

ASSIGNMENT 4

CRYPTOGRAPHY

Neilos Kotsiopoulos

neko@itu.dk

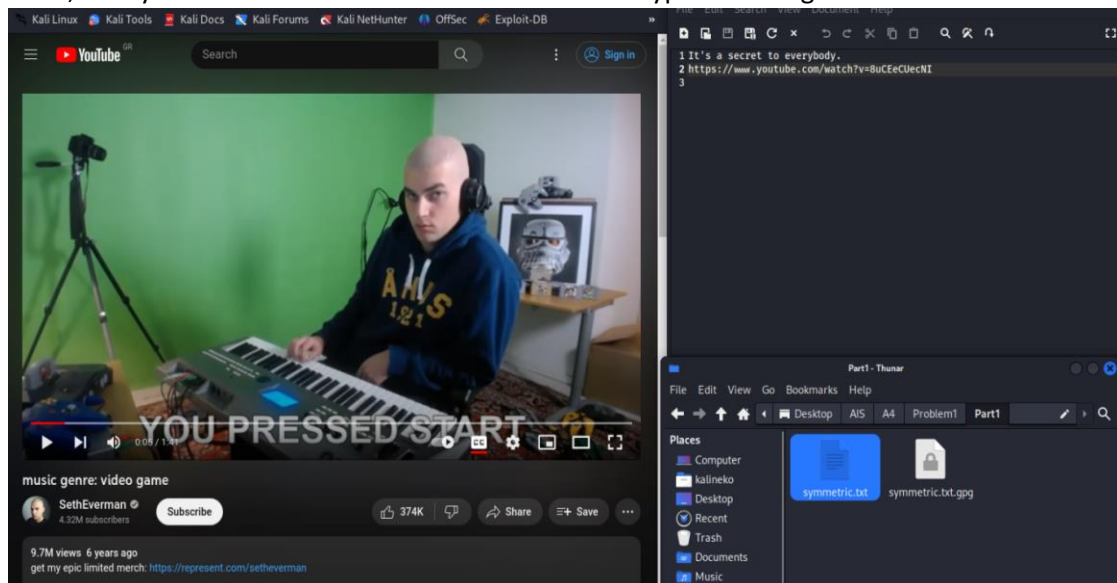
1) Problem 1: Cryptosystems

Part 1: Do the gpg tasks listed under Assignment 4 in LearnIT. Include symmetric.txt, mysymmetric.txt.gpg, mypublic.txt.gpg, and your public key, in your submission

- First, I used the gpg tool in my Kali to decrypt the “symmetric.txt.gpg” using the passphrase “take this” as follows

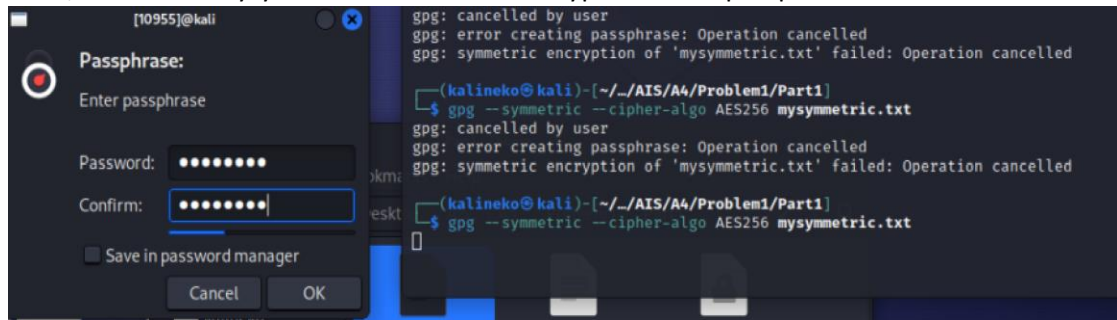
```
(kalineko@kali)-[~/AIS/A4/Problem1/Part1]
$ gpg --output symmetric.txt --decrypt symmetric.txt.gpg
gpg: AES256.CFB encrypted data
gpg: encrypted with 1 passphrase
```

- Then, the symmetric.txt file was created with the decrypted message and link

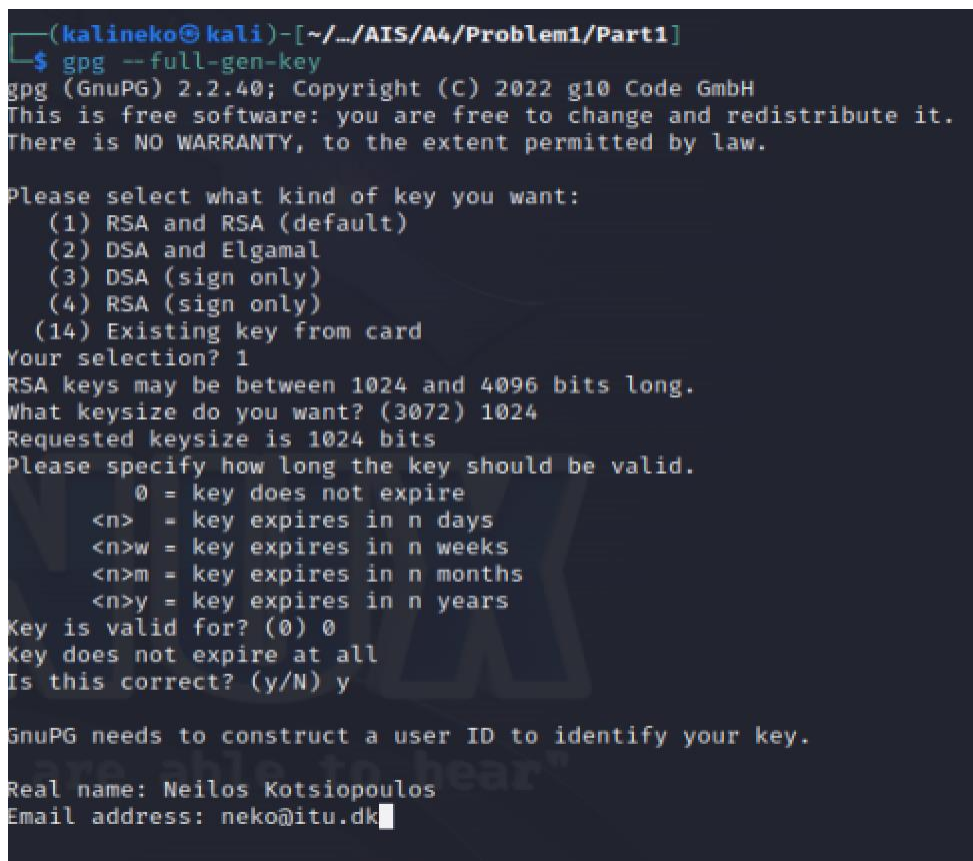


APPLIED INFORMATION TECHNOLOGY – SUMMER COURSE

- Then, I created a mysymmetric.txt file and decrypted it with passphrase “takethis” as follows



- mysymmetric.txt.gpg file was created (included in the submission)
- I created a new public/private key pair by running the command <gpg --full-gen-key> with my name and e-mail



APPLIED INFORMATION TECHNOLOGY – SUMMER COURSE

- Imported Bob's public key

```
(kalineko@kali)-[~/AIS/A4/Problem1/Part1]
$ gpg --import bob_pub.key

gpg: key C1F5AF196A5972A5: public key "Bob Ross <bob@thejoyofpainting.com>" imported
gpg: Total number processed: 1
gpg:          imported: 1

(kalineko@kali)-[~/AIS/A4/Problem1/Part1]
$ cp mysymmetric.txt mypublickey.txt

(kalineko@kali)-[~/AIS/A4/Problem1/Part1]
$ gpg --list-keys
/home/kalineko/.gnupg/pubring.kbx
-----
pub   rsa1024 2023-07-28 [SC]
      7C688E9C92487665D63BA3FC5D0A1D87F23D3486
uid           [ultimate] Neilos Kotsiopoulos <neko@itu.dk>
sub   rsa1024 2023-07-28 [E]

pub   rsa3072 2022-07-27 [SC] [expires: 2024-07-26]
      20383442C4DAFB4BDC58DA67C1F5AF196A5972A5
uid           [ unknown] Bob Ross <bob@thejoyofpainting.com>
sub   rsa3072 2022-07-27 [E] [expires: 2024-07-26]
```

