A Paper on Industrial IoT Security
Glenn Frey Olamit
Developing the Industrial Internet of Things I
Sunday, November 27, 2021

Background

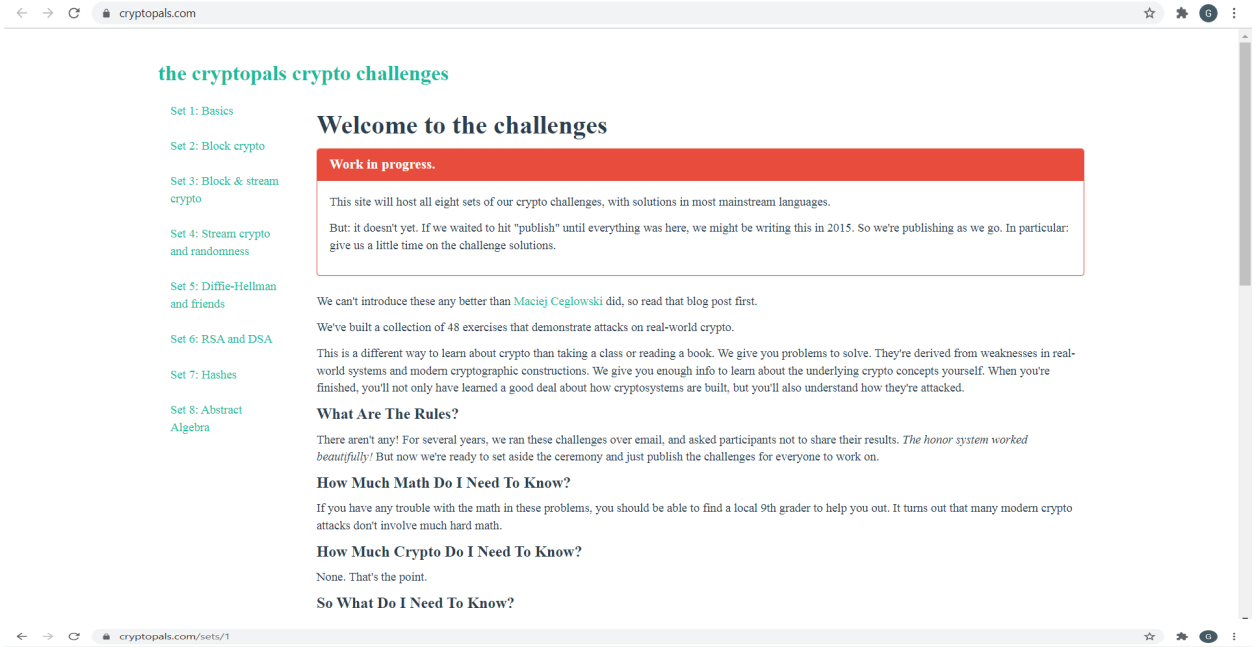
As we are entering a new era of revolution. Namely the 4th industrial revolution. IoT market is estimated to cross 880 billion USD by 2022 and IoT software solutions to hold the largest market in the forecast with a share of 428.7 billion USD followed by services of 408.3 billion USD. Among the software solutions namely network bandwidth management, data management, real-time streaming analytics and remote system monitoring, security solution has the biggest compound annual growth rate of 37.5 percent from 2016 to 2022. It only shows the demand for security solutions is ever growing as companies, government, and people understand the risks of poor security solutions. Among the main challenges are data security and privacy issues. Security plays a vital role in industry 4.0.

