# BISM3205 - Assignment 2

**Part A:**

The Treasure is hidden at [E, 8]

**Part B:**

The Vigenere Cipher was used to decipher the message.

**Part C:**

The key used was **ALEXBISM**.

Regarding the number, it appeared as a 64-length hexadecimal string. This was copied into crackstation.net which had a match on the encrypted letter ‘E’.

He last sentence decoded to *mod(your\_student\_number, 10)*. Since my student number is 45857278, the number modulo 10 is 8. Hence, [E.8] would be the hidden treasure location.

**Part D:**

The hash given in the question is:

694430bed946b0330e4d15e9bc3931123c122166da6d353bad32d4c09da3788c

Using Crackstation.net, this is a known encryption of **universityofqueensland**.

Ordinarily, we would not be able to decrypt a hashed result besides using a brute force technique or looking up hashes in a repository, as we have done.

**Part E:**

The database administrator/security leader at the firm should consider Salting and Peppering the passwords stored in the database. Whilst hashing is secure, salting is the process of addition a random string to it before hashing it.

As we have found in the above example, a simple password and its corresponding hash may already exist in a hash-lookup table, however, a string of random characters appended to a password likely is not. It is highly unlikely that a salted password has been identified in stored in a hash-lookup table.

Peppering passwords is the process of storing the password salt in an object of file storage, instead of in the same database. In the circumstance that a hacker compromises your database, they would be missing the password salt used in combination to originally hash the password.

Besides this, the security leader could consider checking passwords for length and complexity, or checking against dictionaries and deny lists.

**Part F:**

The numbers written in hexadecimal were transformed to base 10 and then looked up on the ASCII table provided in the week 6 tutorial to decipher the question:

*What is it called when 2 strings have the same hash digest?*

This phenomenon is called a collision. As stated, this is when two *different* strings (or any serializable objects) generate the same hash value

**Part G:**

F5 CA 4F 93 5D 44 B8 5C 43 1A 8B F7 88 C0 EA CA

F5CA4F935D44B85C431A8BF788C0EACA

- 32 hexadecimal characters, each representative of 4 bits meaning this image represents a 128-bit stream of characters.

**Part H)**

See Appendix 2 for code:

MD5 hash of the image plane: 253dd04e87492e4fc3471de5e776bc3d

MD5 hash of the image ship: 253dd04e87492e4fc3471de5e776bc3d

**Part I)**

The fact that both images produce the same MD5 hash indicates that MD5 is vulnerable to collisions. This undermines the integrity and reliability of MD5 for ensuring data authenticity.

**Part J)**

This is a trick question. If I have encrypted the message with an asymmetric public key, then I need the private key to decrypt it.

# Q2:

**Part A and B:**

I used a substitution cipher to substitute characters in the initial message as it looked like a website link.

I could decode most of it, except for the two capital letters UV. I assumed it would be QR (for QRcode) but to be certain I wrote a script that every permutation of left over characters I have not substituted. The code snippet and output is provided in Appendix 1. Only <https://alexpudmenzky.com/BISM3205/QRcode.zip> gave me a result.

I downloaded the zip file, extracted it, and found the QR code within it (Figure 1)

Figure 1: QR code found in QRcode.zip

This QR code navigated me to <https://alexpudmenzky.com/BISM3205/sad.jpg>.

Sad face emoji consists of a colon ‘:’ and an opening parenthesis ‘(‘ which on the ASCII table correspond to decimal numbers 58 and 40, respectively.

Combined, this number would be 98, which is a single two-digit number, greater than 0, in decimal notation.

**Part C:**

An IP address is segmented into host and network portions via the use of a subnet mask, which is a series of bits which differentiate which section of the IP address represent the network and which part represent the host.

Bits in the subnet mask that are 1 represent the network portion, so the subnet mask 255.255.128, represented in binary form, i.e.