- Encrypt mypublickey.txt with Bob's public key found above

```
(kalineko@kali)-[~/AIS/A4/Problem1/Part1]
$ gpg --encrypt --sign --armor -r 20383442C4DAFB4BDC58DA67C1F5AF196A5972A5 mypublickey.txt

gpg: A80A06D1346D087D: There is no assurance this key belongs to the named user

sub   rsa3072/A80A06D1346D087D 2022-07-27 Bob Ross <bob@thejoyofpainting.com>
      Primary key fingerprint: 2038 3442 C4DA FB4B DC58 DA67 C1F5 AF19 6A59 72A5
      Subkey fingerprint: C952 C77B 3950 02FB 9E13 79B5 A80A 06D1 346D 087D

It is NOT certain that the key belongs to the person named
in the user ID.  If you *really* know what you are doing,
you may answer the next question with yes.

Use this key anyway? (y/N) y

(kalineko@kali)-[~/AIS/A4/Problem1/Part1]
$ mv mypublickey.txt.asc mypublickey.txt.gpg
```

Note: Instead of mypublic.txt, I name my file as mypublickey.txt and the encrypted one as mypublickey.txt.gpg

Followed the commands as per the assignment description and created a private/shared key and bundle into a key container format for PayBud

[illegible]

APPLIED INFORMATION TECHNOLOGY – SUMMER COURSE

Part 3 (PGP): Go to <https://keyserver.ubuntu.com/>, and look up a friend of mine there, freysteinn@freysteinn.com. Click the top result. Explain what you see:

Search results for 'freysteinn@freysteinn.com'

Type	bits/keyID	cr. time	exp time	key expir
pub	rsa4096/55921c9468e36ab22a937d070e2b84019a7db7b0	2017-10-13T13:56:59Z		
uid	Freysteinn Alfredsson <freysteinn@freysteinn.com>			
sig	sig 0e2b84019a7db7b0	2022-10-20T07:19:59Z	2030-10-18T07:19:59Z	[selfsig]
sig	sig 0e2b84019a7db7b0	2021-08-25T17:56:16Z	2025-08-24T17:56:16Z	[selfsig]
sig	sig 0e2b84019a7db7b0	2017-10-23T18:15:33Z	2022-10-14T11:57:35Z	[selfsig]
sig	sig 0e2b84019a7db7b0	2017-10-18T22:45:13Z	2022-10-14T11:57:35Z	[selfsig]
sig	sig f9810b16e49990b4	2017-10-23T10:02:00Z		f9810b16e49990b4
sig	sig f9810b16e49990b4	2017-10-23T10:02:00Z		f9810b16e49990b4
sig	sig cc53d8a7ba575e6e	2017-10-23T14:34:25Z		cc53d8a7ba575e6e
sig	sig cc53d8a7ba575e6e	2017-10-23T14:34:26Z		cc53d8a7ba575e6e
sig	sig 3aeeb373506fde62	2017-10-26T09:44:01Z		3aeeb373506fde62
sig	sig 3aeeb373506fde62	2017-10-26T09:44:02Z		3aeeb373506fde62
sig	sig 702a4b8a73754b1a	2017-10-30T21:55:04Z		702a4b8a73754b1a
sig	sig 702a4b8a73754b1a	2017-10-30T21:55:15Z		702a4b8a73754b1a
sig	sig d3855d5ec0337770	2017-11-02T10:04:16Z		d3855d5ec0337770
sig	sig d3855d5ec0337770	2017-11-02T10:04:21Z		d3855d5ec0337770
sig	sig e5b97020aad283d0	2018-01-06T17:53:13Z		e5b97020aad283d0
sig	sig e5b97020aad283d0	2018-01-06T17:53:14Z		e5b97020aad283d0
sig	sig 07aaa4d95f3d6695	2018-02-24T18:34:49Z		07aaa4d95f3d6695
sig	sig 07aaa4d95f3d6695	2018-02-24T18:34:51Z		07aaa4d95f3d6695
sig	sig c940e9538fe0f8a0	2018-03-21T21:19:53Z		c940e9538fe0f8a0
sig	sig c940e9538fe0f8a0	2018-03-21T21:20:20Z		c940e9538fe0f8a0
sig	sig b9a81dde19b4b16	2021-09-22T14:29:17Z		b9a81dde19b4b16

I can see signatures for the PGP key associated with the e-mail address freysteinn@freysteinn.com.

I can see the type of the signature 'sig', an indication of the signature, the key ID of the signature and the creation & expiration time of the signature

1. Who vouches for whom?

For the signatures indicated as [selfsig], the owner of the keys that vouches for the e-mail key, is also the owner of the e-mail provided. For the rest, all the people or entities who signed the keys are vouching for the identity of freysteinn@freysteinn.com

2. What does [selfsig] mean?

The term [selfsig] refers to a "self-signature" and indicates that the key has been signed by the private key corresponding to the public key itself. This means the key is self-signed, i.e., the owner of the key pair has verified their own identity

3. What does revok mean?

The term [revok] refers to the action of making a key invalid (making it useless) in case it is no longer needed or we want it to be changed

4. Explain Web of Trust in the context of this example

The Web of Trust is a decentralized system where the trustworthiness of a public key is determined by the number and reliability of signatures from other trusted users vouching for the key owner's identity

Part 4 (SSL / TLS): Run the following command

<echo | openssl s_client -connect www.itu.dk:443 | openssl x509 -text>

Explain what you see:

```
(kalineko@kali)-[~/Desktop]
$ echo | openssl s_client -connect www.itu.dk:443 | openssl x509 -text
depth=2 OU = GlobalSign Root CA - R3, O = GlobalSign, CN = GlobalSign
verify return:1
depth=1 C = BE, O = GlobalSign nv-sa, CN = GlobalSign RSA OV SSL CA 2018
verify return:1
depth=0 C = DK, ST = Copenhagen, L = Copenhagen S, O = IT University in Copenhagen, CN = *.itu
dk
verify return:1
DONE
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      79:3a:05:28:8a:51:b2:3a:ac:13:01:71
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C = BE, O = GlobalSign nv-sa, CN = GlobalSign RSA OV SSL CA 2018
    Validity
      Not Before: Mar 28 10:39:09 2023 GMT
      Not After : Apr 28 10:39:08 2024 GMT
    Subject: C = DK, ST = Copenhagen, L = Copenhagen S, O = IT University in Copenhagen, CN = *.itu.dk
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (2048 bit)
      Modulus:
        00:ab:55:35:4b:0c:dd:3e:4e:1d:a4:96:9f:8c:bd:
        5f:68:32:c4:bd:40:cf:a1:11:f0:57:0c:66:9d:71:
        81:b0:58:aa:8b:43:a2:e8:0a:36:37:c6:03:bd:70:
        03:d5:19:f1:f3:23:4c:c4:67:97:95:5c:f5:f6:58:
        92:1a:16:1f:57:d3:60:81:c6:33:b6:0b:7b:db:68:
        30:33:84:4b:8a:b0:10:ae:46:45:4c:3d:db:63:f9:
        30:a1:1e:54:38:63:36:07:aa:74:90:90:89:9b:89:
        83:2d:58:42:e3:7f:11:01:53:0b:0d:ee:6a:01:77:
        15:e9:18:44:04:1a:24:12:2b:0f:8c:a1:bb:9c:d6:
        3d:d6:76:a2:2d:01:ee:a5:fe:e8:c6:3e:48:36:5e:
        7f:01:b8:8f:c5:3c:9e:98:b5:11:f8:19:8d:cc:3a:
        77:23:4c:dd:55:54:4e:8b:aa:d5:5f:1a:c3:1a:9b:
        fa:fb:03:98:b2:7c:3d:24:9d:f8:e2:4e:e7:f2:78:
        0b:11:8c:c3:87:c1:3e:bf:9e:ah:35:c3:40:da:ee:
        0b:11:8c:c3:87:c1:3e:bf:9e:ah:35:c3:40:da:ee:
      Exponent: 65537 (0x10001)
    Extensions: none
    Signature : ecdsa-with-SHA256
      30:46:02:21:00:F3:BE:B1:74:5E:B6:66:03:72:B7:8F:
      5E:FA:87:C4:4A:8C:CC:91:F8:89:93:D8:E3:FF:C3:03:
      DD:CD:36:3E:7A:02:21:00:82:AA:FF:BE:3B:C2:B9:BC:
      A3:AC:6B:BE:4B:0F:9B:D6:78:B2:75:43:45:39:2C:43:
      44:B5:5B:84:45:23:25:00
  Signed Certificate Timestamp:
    Version   : v1 (0x0)
    Log ID    : 48:B0:E3:6B:DA:A6:47:34:0F:E5:6A:02:FA:9D:30:EB:
      1C:52:01:CB:56:DD:2C:81:D9:BB:BF:AB:39:D8:84:73
    Timestamp : Mar 28 10:39:15.912 2023 GMT
    Extensions: none
    Signature : ecdsa-with-SHA256
      30:46:02:21:00:DB:1B:72:A8:C2:90:6D:14:53:1E:4E:
      95:BF:5D:C4:E0:A6:92:AC:A5:F2:74:ED:E7:CF:ED:76:
      0F:26:5D:17:14:02:21:00:87:A5:95:4F:C4:DB:55:A4:
      30:82:50:2C:90:E9:10:31:FB:01:6B:3E:7D:39:2D:CF:
      A1:DA:B3:D2:24:3F:24:04
  Signature Algorithm: sha256WithRSAEncryption
  Signature Value:
    6e:9c:8d:03:00:5b:5c:ac:00:75:0f:59:87:0b:6e:59:ba:22:
    65:e9:79:b8:f5:9f:a6:7c:93:af:d6:9d:9d:5f:34:b1:4f:43:
    33:90:4a:7b:18:54:17:b0:3c:fb:c2:f7:57:14:cc:d1:2f:ac:
    b0:d3:13:25:5f:05:bf:93:3c:55:47:62:3b:97:fc:74:e1:80:
    06:9e:aa:5a:b3:17:65:ef:01:b0:f9:55:89:c9:a8:36:a4:ba:
    28:8e:1d:6b:1f:fb:84:10:2b:11:a6:45:5d:a0:f0:31:d8:2c:
```