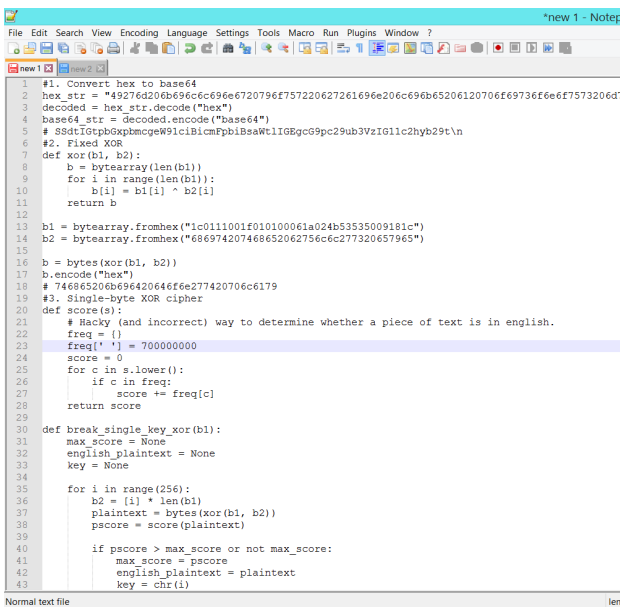
The Industrial internet of things offers a lot of applications. Automotive and Transportations are the top applications to dominate the IoT technology market with a compound annual growth rate of 30.7 percent. Second is industrial with 113.09 billion USD in 2022. Third is healthcare 105.14 billion USD in 2022. Followed by building automation, retail, oil and gas, consumer electronics, agriculture etc. IoT with plenty of applications that it can offer also comes with barriers and risks. What are the key barriers in adopting Industrial IoT? Lack of interoperability or standards are the number one market barrier. Followed closely by security concerns. Then uncertain ROI, legacy equipment, technology immaturity, privacy concerns, lack of skilled workers, and societal concerns. What are the risks of adopting IoT? Security vulnerability due to connectivity to the global network is the number one risk in adopting IoT. Followed by disruption business model or disintermediation. Then privacy breaches. As we see, security plays an important factor in driving the IoT space.

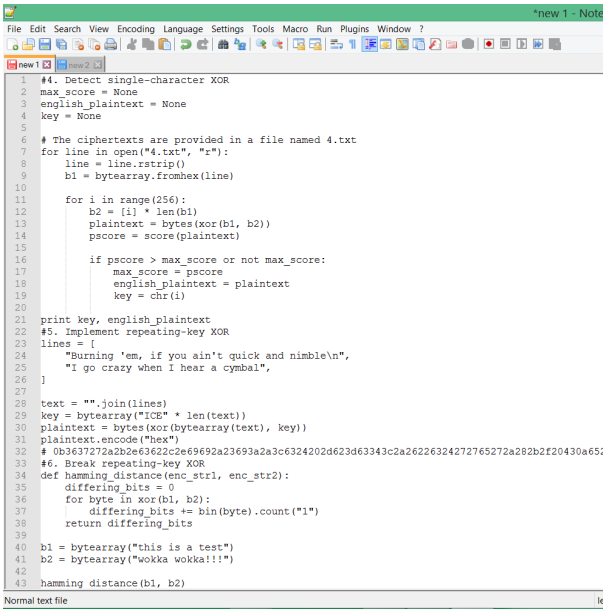
This in turn opens an opportunity for innovative solutions in the IoT space. With Intel ranking number 1 in top players in industrial IoT, followed by IBM and GE. Intel is involved in the design and manufacturing of integrated digital technology which is in the first stage of the IoT ecosystem. Intel is offering customised solutions to its customers to design products for the retail, buildings, transportation, industrial and home market segments. Perhaps we may see these big companies handling more customised hardware solutions in security in the near future.

As an IoT engineer, to be a part of the solution is to understand the nature of the problem. That is understanding how security is implemented and developed, may it be hardware or software solutions. So what does it mean to be secure? The first stage of security is having a secured physical channel. That is no third party should be able to intercept what is communicated between devices. But that is close to impossible as a lot of iot connectivity is shared such as the wireless medium and cables can be tapped. So for us to secure our messages over the channel. We develop encryption techniques to encode and decode the messages transferred across the channel. Encryption, as appealing as it seems, is not perfect as encryption gets sophisticated. Hackers also tried all the methods to decode the original message. With this in mind we try to develop multiple encryption algorithms that are private and public that would take considerable time to break so that even there is an intercepting device capable of reading the

encrypted messages sent across the channel. It will have a hard time decoding the message behind it.