255.255.128.0 = 11111111.11111111.10000000.00000000 (binary)

Indicating that the first 17 bits represent the network portion of the IP address. A bitwise AND operation can be used to yield the relevant network bits.

11000000.10101000.00000101.00000010 (IP Address) AND

11111111.11111111.10000000.00000000 (Subnet mask)

11000000.10101000.00000000.00000000 (Network portion) = 192.168.0.0

Similarly, a bitwise NAND operation can be used to determine the computer (host) portion of the IP address:

11000000.10101000.00000101.00000010 (IP Address) NAND

11111111.11111111.10000000.00000000 (Subnet mask)

00000000.00000000.00000101.00000010 (Network portion) = 0.0.5.2

**Part D:**

This is more likely an encrypted message.

1. 127\*64 bits – odd length of a hash. Typically an index of 2 – e.g 64, 128, 256 (for SHA-256)
2. Password hashes typically do not need to be this long – even a SHA-256 hash in base 64 (as this appears to be in base 64) would be 64 characters long.

# Q3

**Part A:**

In Kerberos, the client’s request for a TGT and a Service Ticket follow a similar process in terms of their interactions with the KDC but the credentials involved differ.

Main similarities:

* Both TGT and Service ticket maintain the client’s username
* Both are encrypted by private keys maintained by the KDC. For a TGT, it is the TGS secret key, and for the Service Ticket it is the Service Secret Key (Ks)
* Both maintain timestamps to mitigate replay attacks.

Differences:

* The rationale for both the TGT and service ticket are different. A TGT is received from the Authentication Server (AS) component of the KDC and is responsible for verifying the client’s identity during the initial login process. On the other hand, a Service Ticket is handles requests for service-specific tickets using the TGT provided by the client.
* A TGT maintains a Kctgs (shared between the client at TGS) whilst the Service ticket maintains a Kcs (shared between the client and relevant service)

**Part B:**

In Kerberos, the client can't decrypt the TGT or Service Ticket because they're encrypted with secret keys only known to the KDC or the target service. The TGT is encrypted with the KDC’s secret key, and the Service Ticket with the service's secret key, preventing the client from reading or tampering with them.

The client can still use the tickets because they contain **session keys (Kctgs for TGT and Kcs for Service Ticket)**. The client uses these session keys to send encrypted authenticators (with timestamps) to prove their identity when requesting services.

**Part C:**

There are two large mitigating factors that may prohibit a hacker from authenticating to services on a network.

1. If the hacker has retrieved the TGT but does not have the hashed PW (i.e. the Kc) then the hacker is unable to decrypt the Kctgs.
2. If the hacker is able to decrypt the Kctgs, the timestamp embedded into the Kctgs is verified by the KDC that the TGT has been sent within a specified time (default is five minutes) (Microsoft, 2009).

*“If a ticket is compromised, it cannot be used outside of a specified time range -usually short enough to make the risk of a replay attack minimal.”* (Microsoft 2009)

**Part D:**

The best option is (3). Plugging unknown USB sticks into your computer risks introducing malware or ransomware that could compromise your company's network. The IT/security team has the tools and expertise to safely investigate the USB without putting systems at risk. They can check for malware and ensure it’s safe to use.

**Part E:**

A False Positive is when an alert is triggered for legitimate activity mistaken as a threat. This is a nuisance that wastes IT staff's time investigating non-threats, contributing to alert fatigue.

A False Negative is when the IDS fails to detect a real attack. This is far less desirable and serious as actual threats go unnoticed, leaving the system vulnerable.

**Part F:**

If a SQL statement existed in the web application that appeared as such:

SELECT \*

FROM USERS

WHERE 1=1

AND Username = '<enteredUsername>'

Password = '<enteredPassword>'

An attacker could input the following:

Username: *admin*

Password: ‘ OR 1=1 –

Token breakdown:

* The *admin* allows them to login as the admin user
* The single quotation ‘’’ would close off the first quotation in the SQL statement
* The ‘OR 1=1’ is a tautology that returns true
* The ‘*–‘* is a inline comment, which will comment out the original closing bracket.

