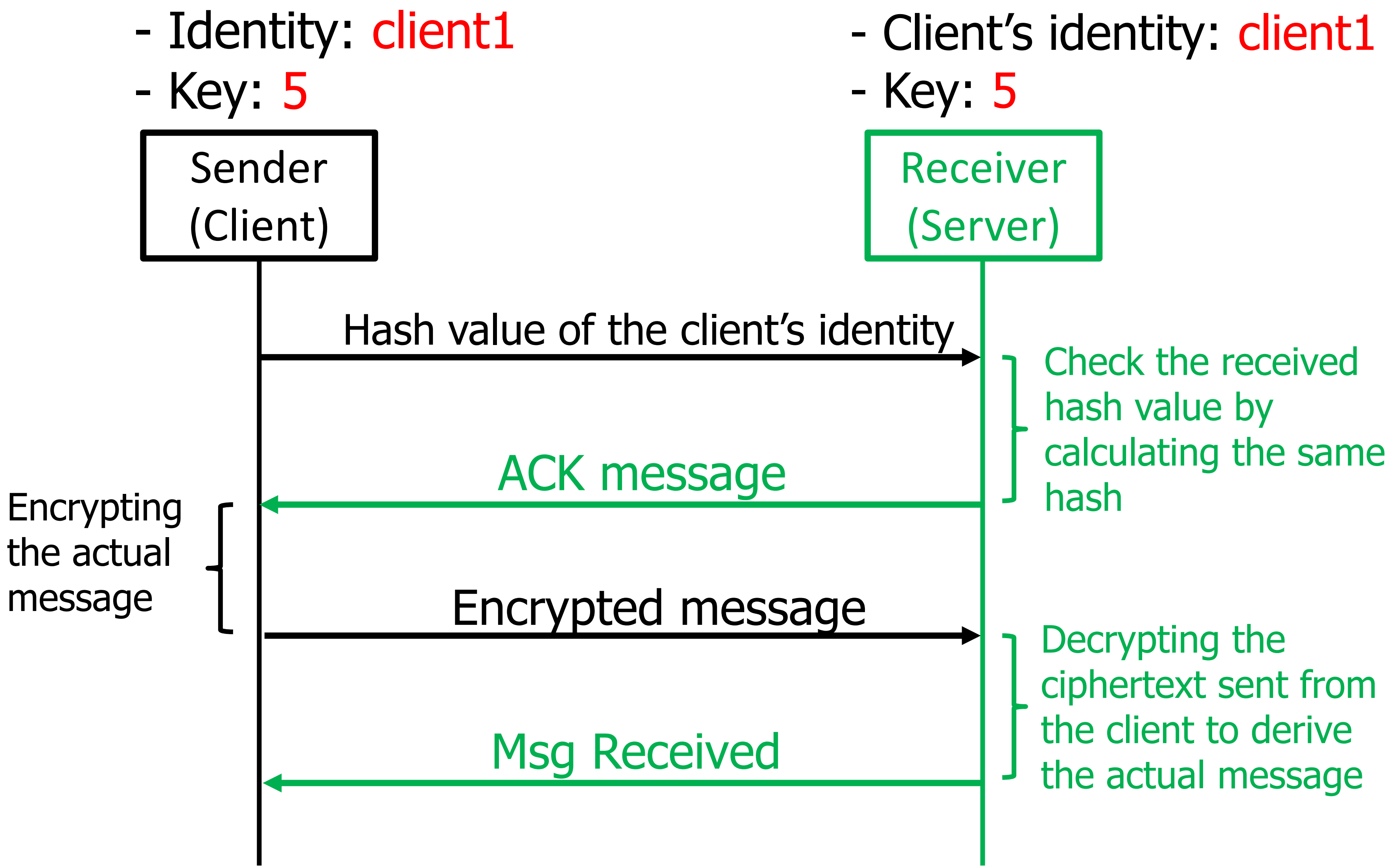
Secure Hash Algorithm: SHA-256

- Is a cryptographic hash algorithm (created by US National Security Agency in 2002) commonly used in Blockchain.
- SHA-256 generates a signature for a text or a data file.
- SHA-256 generates an unique, fixed size 256-bit (32-byte) hash for a text or a data file.
- Hash is a one way function. In other words, a one-way hash can be generated from any piece of data, but the data cannot be generated from the hash.