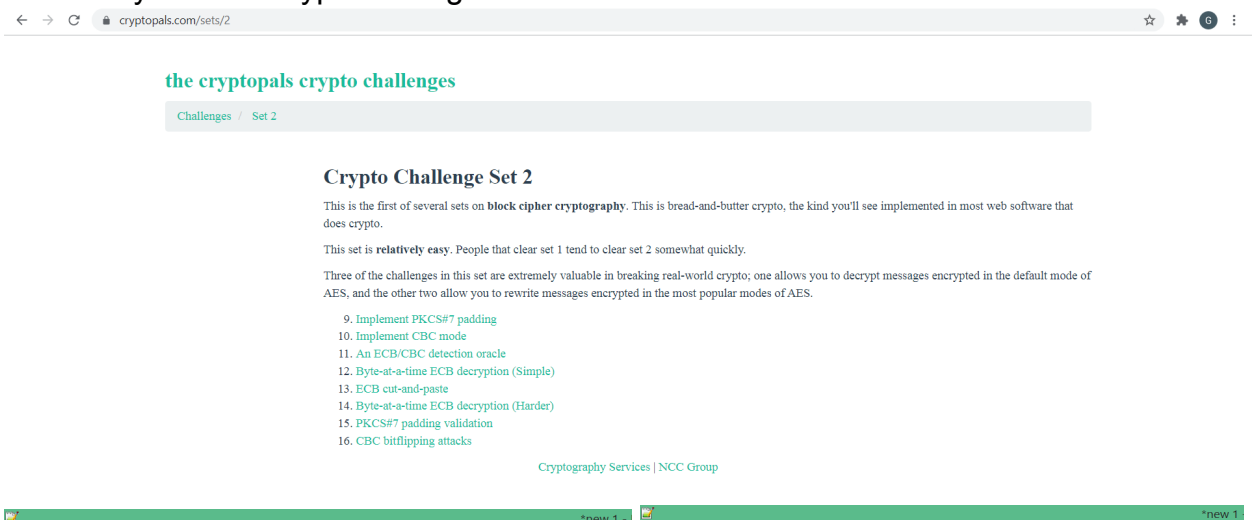


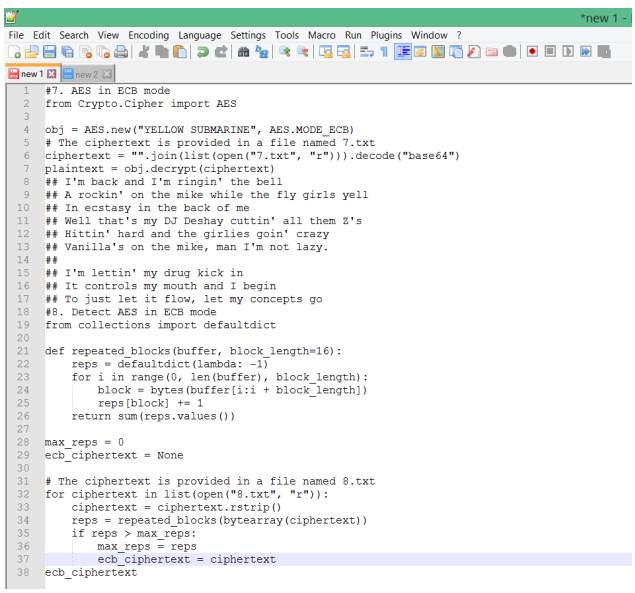


What I did

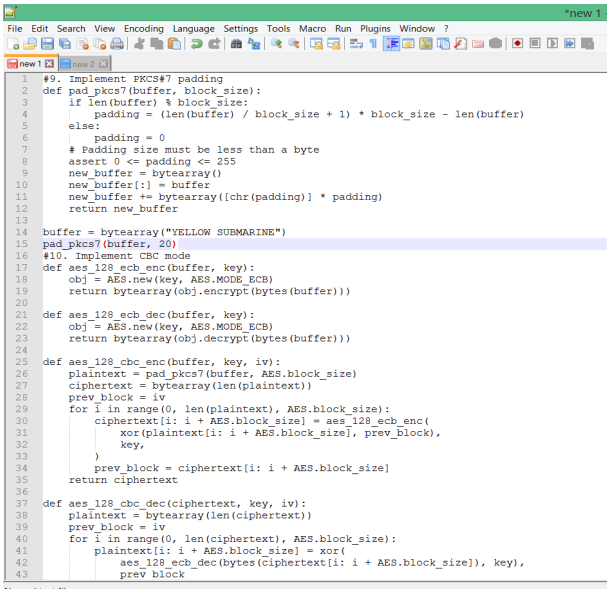
For this paper I chose to explore cryptopals. Cryptopals is a website that has a series of cryptographic challenges ranging from basics to most advanced. You may visit it in http://cryptopals.com/. You will have to write a program for each challenge. You can write your code in the language of your choice but for me I choose python. In this assignment I have to finish set 1(basics) and 2(block crypto) to get a credit. Each set has eight challenges. My first challenge is to convert the hex to base64. I was given a string "49276d206b696c6c696e6720796f757220627261696e206c696b6520612070...". I must be able to convert this string to its base64 that would produce "SSdtIGtpbGxpbmcgeW91ciBicmFpbiBsaWtlIGEgcG9pc29ub3VzIG..". I was able to solve this challenge by converting the hex to binary, then binary to base46. In this example let's use the first three digits of the hex digit for illustrative purposes. The hex started with 492 if we convert this to a binary digit we will get 010010010010. 010010 in base64 is equivalent to S which is the first 6 binary digits in this example. Now converting the seventh to twelfth digit to base64 we get SS which is the first two strings of the answer that must be produced. Now we confirmed our understanding of the problem. The rest is just an implementation of the programming language of your choosing. The second challenge is to XOR inputs 1c0111001f010100061a024b53535.. and 686974207468652062756c6c277320657.. and must produce 746865206b696420646f6e277420706.... This is a pretty straight forward challenge. For illustrative purposes I will be using the first digits of each input to showcase how I solve the problem. So first we convert the hex inputs to equivalent binary format then XOR the inputs then convert the binary format to hex value. 1 is 0001 and 6 is 0100 when XORed will produce 0101 which is equal to seven in hexadecimal format. The third challenge is Single-byte XOR cipher. In this challenge the goal is to find the key and we are given the ciphertext in hexadecimal format which is 1b37373331363f78151b7f2b783431333d78397828372d363c78373e783a393b3736.
There are 256 ASCII characters and we need to convert the to its equivalent binary digit for us to be able to XOR. I try each key and pick the highest frequency comparing it to the english language. The fourth challenge is Detect single-character XOR. This challenge is the opposite of challenge 3 but instead of finding the possible plaintext we have the find the cipher text that are given the text file. The fifth challenge is Implement repeating-key XOR. This challenge is an example of vigenere cipher. We have to encrypt the text "Burning 'em, if you ain't quick and nimble I go crazy