I can see the downloaded ITU server's certificate and its details like the issuer, validity period, public key and extensions and how the certificate should be used

APPLIED INFORMATION TECHNOLOGY – SUMMER COURSE

1. What does depth 0, 1, and 2 mean?

Depth=0 refers to the root or starting point (ITU), depth=1 refers to the first level of nodes or elements branching from the root (CA Certificate from GlobalSign nv-sa), and depth 2 refers to the second level of nodes or elements branching from the nodes at depth 1 (GlobalSign Root)

2. What does “Root CA” mean?

stands for Root Certificate Authority, the highest level of trust in a digital certificate hierarchy responsible for issuing digital certificates, in that case GlobalSign

3. What is that Hex-encoded “Modulus”?

It is a large integer in hexadecimal format that's part of a public key in RSA encryption, representing one of two components (along with the exponent) that make up the key

4. Is ITU a certificate authority?

No, ITU is not a certificate authority, it is the subject of the certificate

5. Explain Chain of Trust in the context of this example

In this example, the Chain of Trusts is depicted by the process by which a client (PC) verifies a server's (ITU) identity by validating a series of interconnected digital certificates, each issued by a trusted Certificate Authority (GlobalSign)

6. Now run

< openssl x509 -text -in paybud.crt>

```

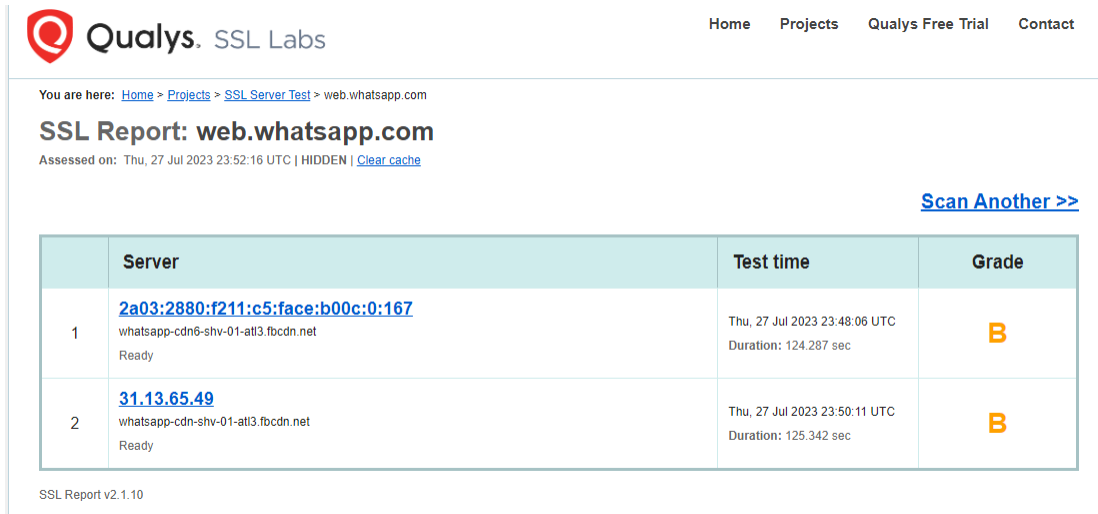
$ openssl x509 -text -in paybud.crt
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      1b:3a:02:61:87:9f:4e:f1:f0:ae:9d:80:b9:9e:bd:56:68:24:dd:b4
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
    Validity
      Not Before: Jul 28 14:08:49 2023 GMT
      Not After : Jul 27 14:08:49 2024 GMT
    Subject: C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (2048 bit)
      Modulus:
        00:a3:91:f4:aa:15:ca:34:6e:26:32:c6:a7:4a:45:
        82:80:b3:88:53:73:68:92:d1:08:20:b2:f1:c1:a8:
        b2:cc:78:6e:82:be:34:d6:44:4e:ab:c5:cb:eb:59:
        5e:85:bd:de:66:96:a0:a7:3e:68:4c:e9:fc:09:23:
        01:de:3d:96:57:5e:81:ac:dc:54:1e:1d:97:b6:ba:
        a9:77:d8:12:fc:28:c5:51:ca:43:0a:1b:1a:af:8d:
        b4:d0:88:a3:39:b2:ad:71:72:9b:3a:df:f7:cf:92:
        57:be:8a:47:2b:18:dd:74:25:b7:de:65:85:9e:dd:
        cd:1f:92:39:34:39:2c:74:a1:8a:b7:1f:b0:87:13:
        6f:31:f9:00:0a:7d:8f:5a:3d:92:66:29:06:da:3c:
        ae:80:69:4c:8c:ac:50:46:e3:27:34:03:08:c7:0f:
        6c:e5:ce:53:f7:f5:6f:ca:2b:5c:50:f2:82:2a:5a:
        d5:50:5a:99:81:bb:b3:33:ae:e5:79:b9:b9:e9:4d:
        7d:bc:02:45:45:8e:b0:83:74:58:94:b1:5b:ed:78:
        6f:fe:1d:29:5f:7e:de:af:2d:90:d1:17:39:18:6b:
        4e:27:51:93:bf:cc:97:8c:d6:a3:9f:1a:21:e5:0f:
        27:89:13:55:16:62:11:9b:4d:9b:40:ee:a0:68:d9:
        f3:27
      Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Subject Key Identifier:
      95:8A:61:77:97:F3:24:FB:79:F0:E0:AC:DA:A7:A9:52:2C:53:19:E2
    X509v3 Authority Key Identifier:
      95:8A:61:77:97:F3:24:FB:79:F0:E0:AC:DA:A7:A9:52:2C:53:19:E2
    X509v3 Basic Constraints: critical
      CA:TRUE
  Signature Algorithm: sha256WithRSAEncryption
  Signature Value:
    99:f7:fb:79:80:09:3f:18:c5:cf:41:64:e7:10:19:38:ef:8c:
    e8:90:a7:a5:f0:84:09:c2:e6:dd:03:f2:fa:13:f7:36:23:d0:
    7d:ef:75:34:f2:63:b9:30:69:05:71:7e:2f:ed:fe:b7:2e:97:
    30:da:aa:c4:41:88:71:60:72:dd:2c:a9:0a:0e:52:06:72:bb:
    25:00:2d:d4:d4:c1:04:62:54:7d:02:7f:79:10:81:30:4e:39:
    0a:76:5e:ad:47:cd:b5:51:8c:04:9f:44:fc:82:d0:47:8a:be:
    7b:26:14:fb:4b:37:d6:67:6f:7b:6c:a9:db:29:5c:78:8b:ac:

```

Who vouches for this key? Is this a root CA?