2^{256} possibilities

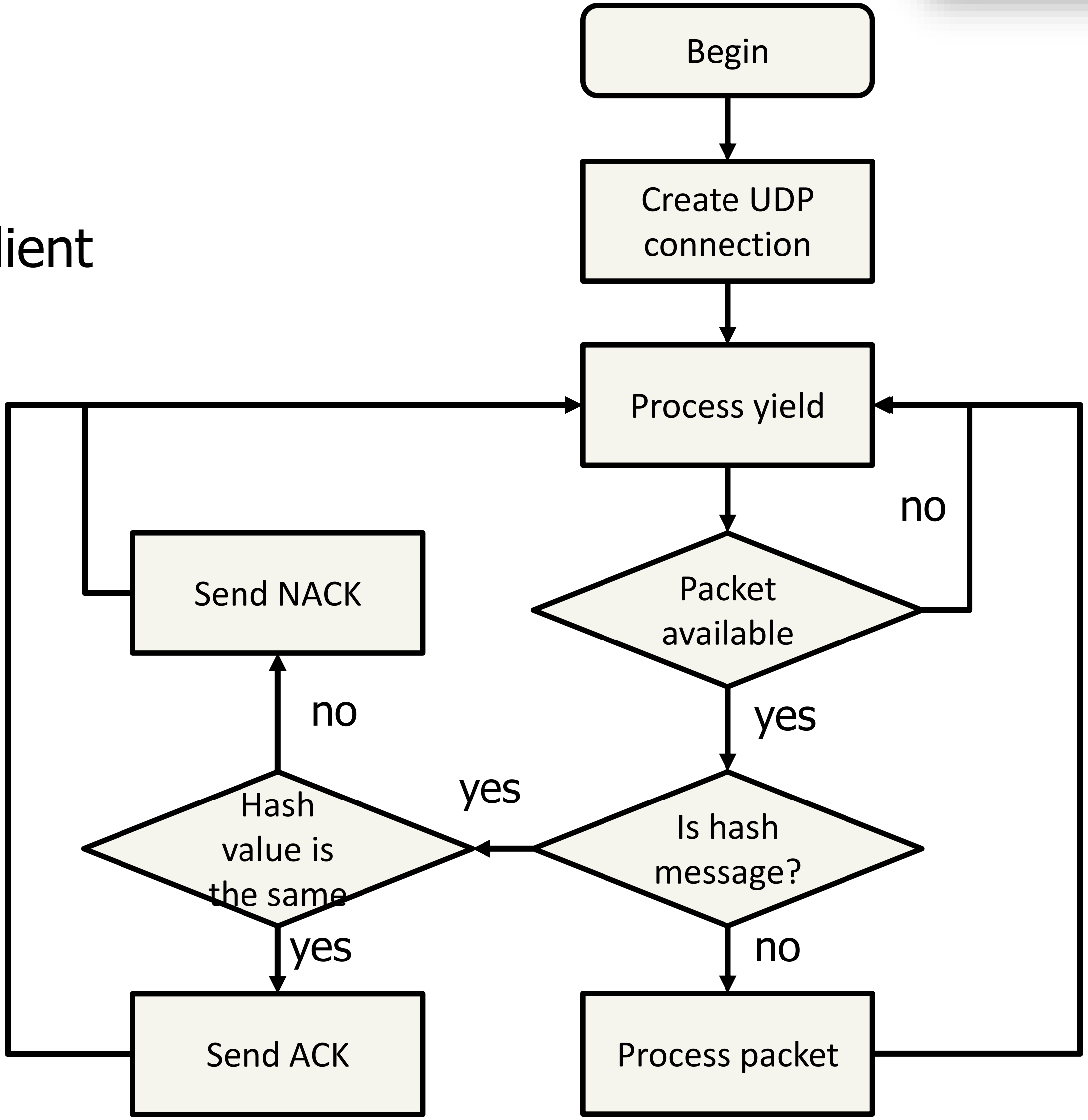
IoT node authentication



UDP Sever

- Create a UDP connection to the client's port.
- Wait for packets from the client.
- Receive hash message
  - Calculate the hash value
  - Compare with the hash value sent from the client