when I hear a cymbal" with a repeating key and the output must match 0b3637272a2b2e63622c2e69692a23693a2a3c6324202d623d63343c2a262263242727 6527... This is quite repetitive and the power of programming is most useful at. The sixth challenge is Break repeating-key XOR. In this challenge we need to write the hamming distance; a distance of differing bits between two encoded digits or strings. By using XOR we can identify the strings which will yield a different result. 0 XOR 1 is 1 thus different result. With this in mind we can determine the distance by XORing them then we count the number of bits up. In order for us to break the cipher we need to determine the key. We need then to determine the size of the key. We can use a group of byte size to guess the size of the key. The key sized with the smallest value is used. Using heuristic I found out the key size is 2 bytes for the cypher text provided. The seventh challenge is AES in ECB mode. In this challenge we are working with a real

world encryption standard. I'm sure we are familiar with the word AES. We may not know how it works. But if you have done network security, password, wifi encryption. You have seen this word. We are told that the ciphertext is encrypted using AES-128. Which means it is 128 bits long. This time I imported the pycrypto module and used the key yellow submarine to decrypt the file. And the final challenge in set 1(basics) is Detect AES in ECB mode. The challenge is to find the hexadecimal string in a file that is most likely to be encrypted using AES-128.





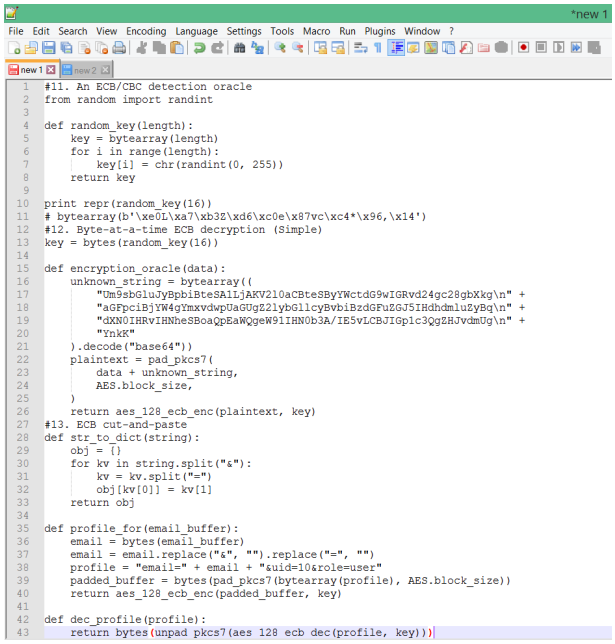
```
#7. AES in ECB mode
from Crypto.Cipher import AES

obj = AES.new("YELLOW SUBMARINE", AES.MODE_ECB)
# The ciphertext is provided in a file named 7.txt
ciphertext = "".join(list(open("7.txt", "r"))).decode("base64")
plaintext = obj.decrypt(ciphertext)
## I'm back and I'm ringin' the bell
## A rockin' on the mike while the fly girls yell
## In ecstasy in the back of me
## Well that's my DJ Deshay cuttin' all them Z's
## Hittin' hard and the girlies goin' crazy
## Vanilla's on the mike, man I'm not lazy.
##
## I'm lettin' my drug kick in
## It controls my mouth and I begin
## To just let it flow, let my concepts go
#8. Detect AES in ECB mode
from collections import defaultdict

def repeated_blocks(buffer, block_length=16):
    reps = defaultdict(lambda: -1)
    for i in range(0, len(buffer), block_length):
        block = bytes(buffer[i:i + block_length])
        reps[block] += 1
    return sum(reps.values())

max_reps = 0
ecb_ciphertext = None

# The ciphertext is provided in a file named 8.txt
for ciphertext in list(open("8.txt", "r")):
    ciphertext = ciphertext.rstrip()
    reps = repeated_blocks(bytearray(ciphertext))
    if reps > max_reps:
        max_reps = reps
        ecb_ciphertext = ciphertext
ecb_ciphertext
```