The entity that vouches for the public key is "Internet Widgits Pty Ltd" as indicated by the Issuer field. This is not a root CA since the Issuer and Subject fields are the same, implying this is a self-signed certificate

Part 5 (rating): Go to <https://www.ssllabs.com/ssltest/>. Type web.whatsapp.com
What are the stated reasons that the scanned server is capped at a B-rating? Are those reasons justified? (What kind of risk is the user at?)



Qualys. SSL Labs

Home Projects Qualys Free Trial Contact

You are here: [Home](#) > [Projects](#) > [SSL Server Test](#) > [web.whatsapp.com](#)

SSL Report: web.whatsapp.com

Assessed on: Thu, 27 Jul 2023 23:52:16 UTC | [HIDDEN](#) | [Clear cache](#)

[Scan Another >>](#)

	Server	Test time	Grade
1	2a03:2880:f211:c5:face:b00c:0:167 whatsapp-cdn6-shv-01-atl3.fbcdn.net Ready	Thu, 27 Jul 2023 23:48:06 UTC Duration: 124.287 sec	B
2	31.13.65.49 whatsapp-cdn-shv-01-atl3.fbcdn.net Ready	Thu, 27 Jul 2023 23:50:11 UTC Duration: 125.342 sec	B

SSL Report v2.1.10

I see the following Protocol reason regarding why the server is capped at a B-rating; "This server supports TLS 1.0 and TLS 1.1. Grade capped to B". It is justified as the TLS protocol indicated is outdated. The above versions have known vulnerabilities like POODLE and CRIME, which can allow attackers to decrypt sensitive information.

2) Problem 2: SSL / TLS

Part 1: Write the two-factor authentication protocol that PayBud now implements, in protocol design syntax. Let U, C and S denote the user, client, and server, respectively. Explain (in English) each step of the protocol. Why does this stop the man-in-the-middle attack from Assignment 2?

Note: if you did not implement two-factor authentication in A3, then write the authentication protocol without a second factor of authentication

Note: instead of including the TLS handshake at the start of your protocol, please assume that a handshake has already been made, and that the browser & server have arrived at a shared secret K. as a shorthand, you can write {m}K for message m encrypted with key K

1. U → C: <userID, password> // User starts login process by entering their user ID and password in the client
2. C → S: <userID, password> // Client sends user's credentials to the server for validation
3. S: If <userID, password> ∈ UserCredentials in DB
then S→C: res with res = "One-time code required"
else S→C: res with res = "Incorrect password"
//Server validates the password; if correct, prompts for a one-time code, else reports incorrect password
4. If res = "One-time code required", S → E: send // If password was correct, server generates a one-time code and sends it to the user's email
5. E → U: one-time code // User's email client delivers the one-time code to the user
6. U → C: one-time code // User inputs the received one-time code into the client
7. C → S: one-time code // Client sends the entered one-time code to the server for validation
8. 10. S: If <userID, one-time-code> is valid
then S -> C: res with res = "Access granted" and consider user as authenticated
else S -> C: res with res = "Access Denied"
//The server validates the one-time code; if it's correct, the user is authenticated, else access is denied
9. C: If server response is "Access granted"
then C -> U: res with res = "Login successful"
else C -> U: res with res = "Incorrect one-time code"
//The client informs the user of the result; either the login was successful or the one-time code was incorrect

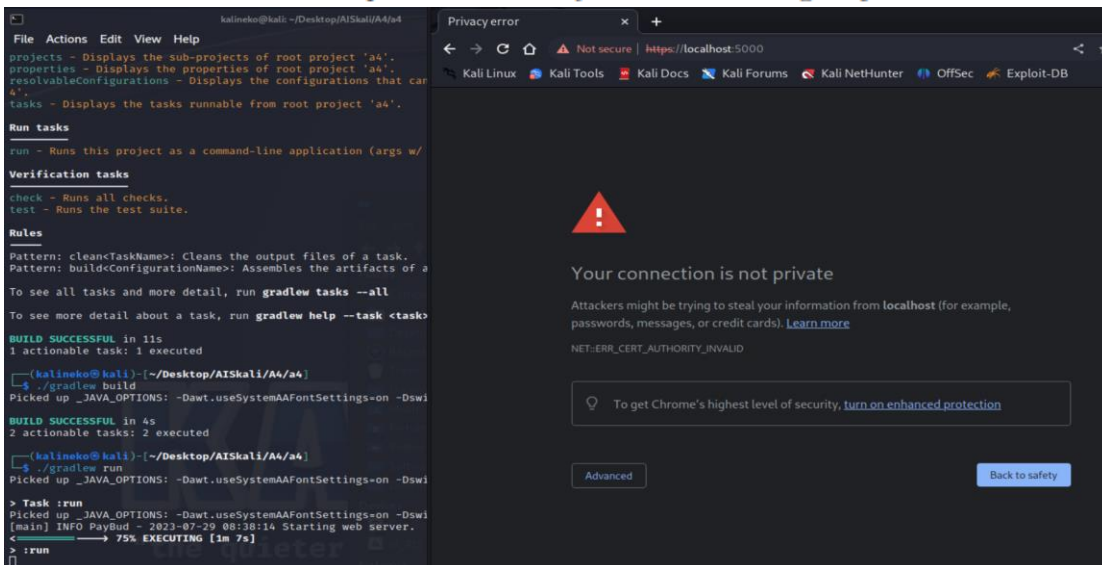
This protocol stops a Man-in-the-Middle (MitM) attack because even if an attacker intercepts the initial credentials (userID and password), they would still need the second factor (one-time code) to authenticate, which is sent via email and is difficult for the attacker to access. This second factor is ephemeral and unique for each session, so stealing it once wouldn't help in successive attacks

APPLIED INFORMATION TECHNOLOGY – SUMMER COURSE

As I got confused by the first note of this assignment, here is a design protocol without the second-factor authentication:

1. U -> C: Initiates login request // The user begins the login process on the client application
2. C -> U: Request for user ID and password // The client requests the user's login credentials (username and password)
3. U -> C: Provides <userID, password> // The user enters their username and password, which is sent to the client
4. C -> S: Sends <userID, password> // The client encrypts the user's credentials and sends it to the server for validation
5. S: Validates <userID, password> // The server checks if the given credentials match a valid user in its database
6. S -> C: Sends validation result // The server sends back a response to the client depending on whether the validation was successful or not
7. C -> U: Delivers validation result // The client relays the server's response to the user. If the validation was successful, the user is logged in. If not, the user is asked to try again.

Place your `paybud.p12` into the PayBud root directory. Start the server, and browse to `https://localhost:5000` (note `https`). Your browser will issue a warning; in Chrome: you'll see "Your connection is not private". That's your browser doing its job.



Part 2: Why do you see this window? Why should you worry about this in general, but be okay with this in our scenario?

I see this window because the browser is designed to warn you when a website's SSL/TLS certificate isn't trusted by a recognized CA. But in our case, I know that I am getting this warning because I am using a self-signed certificate in my local server

Bypass the warning (Chrome: *Advanced* → *Proceed to localhost*). Once PayBud is running, download `https://github.com/drwetter/testssl.sh`, and run it on PayBud:

```
[path/to/]testssl.sh https://localhost:5000
```

APPLIED INFORMATION TECHNOLOGY – SUMMER COURSE

```
(kalineko@kali)~[~/Desktop/AISKali/A4/testssl.sh-3.2]
$ ./testssl.sh https://localhost:5000

#####
testssl.sh      3.2rc2 from https://testssl.sh/dev/

This program is free software. Distribution and
modification under GPLv2 permitted.
USAGE w/o ANY WARRANTY. USE IT AT YOUR OWN RISK!

Please file bugs @ https://testssl.sh/bugs/

#####

Using "OpenSSL 1.0.2-bad (1.0.2k-dev)" [~183 ciphers]
on kali:./bin/openssl.Linux.x86_64
(built: "Sep  1 14:03:44 2022", platform: "linux-x86_64")

Start 2023-07-29 09:59:58          -> 127.0.0.1:5000 (localhost) <-

A record via:      /etc/hosts
rDNS (127.0.0.1):  localhost.
Service detected:  HTTP

Testing protocols via sockets except NPN+ALPN

SSLv2      not offered (OK)
SSLv3      not offered (OK)
TLS 1      offered (deprecated)
TLS 1.1    offered (deprecated)
TLS 1.2    offered (OK)
TLS 1.3    offered (OK): final
NPN/SPDY   not offered
ALPN/HTTP2 not offered
```

