

TLS decryption in Wireshark

When I spoke with some people I found out that most of them had some hard time with TLS decryption in world's foremost and widely-used network protocol analyzer "Wireshark".

TLS decryption could be very useful when we are analyzing some potential malicious web traffic or simply troubleshooting our own web server.

One of the method, I will be mentioning, works for every case covering situation like network traffic decryption of our web browser client "Google Chrome, Firefox, Python...". The second method covers situations where we have RSA Private key of web server and captured network traffic as ".pcap, .pcapng..." or we are performing live network capturing. The second method, using RSA private key is also applicable for TLS decryption of different protocol than HTTP.

Both of these methods have their advantages and disadvantages.

This little post is about TLS decryption possibilities in Wireshark and I hope it will serve someone as a simple guide.

Let's start 😊

As mentioned before, Wireshark supports TLS decryption when appropriate secrets are provided. The two available methods are:

- Key log file using per-session secrets (Using the (Pre)-Master-Secret)
- Decryption using RSA private key

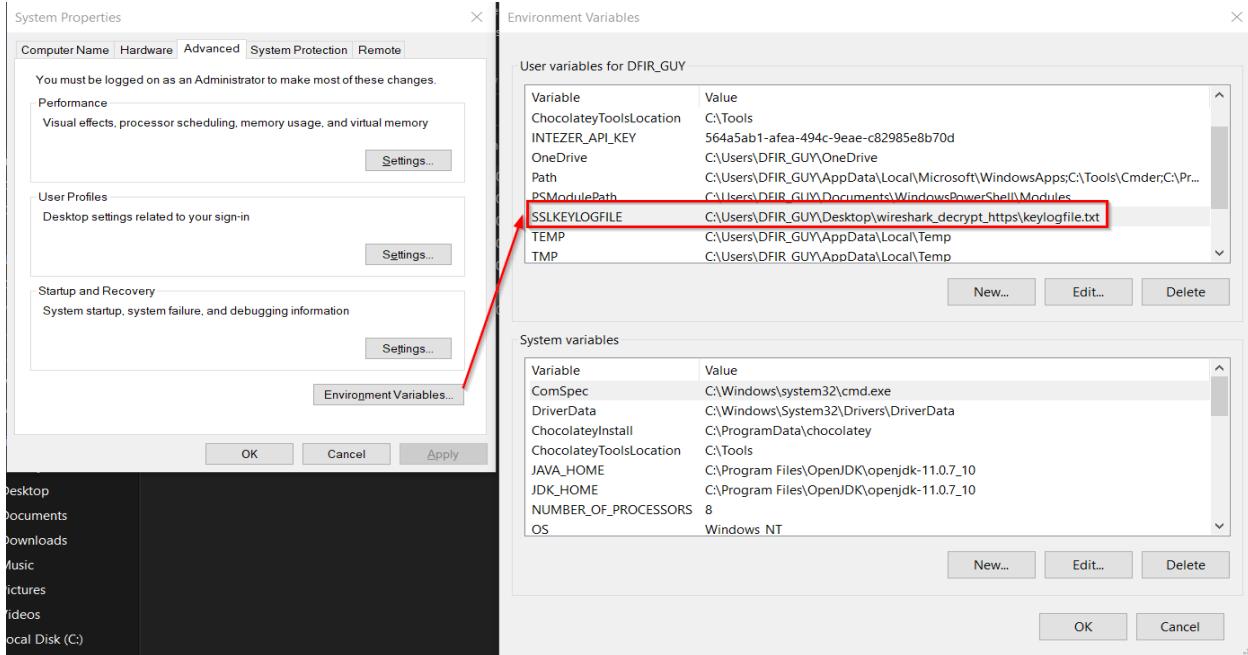
Using the (Pre)-Master-Secret

This method used for TLS decryption is using pre-master key logging. It is very simple and quick to set up but works only with some web browser clients (example: Google Chrome, Firefox, Python).

It works thanks to setting up the environment variable "SSLKEYLOGFILE". The value of this environment variable is path, pointing to some writable location where our key logging file will be created. Mentioned web browser clients check this environment variable and if it is set, they are able to log there per-session secrets which are immediately used in Wireshark.