The result would appear as so:

SELECT \*

FROM USERS

WHERE 1=1

AND Username = 'admin'

Password = '' OR 1=1 -- '

SQL injection attacks the Application layer in the OSI model.

**Part G:**

You can access parts of the deep web using normal browsers like Chrome or Firefox. The deep web includes content not indexed by search engines, such as private databases, email accounts, or subscription sites. However, to access the dark web (a subset of the deep web that requires special anonymisation), you need the Tor browser, which allows you to connect to ".onion" sites not accessible by regular browsers and provides anonymity.

# Question 4

**Part A:**

1. Sender’s end:

Keys:

* A symmetric session key - encrypt email/attachments.
* Recipient’s (asymmetric) public key. Ensure confidentiality - only the recipient can decrypt the message with their private key.
* The sender’s asymmetric private key. This is used as a digital signature to provide integrity/authentication. The hashed digest of the encrypted email and attachments is hashed and encrypted with this private key for the receiver to decrypt.

Certificates:

* The digital certificate to verify the receiver’s public key.

1. Receiver’s end

Keys:

* The sender’s public key to decrypt the hash used in the digital signature. Using the public key of the sender facilitates authentication.
* The receiver’s private key, to decrypt the symmetric session key.
* The sender’s symmetric key, to decrypt the email and attachments.

Certificates:

* The digital certificate to verify the sender’s public key.

**Part B:**

The wireless access point is placed behind a firewall and directly within reach of a web server and database server in the cardholder database environment.

The web and database server should be screened off with a firewall and a NIDS, in addition to network segmentation. Alternatively, remove the wireless access point entirely and make access to this cardholder database environment only capable through the internet.

Could also consider add a honeypot in the demilitarised zone between the firewall and border router.

**Part C:**

Kerberos can *technically* be used for web-based solutions over the internet if there is a trusted KDC that both the client and sever can communicate securely with. However, there are practical limitations:

1. By sending some password to a server (through a web-based application) you are not fully leveraging Kerberos’s security model, which works without transmitting the user’s password.
2. Managing Kerberos tickets and session keys across a globally-distributed user base (e-commerce) would add significant complexity compared to established web standards like HTTPS and TLS.
3. Kerberos main function is to provide authentication, whilst TLS can provide confidentiality also.

**Part D:**

S/MIME operates at the Application Layer (OSI layer 7), securing email messages through encryption and signing.

TLS functions at the Transport Layer (OSI layer 4), providing a secure channel for various protocols, including HTTPS.

**Part E:**

To prepare for the threats posed by quantum computing. Cracking a 128-bit key with modern hardware would take around 500 billion years but quantum computers have significantly reduced this time and could potentially do it in about 185 years.

If quantum computers discover collisions or vulnerabilities in existing algorithms, it could undermine their security and threaten current encryption methods. Since the security of encrypted data relies on the strength of the key and the algorithm, not on keeping the algorithm secret, adopting quantum-resistant solutions now ensures long-term data protection against future risks.

**Part F:**

The primary objective of a security audit is to assess an organisation’s risk management and validate the effectiveness of its security controls. Audits ensure controls are 1) correctly implemented, 2) suited to their purpose, and 3) effectively mitigate associated risks. Through methods like documentation reviews, penetration tests, and performance testing, audits identify vulnerabilities and provide insights into the organisation’s security posture. Post-audit activities, including data analysis and reporting, offer actionable recommendations, security performance metrics, with timelines and risk levels to address gaps.