Part 3: testssl.sh detected two known vulnerabilities in PayBud. Name and describe each of the two vulnerabilities

```
Testing vulnerabilities

Heartbleed (CVE-2014-0160)      not vulnerable (OK), no heartbeat extension
CCS (CVE-2014-0224)            not vulnerable (OK)
Ticketbleed (CVE-2016-9244), experiment. not vulnerable (OK), reply empty
ROBOT                          not vulnerable (OK)
Secure Renegotiation (RFC 5746) supported (OK)
Secure Client-Initiated Renegotiation VULNERABLE (NOT ok), DoS threat (6 attempts)
CRIME, TLS (CVE-2012-4929)      not vulnerable (OK)
BREACH (CVE-2013-3587)         no gzip/deflate/compress/br HTTP compression (OK) - only supplied
"/" tested
POODLE, SSL (CVE-2014-3566)     not vulnerable (OK), no SSLv3 support
TLS_FALLBACK_SCSV (RFC 7507)   Rerun including POODLE SSL check. Downgrade attack prevention NOT
supported
SWEET32 (CVE-2016-2183, CVE-2016-6329) not vulnerable (OK)
FREAK (CVE-2015-0204)          not vulnerable (OK)
DROWN (CVE-2016-0800, CVE-2016-0703) not vulnerable on this host and port (OK)
led services, see              make sure you don't use this certificate elsewhere with SSLv2 enabled
https://search.censys.io/search?resource=hosts&virtual_hosts=INCLUDE
DE6q=9994F7DC462888460D0316E0417056B272C8F60EFA7A2196CA2FCC6E2D04E6C
LOGJAM (CVE-2015-4000), experimental VULNERABLE (NOT ok): common prime: RFC2409/Oakley Group 2 (1024 bits),
but no DH EXPORT ciphers
BEAST (CVE-2011-3389)          TLS1: ECDHE-RSA-AES256-SHA ECDHE-RSA-AES128-SHA
DHE-RSA-AES256-SHA DHE-RSA-AES128-SHA
AES256-SHA AES128-SHA
VULNERABLE -- but also supports higher protocols TLSv1.1 TLSv1.2
(likely mitigated)            potentially VULNERABLE, uses cipher block chaining (CBC) ciphers with
LUCKY13 (CVE-2013-0169), experimental
ith TLS. Check patches         not vulnerable (OK)
Winshock (CVE-2014-6321), experimental no RC4 ciphers detected (OK)
RC4 (CVE-2013-2566, CVE-2015-2808)
```

Two main vulnerabilities found as follows:

1. **Secure Client – Initiated Renegotiation**

This vulnerability indicates that the server accepts client-initiated renegotiation requests, which could potentially lead to Denial of Service (DoS) attacks – 6 attempts according to testssl

2. **LOGJAM (CVE – 2015 – 4000)**

The server is using a common Diffie-Hellman prime (specifically, the 1024-bit prime specified in RFC 2409, known as Oakley Group 2). This is problematic because if an attacker were to invest sufficient computational resources into factoring this common prime, they would then be able to quickly decrypt any communication that uses it

Bonus: What other weaknesses does testssl.sh reveal in PayBud?

3. **BEAST (CVE-2011-3389)**

The server is susceptible to a BEAST (Browser Exploit Against SSL/TLS) attack, in which an attacker can decrypt data exchanged between a client and server if they are able to execute arbitrary JavaScript code on the client's browser. However, it's noted that the server also supports higher protocols (TLSv1.1 and TLSv1.2), which are not vulnerable to BEAST, making the practical risk lower in modern systems where these protocols are preferred

4. **LUCKY13 (CVE-2013-0169)**

The server may be potentially vulnerable to LUCKY13 attacks, which target the timing differences arising during decryption processing in the TLS protocol. This vulnerability could allow an attacker to recover plaintext information from an encrypted session

Part 4: Fix those two vulnerabilities. Show (with testssl.sh) that your fix was successful. Explain how you modified the code (which methods did you add/modify & how?)

1. **Secure Client – Initiated Renegotiation**

For this vulnerability I could not find a fix, without changing the fundamental setup of the server. I found out that the Java application is using the HttpServer from the **com.sun.net.httpserver** package, which is a simple HTTP server included with Java and that I should use a reverse proxy server such as Nginx or Apache HTTP Server

Otherwise, the following solution was attempted:

```
public static void main(final String... args)
    throws IOException, NoSuchAlgorithmException, KeyManagementException, KeyStoreException, CertificateException, UnrecoverableKeyException {

    String disalg; String passwd; KeyStore keystore; KeyManagerFactory keyman; TrustManagerFactory truman; SSLContext sslctx; HttpsConfigurator c

    //fix Secure Client-Initiated Renegotiation vulnerability
    System.setProperty(key:"jdk.tls.rejectClientInitiatedRenegotiation", value:"true");
```


APPLIED INFORMATION TECHNOLOGY – SUMMER COURSE

2. **LOGJAM (CVE – 2015 – 4000)**

To fix the Logjam vulnerability, I had to ensure that your server is configured to reject Diffie-Hellman groups smaller than 2048 bits. I fixed this vulnerability by inserting the following code in the server code

```
public static void main(final String... args)
    throws IOException, NoSuchAlgorithmException, KeyManagementException, KeyStoreException, CertificateException, UnrecoverableKeyException, NoSuchProviderException, GeneralSecurityException {
    //fix Secure Client-Initiated Renegotiation vulnerability
    System.setProperty(key:"jdk.tls.rejectClientInitiatedRenegotiation", value:"true");

    //fix LOGJAM vulnerability
    System.setProperty(key:"jdk.tls.ephemeralDHKeySize", value:"2048");
}
```

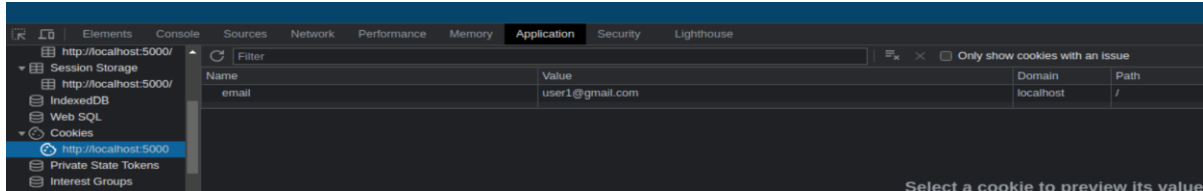
As we can see, both issues are not highlighted as vulnerable items in the new testssl!

Testing vulnerabilities	
Heartbleed (CVE-2014-0160)	not vulnerable (OK), no heartbeat extension
CCS (CVE-2014-0224)	not vulnerable (OK)
Ticketbleed (CVE-2016-9244), experiment.	not vulnerable (OK), reply empty
ROBOT	not vulnerable (OK)
Secure Renegotiation (RFC 5746)	supported (OK)
Secure Client-Initiated Renegotiation	not vulnerable (OK)
CRIME, TLS (CVE-2012-4929)	not vulnerable (OK)
BREACH (CVE-2013-3587)	no gzip/deflate/compress/br HTTP compression (OK) - only supplied
"/" tested	
POODLE, SSL (CVE-2014-3566)	not vulnerable (OK), no SSLv3 support
TLS_FALLBACK_SCSV (RFC 7507)	Rerun including POODLE SSL check. Downgrade attack prevention NOT
supported	
SWEET32 (CVE-2016-2183, CVE-2016-6329)	not vulnerable (OK)
FREAK (CVE-2015-0204)	not vulnerable (OK)
DROWN (CVE-2016-0800, CVE-2016-0703)	not vulnerable on this host and port (OK)
	make sure you don't use this certificate elsewhere with SSLv2 enabled services, see
DE8q-9994F7DC462888460D0316E0417056B272C8F60EFA7A2196CA2FCC6E2D04E6C	https://search.censys.io/search?resource=hosts&virtual_hosts=INCLUDE
LOGJAM (CVE-2015-4000), experimental	common prime with 2048 bits detected: RFC7919/ffdhe2048 (2048 bits)
	but no DH EXPORT ciphers
BEAST (CVE-2011-3389)	TLS1: ECDHE-RSA-AES256-SHA ECDHE-RSA-AES128-SHA DHE-RSA-AES256-SHA DHE-RSA-AES128-SHA AES256-SHA AES128-SHA VULNERABLE -- but also supports higher protocols TLSv1.1 TLSv1.2
(likely mitigated)	
LUCKY13 (CVE-2013-0169), experimental	potentially VULNERABLE, uses cipher block chaining (CBC) ciphers with TLS. Check patches
Winshock (CVE-2014-6321), experimental	not vulnerable (OK)
RC4 (CVE-2013-2566, CVE-2015-2808)	no RC4 ciphers detected (OK)

3) Problem 4: Hashing

Part 1: What vulnerability (in PayBud) is this hack exploiting?