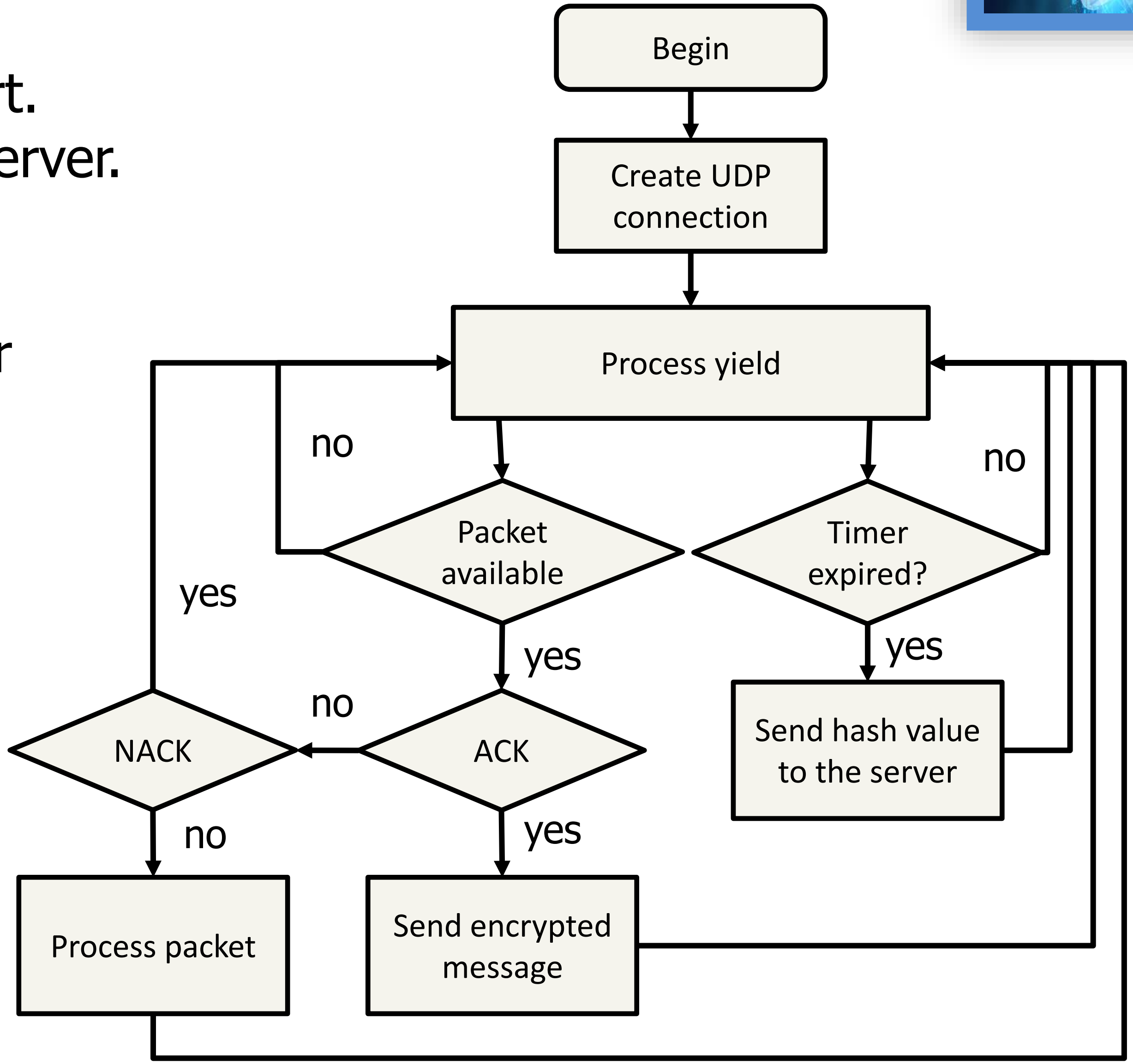
- If the hash values are the same, send ACK to the client. Otherwise, send NACK.
- Receive actual message from the client
  - Decrypt message
  - Print to mote output
  - Send message "Msg Received" to the client.