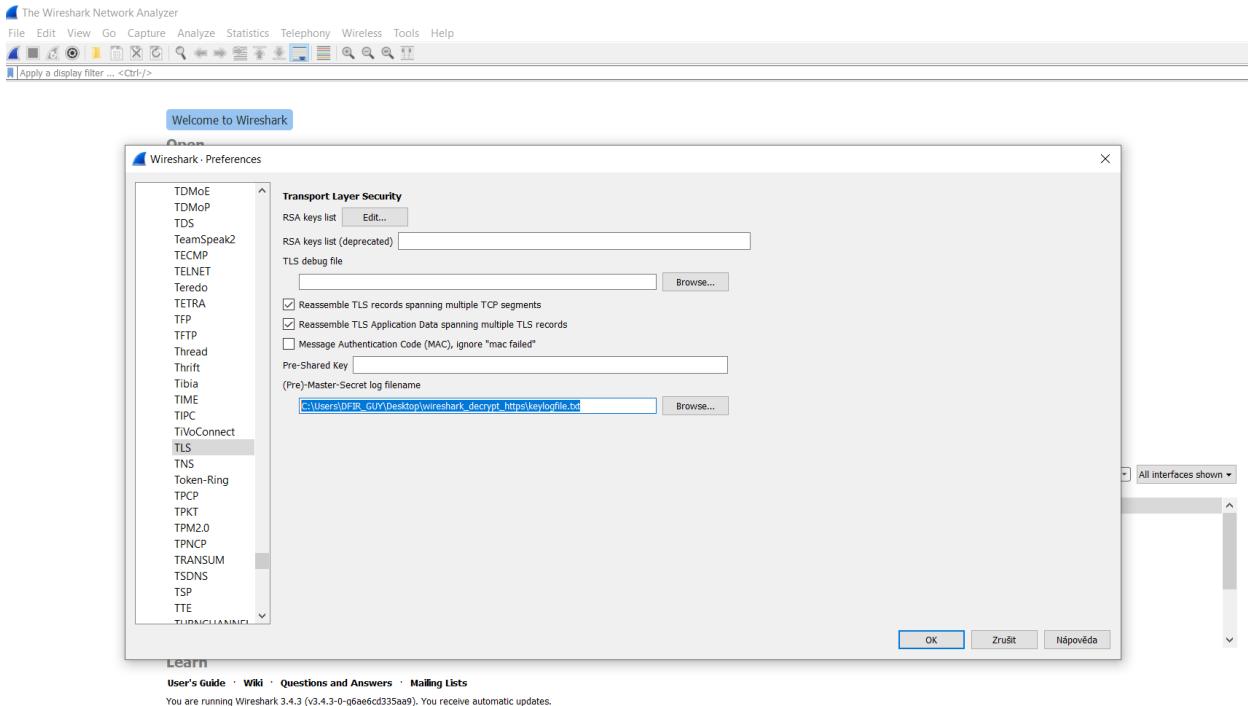
Advantage of this method is that it works in every case (SSL3. TLS1.0 -1.3), also when TLS cipher suite selected by the server is using (EC)DHE for key exchange algorithm.

We start with setting environment variable “SSLKEYLOGFILE”. As we can see in the picture below: (“SSLKEYLOGFILE = C:\Users\DFIR_GUY\Desktop\wireshark_decrypt_https\keylogfile.txt”)



After that, we have to set this path to Key log file containing our per-session secrets in Wireshark.

Start Wireshark and go to -> Edit -> Preferences -> Protocols -> TLS -> Pre Master-Secret log filename. Set the appropriate path as we set before in environment variable.