This hack is exploiting a vulnerability in PayBud's session management where authentication information (the user's email in this case) is stored in a client-side cookie without proper protection - unencrypted, thus allowing anyone to forge a cookie and impersonate another user



Note: As per the Slack's thread, I was unable to cookie-hijack the account, so I continued with the A3 server

Part 2: Show how the logging you implemented in Assignment 3 can reveal that this vulnerability is being exploited?

The logging that I had in place did not exploit this vulnerability, I modified the code as following so cookie hijacking could be detected

```
private static void logout(final HttpExchange io) {
    Logger logger = LoggerFactory.getLogger(clazz:HttpServer.class);
    if (! authenticated(io)) {
        respond(io, response_code:409, mime:"application/json", json(text:"Not logged in.));
        logger.warn(format:"Unauthorized attempt to logout without being logged in from IP: {}", io.getRemoteAddress().toString());
        return;
    }

    deauthenticate(io);
    respond(io, response_code:200, mime:"application/json", json(text:"Logout successful.));
    logger.info(format:"Successful logout for user {} from IP: {}", getEmail(io), io.getRemoteAddress().toString());
}
```

```
private static boolean hasCookie(final HttpExchange io){
    Logger logger = LoggerFactory.getLogger(clazz:HttpServer.class);
    logger.warn(dateformat.format(new Date()) + "Unauthorized attempt to access a protected resource without a cookie from IP: {}", io.getRemoteAddress().toString());
    return io.getRequestHeaders().containsKey(key:"Cookie");
}
```

Now the logger, can potentially exploit cookie hijacking vulnerabilities, in the below snippet we can see that I successfully logged in and logged out by inserting user1@gmail.com credentials and then I cookie hijacked this account:

APPLIED INFORMATION TECHNOLOGY – SUMMER COURSE

```
> Task :run
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
[main] INFO PayBud - 2023-07-29 12:11:51 Starting web server.
[HTTP-Dispatcher] WARN com.sun.net.httpserver.HttpServer - 2023-07-29 12:11:59Unauthorized att
empt to access a protected resource without a cookie from IP: /[0:0:0:0:0:0:1]:59644
[HTTP-Dispatcher] WARN com.sun.net.httpserver.HttpServer - 2023-07-29 12:12:09Unauthorized att
empt to access a protected resource without a cookie from IP: /[0:0:0:0:0:0:1]:59650
[HTTP-Dispatcher] INFO com.sun.net.httpserver.HttpServer - Successful login for user user1@gma
il.com from IP: /[0:0:0:0:0:0:1]:59650
[HTTP-Dispatcher] WARN com.sun.net.httpserver.HttpServer - 2023-07-29 12:12:09Unauthorized att
empt to access a protected resource without a cookie from IP: /[0:0:0:0:0:0:1]:59650
[HTTP-Dispatcher] WARN com.sun.net.httpserver.HttpServer - 2023-07-29 12:12:09Unauthorized att
empt to access a protected resource without a cookie from IP: /[0:0:0:0:0:0:1]:59650
[HTTP-Dispatcher] WARN com.sun.net.httpserver.HttpServer - 2023-07-29 12:12:13Unauthorized att
empt to access a protected resource without a cookie from IP: /[0:0:0:0:0:0:1]:59650
[HTTP-Dispatcher] INFO com.sun.net.httpserver.HttpServer - 2023-07-29 12:12:13Successful logou
t for user user1@gmail.com from IP: /[0:0:0:0:0:0:1]:59650
[HTTP-Dispatcher] WARN com.sun.net.httpserver.HttpServer - 2023-07-29 12:12:13Unauthorized att
empt to access a protected resource without a cookie from IP: /[0:0:0:0:0:0:1]:59650
[HTTP-Dispatcher] WARN com.sun.net.httpserver.HttpServer - 2023-07-29 12:12:13Unauthorized att
empt to access a protected resource without a cookie from IP: /[0:0:0:0:0:0:1]:59644
[HTTP-Dispatcher] WARN com.sun.net.httpserver.HttpServer - 2023-07-29 12:12:13Unauthorized att
empt to access a protected resource without a cookie from IP: /[0:0:0:0:0:0:1]:59644
[HTTP-Dispatcher] WARN com.sun.net.httpserver.HttpServer - 2023-07-29 12:12:24Unauthorized att
empt to access a protected resource without a cookie from IP: /[0:0:0:0:0:0:1]:59652
[HTTP-Dispatcher] WARN com.sun.net.httpserver.HttpServer - 2023-07-29 12:12:24Unauthorized att
empt to access a protected resource without a cookie from IP: /[0:0:0:0:0:0:1]:59652
[HTTP-Dispatcher] WARN com.sun.net.httpserver.HttpServer - 2023-07-29 12:12:24Unauthorized att
empt to access a protected resource without a cookie from IP: /[0:0:0:0:0:0:1]:59650
<===== 75% EXECUTING [41s]
> :run
```

Part 3: Fix this vulnerability, by protecting the integrity of the session cookie. Explain how you modified the code (which methods did you add/modify & how?)

I implemented the simplest solution provided by the stackoverflow link, before the creation of cookies method in the WebServer.java class

```
/*Hashing cookies */

private static final String HMAC_SHA512 = "HmacSHA512";

private static String toHexString(byte[] bytes) {
    try (Formatter formatter = new Formatter()) {
        for (byte b : bytes) {
            formatter.format(format:"%02x", b);
        }
        return formatter.toString();
    }
}

public static String calculateHMAC(final String email, String key)
    throws SignatureException, NoSuchAlgorithmException, InvalidKeyException {
    SecretKeySpec secretKeySpec = new SecretKeySpec(key.getBytes(), HMAC_SHA512);
    Mac mac = Mac.getInstance(HMAC_SHA512);
    mac.init(secretKeySpec);
    return toHexString(mac.doFinal(email.getBytes()));
}
```

APPLIED INFORMATION TECHNOLOGY – SUMMER COURSE

Then, I had to implement the calculateHMAC in the createCookie method as follows:

```
private static void createCookie(final HttpExchange io, final String email){
    String hashString = "";
    try{
        String hmac = calculateHMAC(email, key:"password");
        hashString = hmac;
    } catch (Exception e){
        Logger logger = LoggerFactory.getLogger(clazz:HttpServer.class);
        logger.error(dateformat.format(new Date()) + "Error while creating cookie for user {} from IP: {}", email, io.getRemoteAddress().toString());
    }

    List<String> l = new ArrayList<String>();
    l.add("email=" + email + "; path=/");
    l.add("hash=" + hashString + "; path=/");
    io.getResponseHeaders().put(key:"Set-Cookie", l);
}
```

The same has applied to the goodCookie method

```
private static boolean goodCookie(final HttpExchange io){
    String hashString = "";
    final Optional<String> result = DB.user(getEmail(io));

    try{
        String hmac = calculateHMAC(getEmail(io), key:"password");
        hashString = hmac;
    } catch (Exception e) {
        Logger logger = LoggerFactory.getLogger(clazz:HttpServer.class);
        logger.error(dateformat.format(new Date()) + "Error while creating cookie for user {} from IP: {}", getEmail(io), io.getRemoteAddress().toString());
    }

    final boolean userSuccess = (result != null);
    if ( ! userSuccess ){
        return false; // SQL query failed
    }

    final boolean userExists = result.isPresent();
    if ( ! userExists ){
        return false; // user given in cookie does not exist in PayBud
    }

    final boolean hashGood = getHash(io).equals( hashString );
    if ( ! hashGood ){
        return false; // hash given in cookie failed integrity check
    }

    return true;
}
```

The hints inside the A4 server code helped a lot, in order to identify where the cookie hashing had to be implemented