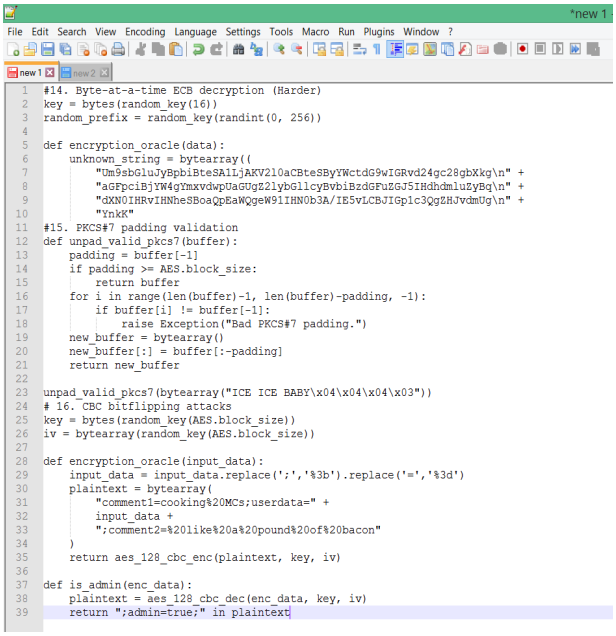
```
#9. Implement PKCS#7 padding
def pad_pkcs7(buffer, block_size):
    if len(buffer) % block_size:
        padding = (len(buffer) / block_size + 1) * block_size - len(buffer)
    else:
        padding = 0
    # Padding size must be less than a byte
    assert 0 <= padding <= 255
    new_buffer = bytearray()
    new_buffer[:] = buffer
    new_buffer += bytearray([chr(padding)] * padding)
    return new_buffer

buffer = bytearray("YELLOW SUBMARINE")
pad_pkcs7(buffer, 20)
#10. Implement CBC mode
def aes_128_ecb_enc(buffer, key):
    obj = AES.new(key, AES.MODE_ECB)
    return bytearray(obj.encrypt(bytes(buffer)))

def aes_128_ecb_dec(buffer, key):
    obj = AES.new(key, AES.MODE_ECB)
    return bytearray(obj.decrypt(bytes(buffer)))

def aes_128_cbc_enc(buffer, key, iv):
    plaintext = pad_pkcs7(buffer, AES.block_size)
    ciphertext = bytearray(len(plaintext))
    prev_block = iv
    for i in range(0, len(plaintext), AES.block_size):
        ciphertext[i: i + AES.block_size] = aes_128_ecb_enc(
            xor(plaintext[i: i + AES.block_size], prev_block),
            key,
        )
        prev_block = ciphertext[i: i + AES.block_size]
    return ciphertext

def aes_128_cbc_dec(ciphertext, key, iv):
    plaintext = bytearray(len(ciphertext))
    prev_block = iv
    for i in range(0, len(ciphertext), AES.block_size):
        plaintext[i: i + AES.block_size] = xor(
            aes_128_ecb_dec(bytes(ciphertext[i: i + AES.block_size]), key),
            prev_block
```