UDP Client

- Create UDP connection to the server's port.
- If timer expired, send hash value to the server.
- Receive ACK from the server
  - Encrypt actual message
  - Send encrypted message to the server

- Receive NACK from the server, resend hash value.
- Receive confirm message "Msg Received" from the server
  - Print to mote output.



Simulation

- The sample code has been uploaded to Canvas. The code should be in *contiki/examples/ipv6/rpl-udp/*
- See comments in the code to understand the purpose of each function.
- Run simulation with cooja.

```
00:00.657  ID:1  Rime started with address 0.18.116.1.0.1.1.1
00:00.666  ID:1  MAC 00:12:74:01:00:01:01:01 Contiki-2.6-900-ga6227e1 started. Node id is set to 1.
00:00.675  ID:1  CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26, CCA threshold -45
00:00.686  ID:1  Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7401:0001:0101
00:00.688  ID:1  Starting 'UDP client process'
00:00.690  ID:1  UDP client process started
00:00.694  ID:1  Client IPv6 addresses: aaaa::212:7401:1:101
00:00.697  ID:1  fe80::212:7401:1:101
00:00.703  ID:1  Created a connection with the server :: local/remote port 8765/5678
01:09.782  ID:1  sending hash...
01:09.801  ID:1  hash sent : 654d654d7a7a7a7affa371ffa371ffe4fff9ffe4fff9ff8451ff8451225f225f79ffea79ffeafff5ff9afff5ff9a
01:09.935  ID:2  receiving...
01:09.937  ID:2  Received hash from client.
01:09.938  ID:2  Processing...
01:09.946  ID:2  Server sending reply....
01:10.056  ID:1  Server Respone received is 'ACK'
01:10.057  ID:1  Encrypting.......
01:10.059  ID:1  Sending encrypted messsage....
01:10.063  ID:1  Message sent
01:10.175  ID:2  Receiving...
01:10.177  ID:2  Message Received : frqilghqwldo
01:10.179  ID:2  Decrypting.......
01:10.181  ID:2  Decrypted message : confidential
01:10.183  ID:2  Server sending reply....
01:10.306  ID:1  Server Respone received is 'Msg Received'
```

Exercises

1. Change value of the key for Caesar cipher algorithm.
2. Try to implement similar encryption algorithms (e.g., Blowfish, DES, AES.)
3. Try to implement simple hash functions (e.g., Djb2, Adler32).

References

1. Get Started with Contiki: http://www.contiki-os.org/start.html
2. Contiki Tutorial: http://anrg.usc.edu/contiki/index.php/Contiki_tutorials
3. http://practicalcryptography.com/ciphers/caesar-cipher/
4. https://www.movable-type.co.uk/scripts/sha256.html
5. https://en.bitcoinwiki.org/wiki/SHA-256
6. A. L. Colina, A. Vives, A. Bagula, M. Zennaro, and E. Pietrosemoli, “IoT in 5 days” [Online]. Available: http://www.iet.unipi.it/c.vallati/files/IoTinfivedays-v1.1.pdf

Thank
You
Vinaka
köszönöm
Kiitos
Maake
Asante
Shukria
Dhanyavadagalu
감사합니다
Dankscheen
спасибо
Kam Sah Hammida
شكرا
Manana
Dankon
Dank Je
Mauruuru
Biyan
Diolch i Chi
Terima Kasih
Matondo
Taiku
Blagodaram
Ngiyabonga
Dziekuje
Chokrane
Tack
Grazie
Juspaxar
Arigato
Mochchakkeram
Gracias
நன்றி
धन्यवाद
Bedankt
Dakujem
Grazas
Di Ou Mèsi
cảm ơn bạn
Kop Khun Khap
Paldies
Tingki
Gratias Tibi
Ua Tsaug Rau Koj
Obrigado
Kia Ora
Děkuji
Nirringrazzjak
ありがとう
Suksama
Matur Nuwun
Rahmat
谢谢
хвала
Hvala
Welalin
Djiere Dieuf
Misaotra
Danke
Merci
Go Raibh Maith Agat
Eskerrik Asko
Salamat
ขอบขอบคุณคุณ
Najis Tuke