Then, I generated and stored a HmacSHA512 secret key to paybud.p12

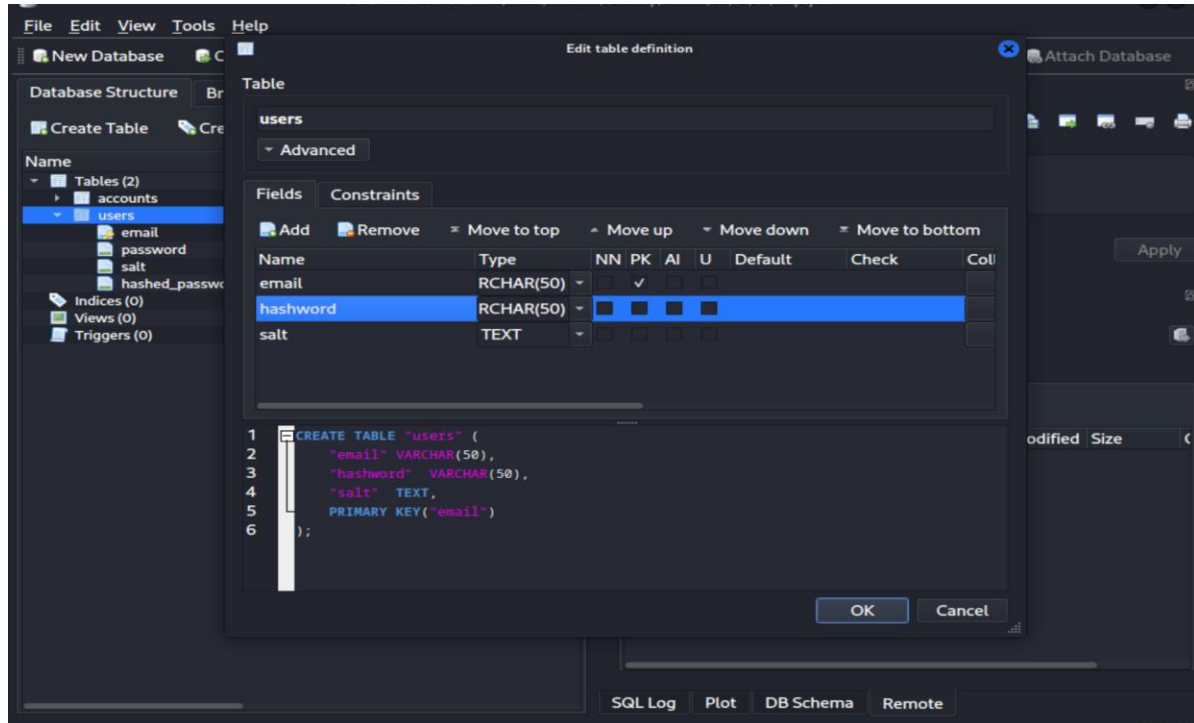
```
—(kalineko@kali)-[~/Desktop/AISkali/A4/a4]
—$ keytool -genseckey -keystore paybud.p12 -storetype pkcs12 -keyalg HMacSHA512 -keysize 2048
-alias HMACkey -keypass password
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Enter keystore password:
Generated 2,048-bit HmacSHA512 secret key
```


4) Problem 4: Password Storage

An attacker that gains access to the PayBud database (e.g. via. SQL injection) obtains all passwords, and thus gains access to all user accounts.

Part 1: Fix this problem, by storing passwords securely. Follow the instructions in Schneider, Section 5.1.2, to store only salted hashed passwords. Explain how you modified the code (which methods did you add/modify & how?)

First, I added the column "salt" and replaced the column password with the column the "hashword" at the empty and paybud DBs



Then, I added a method to generate salts and a hashcode to create DB

```
public class DB {

    private static final String URL = "jdbc:sqlite:db/paybud.db"; // database we are connecting to.

    public static boolean create( final String email, final String password ) throws NoSuchAlgorithmException, InvalidKeySpecException {

        SecureRandom random = new SecureRandom();
        byte[] salt = new byte[16];
        random.nextBytes(salt);

        KeySpec spec = new PBEKeySpec(password.toCharArray(), salt, 65536, 128);
        SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
        byte[] hashword = factory.generateSecret(spec).getEncoded();

        final String iu = "INSERT INTO users VALUES ('" + email + "', '" + hashword + "', '" + salt + "')";
        final String ia = "INSERT INTO accounts VALUES ('" + email + "', '0')";
        try {
```

APPLIED INFORMATION TECHNOLOGY – SUMMER COURSE

But when tried to create a new user, the server crashed and I received a cookie hijacking warning error

```

46
[pool-1-thread-2] WARN com.sun.net.httpserver.HttpServer - 2023-07-29 16:53:54Unauthorized att
empt to access a protected resource without a cookie from IP: localhost/[0:0:0:0:0:0:1]:3926
2
[pool-1-thread-4] WARN com.sun.net.httpserver.HttpServer - 2023-07-29 16:53:54Unauthorized att
empt to access a protected resource without a cookie from IP: localhost/[0:0:0:0:0:0:1]:5258
8
[pool-1-thread-8] WARN com.sun.net.httpserver.HttpServer - 2023-07-29 16:54:06Unauthorized att
empt to access a protected resource without a cookie from IP: localhost/[0:0:0:0:0:0:1]:4054
8
org.sqlite.SQLiteException: [SQLITE_ERROR] SQL error or missing database (no such column: pass
word)
    at org.sqlite.core.DB.newSQLException(DB.java:941)
    at org.sqlite.core.DB.newSQLException(DB.java:953)
    at org.sqlite.core.DB.throwex(DB.java:918)
    at org.sqlite.core.NativeDB.prepare_utf8(Native Method)
    at org.sqlite.core.NativeDB.prepare(NativeDB.java:134)
    at org.sqlite.core.DB.prepare(DB.java:257)
    at org.sqlite.core.CorePreparedStatement.<init>(CorePreparedStatement.java:47)
    at org.sqlite.jdbc3.JDBC3PreparedStatement.<init>(JDBC3PreparedStatement.java:30)
    at org.sqlite.jdbc4.JDBC4PreparedStatement.<init>(JDBC4PreparedStatement.java:19)
    at org.sqlite.jdbc4.JDBC4Connection.prepareStatement(JDBC4Connection.java:35)
    at org.sqlite.jdbc3.JDBC3Connection.prepareStatement(JDBC3Connection.java:241)
    at org.sqlite.jdbc3.JDBC3Connection.prepareStatement(JDBC3Connection.java:205)
    at paybud.DB.login(DB.java:60)
    at paybud.WebServer.login(WebServer.java:286)
    at paybud.WebServer.lambda$main$8(WebServer.java:122)
    at jdk.httpserver/com.sun.net.httpserver.Filter$Chain.doFilter(Filter.java:95)
    at jdk.httpserver/sun.net.httpserver.AuthFilter.doFilter(AuthFilter.java:82)
    at jdk.httpserver/com.sun.net.httpserver.Filter$Chain.doFilter(Filter.java:98)
    at jdk.httpserver/sun.net.httpserver.ServerImpl$Exchange$LinkHandler.handle(ServerImpl
.java:851)
    at jdk.httpserver/com.sun.net.httpserver.Filter$Chain.doFilter(Filter.java:95)

```

Tried to use the <paybud.db> file provided in slack, but got a similar error. It seems that the code should be inserted in a different piece of code

```

(kalineko@kali)-[~/Desktop/a4_problem4]
$ ./gradlew run
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true

> Task :run
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
[main] INFO PayBud - 2023-07-30 10:22:21 Starting web server.
[pool-1-thread-3] WARN com.sun.net.httpserver.HttpServer - 2023-07-30 10:22:38Unauthorized att
empt to access a protected resource without a cookie from IP: localhost/[0:0:0:0:0:0:1]:5639
4
[pool-1-thread-11] WARN com.sun.net.httpserver.HttpServer - 2023-07-30 10:22:48Unauthorized at
tempt to access a protected resource without a cookie from IP: localhost/[0:0:0:0:0:0:1]:564
32
[pool-1-thread-12] WARN com.sun.net.httpserver.HttpServer - 2023-07-30 10:22:50Unauthorized at
tempt to access a protected resource without a cookie from IP: localhost/[0:0:0:0:0:0:1]:564
32
org.sqlite.SQLiteException: [SQLITE_ERROR] SQL error or missing database (no such table: users
)
    at org.sqlite.core.DB.newSQLException(DB.java:941)
    at org.sqlite.core.DB.newSQLException(DB.java:953)
    at org.sqlite.core.DB.throwex(DB.java:918)
    at org.sqlite.core.NativeDB.prepare_utf8(Native Method)
    at org.sqlite.core.NativeDB.prepare(NativeDB.java:134)
    at org.sqlite.core.DB.prepare(DB.java:257)