# Question 5

Part A:

|  |  |
| --- | --- |
| **Number 1-4** | **Area Name** |
| 2 | DMZ |
| 1 | General Staff |
| 4 | Cardholder Environment |
| 3 | Sales Hot desks |
|  |  |
| **Letter (A-M** | **Device Name** |
| A | File Server |
| I | Stateful inspection Firewall |
| H | Wireless Access Point |
| E | Email Server |
| F | Web Proxy |
| K | NIDS |
| L | Database Server |
| J | VPN Server |
| G | Border Router |
| B | Print Server |
| C | Dynamic Filtering Firewall |
| M | Web server |
| D | Dynamic Filtering Firewall (2) |

**Part B:**

**Part C:**

In Bitcoin’s Proof of Work (PoW) mining process, the "Golden Nonce" is a specific value that miners search for to create a new block and earn a reward.

The golden nonce is a 32-bit unsigned integer ‘used once’ and is included in the block header. Miners combine it with other block data and hash it to produce a block hash, aiming to find a nonce that results in a hash less than or equal to the current target difficulty.

The target difficulty measures how hard it is to find a new block. It changes periodically in Bitcoin, to maintain a consistent block mining rate of about every 10 minutes. When a miner finds a Golden Nonce that meets this requirement, they successfully create a valid block for the network.

# Question 6

Part A:

# Appendix 1

Script used to find if there were any ‘hiding’ websites.

import requests

base\_url = "https://alexpudmenzky.com/BISM3205/"

letters\_left = ['F', 'G', 'J', 'Q', 'R', 'V', 'W', 'Z']

for letter1 in letters\_left:

    for letter2 in letters\_left:

        url = base\_url + letter1 + letter2 + 'code.zip'

        try:

            response = requests.get(url)

            # Check if the response status code is 200 (OK)

            if response.status\_code == 200:

                print(f"Valid route found: {url}")

            else:

                print(f"Route not found: {url} (Status code: {response.status\_code})")

        except requests.exceptions.RequestException as e:

            print(f"Error accessing {url}: {e}")

(base) PS C:\git\DanielCiccC.github.io\BISM3205\Assignments\A2> python routes.py

Route not found: https://alexpudmenzky.com/BISM3205/FFcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/FGcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/FJcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/FQcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/FRcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/FVcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/FWcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/FZcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/GFcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/GGcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/GJcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/GQcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/GRcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/GVcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/GWcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/GZcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/JFcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/JGcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/JJcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/JQcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/JRcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/JVcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/JWcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/JZcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/QFcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/QGcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/QJcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/QQcode.zip (Status code: 404)

Valid route found: https://alexpudmenzky.com/BISM3205/QRcode.zip

Route not found: https://alexpudmenzky.com/BISM3205/QVcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/QWcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/QZcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/RFcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/RGcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/RJcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/RQcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/RRcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/RVcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/RWcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/RZcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/VFcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/VGcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/VJcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/VQcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/VRcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/VVcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/VWcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/VZcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/WFcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/WGcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/WJcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/WQcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/WRcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/WVcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/WWcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/WZcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/ZFcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/ZGcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/ZJcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/ZQcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/ZRcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/ZVcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/ZWcode.zip (Status code: 404)

Route not found: https://alexpudmenzky.com/BISM3205/ZZcode.zip (Status code: 404)

# Appendix 2

import hashlib

def calculate\_md5(file\_path):

    """Calculate the MD5 hash of a file."""

    md5\_hash = hashlib.md5()

    with open(file\_path, 'rb') as file:

        for byte\_block in iter(lambda: file.read(4096), b""):

            md5\_hash.update(byte\_block)

    return md5\_hash.hexdigest()

md5\_result = calculate\_md5('plane.jpg')

print(f"MD5 hash of the image plane: {md5\_result}")

md5\_result = calculate\_md5('ship.jpg')

print(f"MD5 hash of the image ship: {md5\_result}")

(base) PS C:\git\DanielCiccC.github.io\BISM3205\Assignments\A2> python .\md\_5\_hash.py

MD5 hash of the image plane: 253dd04e87492e4fc3471de5e776bc3d

MD5 hash of the image ship: 253dd04e87492e4fc3471de5e776bc3d