The first challenge in set 2 is Implement PKCS#7 padding. The challenge is to implement PKCS#7 padding. Padding is used when the input messages are irregular in size. We just need to append a number of times to the end of the message such that the buffer length is in multiple of block size. After doing the operation we need to unpad the buffer. The second challenge is Implement CBC mode. The challenge is to encrypt and decrypt using AES-128 in CBC mode. The plaintext is XOR with the ciphertext then. We can check if the operation is correct by reverse direction using ciphertext to arrive at plaintext. The third challenge is An ECB/CBC detection oracle. In this challenge we

have to write a function that randomly encrypt the buffer with AES-128. We also write oracle that determines if the ciphertext is encrypted using CBC or ECB mode. I imported random module to generate random bytes. I then write a function that encrypt a string. I prepend and append 5 to 10 bytes to the plaintext. I encrypt with AES-128 ECB and CBC equally. I then choose random key to encrypt the input. The forth challenge is Byte-at-a-time ECB decryption (Simple). As you might wonder this challenge is getting insane. In this challenge I used the encryption oracle 2.0. Unlike the previous challenge this should not add 5 to 10 random bytes. We then need to determine the block size. I increase the byte size of the plaintext byte-by-byte until the ciphertext increases. The plaintext is padded with multiple of the block size. I then used the ECB function to make sure that the plaintext contains two identical block sizes. Which is “yellow submarineyellow submarine”. Then I look for the unknown string. We can compare the first block of the ciphertext with the plaintext with returned AAAAAAAA if passed. We then detect the unknown string size by cracking the unknown string. The fifth challenge is ECB cut-and-paste. The goal of the challenge is to change the content of the ciphertext which is the result of AES-128 in ECB mode. I created an arbitrary email such that the length of that is of multiple of the block size. I then created a second email with constraints by taking only a ciphertext to the admin profile. The sixth challenge is Byte-at-a-time ECB decryption (Harder). This challenge is pretty much the same as the fourth challenge with decryption being the opposite in a sense. First create a buffer with a block size length. Then append your buffer with itself X times. Pass your input to the encryption oracle to get a ciphertext. If an error occur prepend 1 byte to your input and go back to third steps.The seventh challenge is PKCS#7 padding validation. In this challenge we have to determine if the plaintext has the proper padding. If it does we have to throw an exception. We can accomplish this with the function unpad_pkcs7. The eight and final challenge in set 2 is CBC bit flipping attacks. In this challenge we need to crack the encryption in CBC mode. We need to change some of the cipher from the encryption oracle so that the encryption oracle sees the string. We then start flipping off the bits. We change the byte in the plaintext by changing the byte at the same index of the previous block. By passing the buffer with a throw away block and appending with a "AadminAtrueA" as the user input. We can change the "A" bytes in "AadminAtrueA" to either ";" or "=" using XOR. And this is the end of the final challenge of the second set 2. As we may observe as the challenge is increasing the complexity is harder and the time requires a lot. Understanding the problem and implementing the code are two different skills you need to practice in order to advance in cryptopals.



```
#11. An ECB/CBC detection oracle
from random import randint

def random_key(length):
    key = bytearray(length)
    for i in range(length):
        key[i] = chr(randint(0, 255))
    return key

print repr(random_key(16))
# bytearray(b'\xe0L\xa7\xb3Z\xd6\xc0e\x87vc\xc4*\x96,\x14')
#12. Byte-at-a-time ECB decryption (Simple)
key = bytes(random_key(16))

def encryption_oracle(data):
    unknown_string = bytearray((
        "Um9sbGluJyBpbiBteSAlLjAKV2l0aCBteSByYWctdG9wIGRvd24gc28gbXkg\n" +
        "aGFpciBjYW4gYmxvdwpUaGUgZ2lybGllcyBvbiBzdGFuZGJ5IHdhdmluZyBq\n" +
        "dXN0IHRvIHNheSBoaQpEaWQgeW91IHN0b3A/IE5vLCBJIGp1c3QgZHJvdmUg\n" +
        "YnkK"
    ).decode("base64"))
    plaintext = pad_pkcs7(
        data + unknown_string,
        AES.block_size,
    )
    return aes_128_ecb_enc(plaintext, key)
#13. ECB cut-and-paste
def str_to_dict(string):
    obj = {}
    for kv in string.split("&"):
        kv = kv.split("=")
        obj[kv[0]] = kv[1]
    return obj

def profile_for(email_buffer):
    email = bytes(email_buffer)
    email = email.replace("&", "").replace("=", "")
    profile = "email=" + email + "&uid=10&role=user"
    padded_buffer = bytes(pad_pkcs7(bytearray(profile), AES.block_size))
    return aes_128_ecb_enc(padded_buffer, key)

def dec_profile(profile):
    return bytes(unpad_pkcs7(aes_128_ecb_dec(profile, key)))
```



```
#14. Byte-at-a-time ECB decryption (Harder)
key = bytes(random_key(16))
random_prefix = random_key(randint(0, 256))

def encryption_oracle(data):
    unknown_string = bytearray((
        "Um9sbGluJyBpbiBteSAlLjAKV2l0aCBteSByYWctdG9wIGRvd24gc28gbXkg\n" +
        "aGFpciBjYW4gYmxvdwpUaGUgZ2lybGllcyBvbiBzdGFuZGJ5IHdhdmluZyBq\n" +
        "dXN0IHRvIHNheSBoaQpEaWQgeW91IHN0b3A/IE5vLCBJIGp1c3QgZHJvdmUg\n" +
        "YnkK"
#15. PKCS#7 padding validation
def unpad_valid_pkcs7(buffer):
    padding = buffer[-1]
    if padding >= AES.block_size:
        return buffer
    for i in range(len(buffer)-1, len(buffer)-padding, -1):
        if buffer[i] != buffer[-1]:
            raise Exception("Bad PKCS#7 padding.")
    new_buffer = bytearray()
    new_buffer[:] = buffer[:-padding]
    return new_buffer

unpad_valid_pkcs7(bytearray("ICE ICE BABY\x04\x04\x04\x03"))
# 16. CBC bitflipping attacks
key = bytes(random_key(AES.block_size))
iv = bytearray(random_key(AES.block_size))

def encryption_oracle(input_data):
    input_data = input_data.replace(';','%3b').replace('=','%3d')
    plaintext = bytearray(
        "comment1=cooking%20MCs;userdata=" +
        input_data +
        ";comment2=%20like%20a%20pound%20of%20bacon"
    )
    return aes_128_cbc_enc(plaintext, key, iv)

def is_admin(enc_data):
    plaintext = aes_128_cbc_dec(enc_data, key, iv)
    return ";admin=true;" in plaintext
```