```

(no such table: users)

APPLIED INFORMATION TECHNOLOGY – SUMMER COURSE

Part 2: PayBud is vulnerable to an Offline Dictionary Attack. Exploit this vulnerability by following the instructions provided alongside this assignment

I was not able to modify the DB in such a way to encrypt the passwords, thus I was unable to exploit the vulnerability by performing an offline dictionary attack, as per the abstract Assignments Notes

- offline dictionary attack
 - if you want to dictionary attack PayBud, modify the following command accordingly. (delete `~/john/john.pot` to re-crack the hashes). here's [users.txt](#).

```
$ john --wordlist=/usr/share/john/password.lst users.txt
```

Why would the attack have been even easier if we had only hashed, and not salted, the passwords? (i.e. what is the purpose of salt?)

Salting passwords before hashing them prevents offline dictionary attacks (and rainbow table attacks) by ensuring that even if two users have the same password, their hashes will be different due to the unique salts, thus making it more difficult for an attacker to precompute hash values for common passwords and simply looking for matches in the database – the dictionaries are becoming huge

Part 3: Fix this vulnerability, by enforcing a password policy, to disallow weak passwords. Define a policy; what is a valid password according to your policy?

Since my server is cracked I cannot continue with the implementation of this step. So, I implemented the password policy in the working version of the server without the hashed passwords, by changing the code in DB.java – my password policy can be seen in the below screenshot

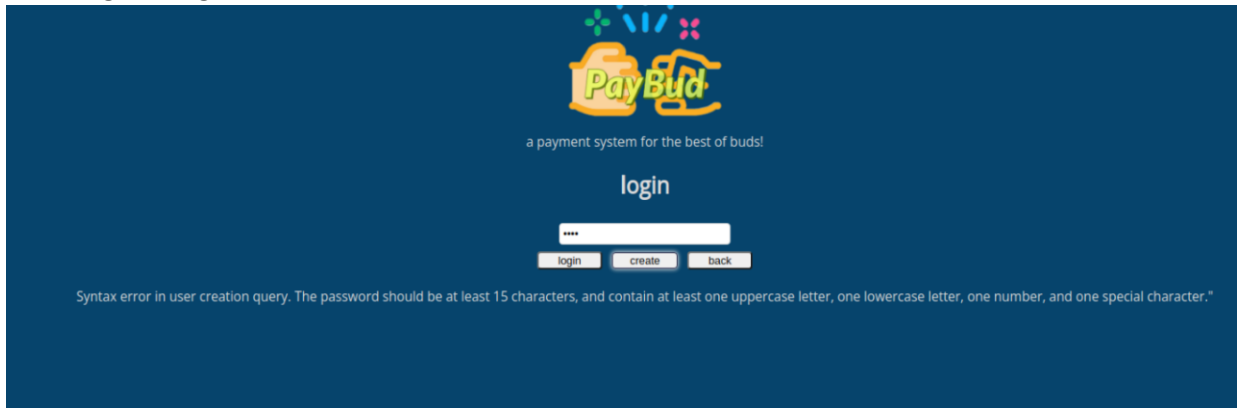
Implement your policy in PayBud. (which methods did you add/modify & how?)

```
public static boolean create(final String email, final String password) {  
    // Define your password policy here.  
    PasswordValidator validator = new PasswordValidator(Arrays.asList(  
        // At least 8 characters length  
        new LengthRule(minLength:8, maxLength:128),  
        // At least one upper-case character  
        new CharacterRule(EnglishCharacterData.UpperCase, num:1),  
        // At least one lower-case character  
        new CharacterRule(EnglishCharacterData.LowerCase, num:1),  
        // At least one digit character  
        new CharacterRule(EnglishCharacterData.Digit, num:1),  
        // At least one symbol (special character)  
        new CharacterRule(EnglishCharacterData.Special, num:1),  
        // No whitespace  
        new WhitespaceRule()  
    ));  
  
    RuleResult result = validator.validate(new PasswordData(password));  
  
    if (result.isValid()) {  
        final String iu = "INSERT INTO users VALUES ('" + email + "', '" + result + "');";  
        final String ia = "INSERT INTO accounts VALUES ('" + email + "', '0')";  
        try {  
            Connection c; Statement s;  
            c = DriverManager.getConnection(URL);  
            c.setAutoCommit(autoCommit:false); // enter transaction mode  
            s = c.createStatement();  
            s.executeUpdate(iu);  
            s = c.createStatement();  
            s.executeUpdate(ia);  
            c.commit(); // commit transaction  
            c.setAutoCommit(autoCommit:true); // exit transaction mode  
            c.close();  
            return true;  
        } catch (Exception e) {}  
        return false; // exception occurred; malformed SQL query?  
    } else {  
        // If password does not meet the policy, handle it here.  
        // You can fetch the failed rules and return appropriate messages.  
        return false;  
    }  
}
```

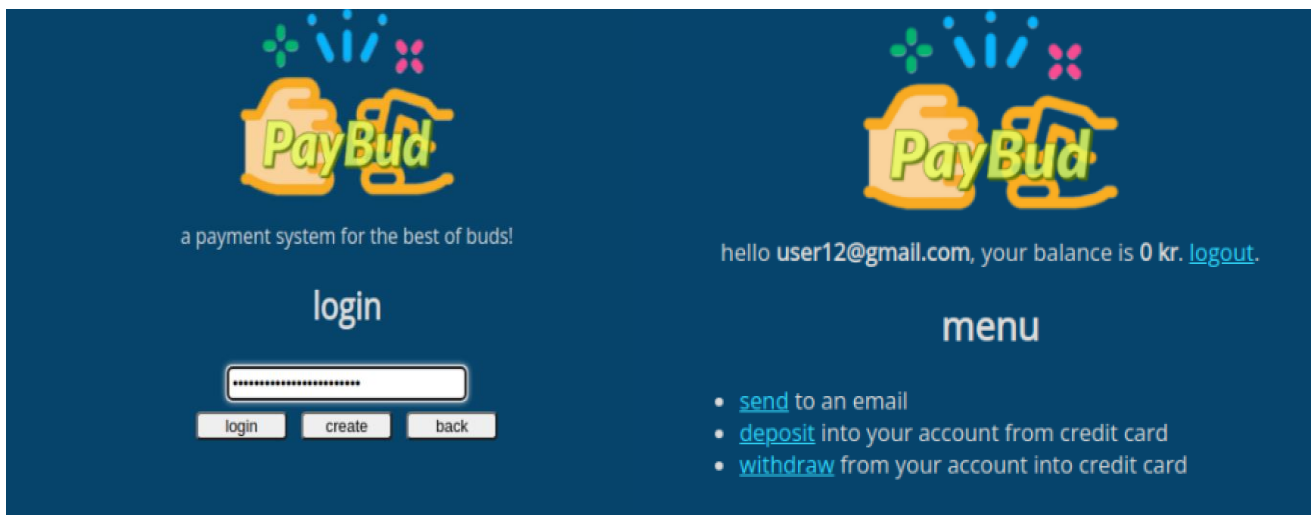
I have decided to accept passwords only of 15 lengths that need to have at least one uppercase letter, one lowercase and one special character.

APPLIED INFORMATION TECHNOLOGY – SUMMER COURSE

Now, when I try to create a new user with a password that does not meet the criteria, I get the following message:



But, once I implement a password according to the password policy, I am able to login!



Note: As I am validating only in the user creation method, users with passwords that do not meet the criteria, they can still log in

Part 4: What can you do to prevent Offline Dictionary / Brute-Force Attacks when the passwords are weak by nature (PIN numbers)?

Employ additional encryption to salted hashes with a secret key not stored in the database, known as "peppering", can enhance the security of weak stored passwords like PINs

What can you do (besides 2FA) to prevent Online Dictionary / Brute-Force Attacks?

I could implement account lockouts or delays after a certain number of failed attempts