What I Learned

In the week 1 lectures, I learned the breadth and depth of the industrial internet of things. We navigate a list of topics that an internet of things engineer should have knowledge and skills. Areas such as networking, big data analytics, platform, operating system and real-time operating system, wireless communication protocols, security, sensors, file system, machine learning, markets and debugging embedded systems. We then delve into security. We explore simple encryption techniques such as caesar cipher, advanced encryption standard, asymmetric encryption, diffie hellman and hash function. We explore machine learning concepts like linear regression, training data and best fit curve. So basically what it does is trying to model the datas in an equation that best describes it. Then we are introduced to the 4th industrial revolution. Before we only had mechanization, water power and steam power which is the 1st industrial revolution. Then comes the mass production of assembly lines and electricity which is the 2nd industrial revolution. Then we have computers and automation which is the 3rd industrial revolution. And now is the 4th industrial revolution which is a cyber physical system. Industry 4.0 group help develop the 4 design principles which consists of interoperability, information transparency, technical assistance and decentralized decision. We then watch a video lecture of the history of industry 4.0. We are also introduced to factors that enable industry 4.0 which consist of IPV6 and IPV4 address spaces, the dramatic reduction of cost of electronics sensors, compute, bandwidth and storage, power efficient computers and wireless systems, smaller chip size and mature m2m communication. We then look at the influencing factors: the drivers, restraints, opportunities and challenges. We then have a quick tour of industry 4.0 as cyber physical systems which consists of the physical plant and cloud services. Physical plant is where the physical process happens. We connect the embedded sensors to collect data in the process and send data to the platform via wired or wireless connectivity which is then connected to the cloud for further processing and services required such

as machine learning and data analytics. And finally we discussed the market of Internet of Things. In this assignment I was able to review fundamental concepts of the Internet of Things as I am writing the background for this paper.

In the week 2 of the lectures we study the platforms, software solutions and services. Among the platforms introduced is Iotivity by Linux Foundation. It is an open source project for developing standards and certifications. Big companies such as Intel and Samsung are a part of this group. Other platforms mentioned areIBM watsons, Apple Homekit, GE predix, Cisco Jasper and Neura. Amazon and Microsoft today offer their respective platforms. We then discuss the three categorizations of the platform namely device management, network management and application management. We then proceed to a design example 5 story-building. In the example deployment we identify the sensors and actuators in the occupancy, daylight, thermostat, camera, power meter, locks, smoke and gas detector, lighting control and its respective protocol. We then count the number of devices needed. We then look for a platform that will connect and manage all of these together. The platform needs to have components such as API's, Analytics, Data Visualization, Database Integration, Processing Translation, End-node Management, Connectivity of sensors and actuators and lastly security which must be present in all stages and components. We also discussed software solutions for real-time analytics, bandwidth management, remote monitoring, security and data management. Real-time analytics is to transform data into important information to gain insight on the system on demand. Remote monitoring is the monitoring and management of systems remotely. It takes care of the updates of field device, methods of customization, security, and device selection. Network management ensures to increase network efficiency by tracking its bandwidth and informs when outages occur.

In the week 3 of the lectures we learn the markets by application and operating system. There are 5. There are 5 applications that are explored namely automotive and transportation, industrial and manufacturing, building automation, oil and gas and agriculture.

In week 4 of the lectures we learn the networks and wireless communications for the internet of things. We made a short survey of internet service providers. We also discuss the top world players in the cellular IoT market. We revisit the ISO model and TCP/IP model. We revisit the network topology, its physical and logical topology. Network topology can be star, mesh and full-mesh. We learn the network virtualization function which allows the network to dynamically place network capabilities based on usage demands. We learn about the Software Defined Network. The idea is that you can programmatically configure the network using a software application in a centralized location via graphical user interface. We learn the various messaging schemes like polling, interrupts, push status, and publish and subscribe, Polling is used in slow and high latency applications. Interrupts can have high latency due to save and restore operations. In publish and subscribe, the publisher pushes a status when an event occurs to the subscribers. We talk about the leading wireless protocols starting from long range such as cellular, lora, ingenu, and wimax to short range such as ANT+, Bluetooth Smart, Zigbee, Wifi, NFC, EnOcean, Wireless HART, Z-wave and 6LoWPAN.

Cellular has a range of 5-30kmm, depending on the technology the range of speed varies from GSM to UMTS to LTE to 5G. Lorawan is applicable to several kilometer distance and has a speed of 0.3 to 50 kbps. Ingenu can reach up to 50km and is good for low data applications. Wimax has a range of 10 km and has a speed of approximately 70 mbps depending on antenna. Smart Bluetooth has a range of less than 10 meters, speed of 1-2 mbps, capable of mesh and peer to peer topology, and has ultra low power capabilities. ANT+ is ultra low power, flexible for almost any topology, range of 10 meters below, and has speed of 1-2 mbps. Wifi has peer to peer topology, has a range of 20 to 80 meters and speed of 54 to 600 mbps. ZigBee is ultra low power, has mesh topology range of 20 to 80 meters and speed of 20-250 kbps. Wireless HART has mesh topology, range of nominal to 200 meter, speed of 20-250 kbps and is used in industrial wireless sensing. EnOcean is ultra low power, energy harvesting, and self powered. Has peer to peer topologies, range of 30 m indoors and speed of 2 mbps. Z-wave is focused on home automation and can be found in lights, locks, and thermostats. Has peer to peer topology, range of 100m and speed of 40 kbps. 6LowPAN is wireless personal area network, uses IPV6 and mesh topology.
NFC has ultra low to no power, peer to peer, speed of 13.56 mbps and range of 10cm.

In the final week of lecture, we talk about security. What it means to be secure. How to have a secure mindset. Discuss about common encryption algorithms from basics to intermediate. And lastly we talk about black chain.

This Paper and How it Relates to What I Learned in Class

In this assignment I have learned how to code for security encryption techniques from the most basic to intermediate. This is tied to what I learned in the 5th week of this course about security. Where we discussed one of the simplest encryptions, the caesar cipher. In that we shift the corresponding character of the plaintext. Thus making the plaintext unreadable. We also talk about AES or advanced encryption standard which has a key length of 128, 192 and 256. It is believed to be secure but we don't know how to prove it. We also discussed the three modes of AES, XTS, ECB and CBC. We discussed the symmetric and asymmetric encryption. The topics include pretty good privacy, RSA, hash functions, message authentication codes, key generations, key protections, man in the middle attacks, replay attacks, hardware techniques and software techniques, chain of trust, vulnerabilities of bluetooth, certification authentication, and TLS/SSL. All of which is connected to the challenge I encounter in the cryptopals. And among the topics AES is the most common challenge I receive. The class lecture provides me with a basic understanding of the encryption techniques that I used to overcome some of the challenges in the cryptopals. As I went through the challenges I was able to appreciate the methods and techniques used in encryption. I learned the importance of encryption in security in general. I learned data manipulation techniques such as converting binary to hexadecimal, hexadecimal to base46, hexadecimal to ASCII and vice versa. I learned to carefully observe input and output data and to look for patterns. I learned to import modules related to cryptography in the python language. And make the most out of it. Truly the challenge is quite challenging but I learn to be patient and to persevere but it is all worthwhile.