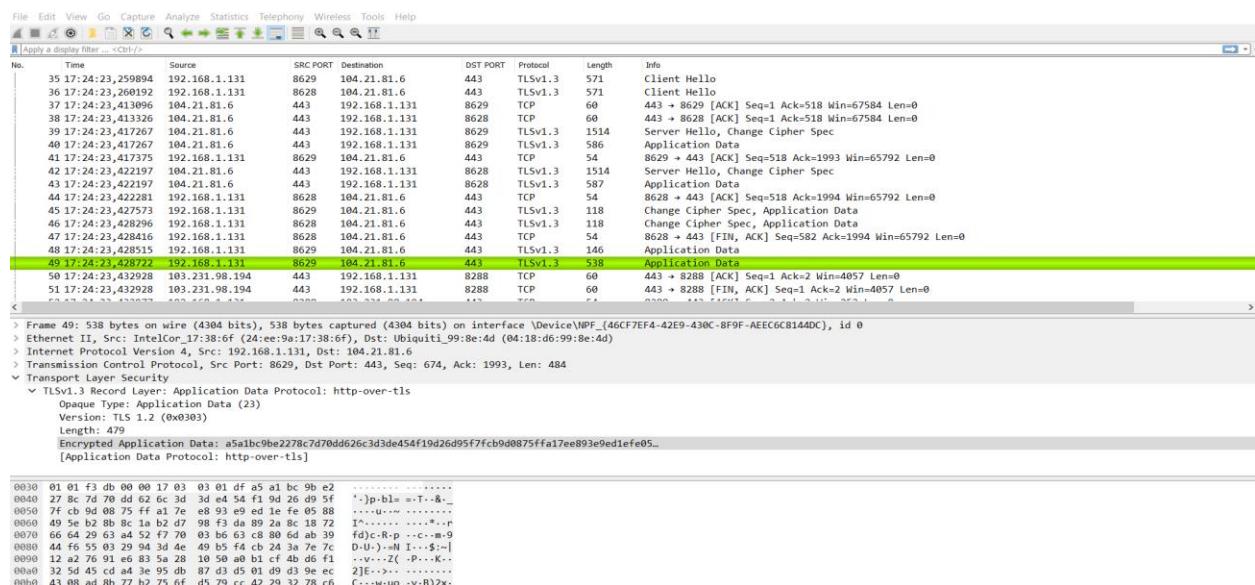


Now we are ready. Below is a network traffic of Google Chrome before and after setting the Key log file containing our per-session secrets:

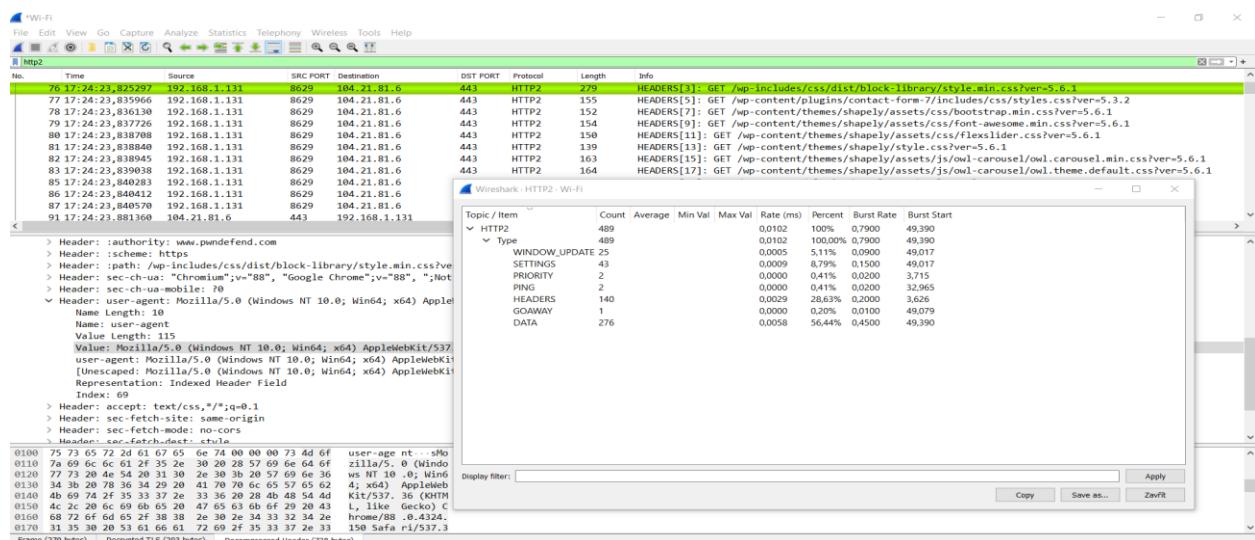
Before (No HTTP traffic available):



49. packet after TLS Handshake – Only TLS encrypted application data.



After:



49. packet after TLS Handshake – TLS decrypted application data HTTP2.

No.	Time	Source	Src Port	Destination	Dst Port	Protocol	Length	Info
38	17:24:23,413326	104.21.81.6	443	192.168.1.131	8628	TCP	60	443 → 8628 [ACK] Seq=1 Ack=518 Win=67584 Len=0
39	17:24:23,417267	104.21.81.6	443	192.168.1.131	8629	TLSv1.3	1514	Server Hello, Change Cipher Spec
40	17:24:23,417267	104.21.81.6	443	192.168.1.131	8629	TLSv1.3	586	Encrypted Extensions, Compressed Certificate, Certificate Verify, Finished
41	17:24:23,417375	192.168.1.131	8629	104.21.81.6	443	TCP	54	8629 → 443 [ACK] Seq=518 Ack=1993 Win=65792 Len=0
42	17:24:23,422197	104.21.81.6	443	192.168.1.131	8628	TLSv1.3	1514	Server Hello, Change Cipher Spec
43	17:24:23,422197	104.21.81.6	443	192.168.1.131	8628	TLSv1.3	587	Encrypted Extensions, Compressed Certificate, Certificate Verify, Finished
44	17:24:23,422281	192.168.1.131	8628	104.21.81.6	443	TCP	54	8628 → 443 [ACK] Seq=518 Ack=1994 Win=65792 Len=0
45	17:24:23,427573	192.168.1.131	8629	104.21.81.6	443	TLSv1.3	118	Change Cipher Spec, Finished
46	17:24:23,428296	192.168.1.131	8628	104.21.81.6	443	TLSv1.3	118	Change Cipher Spec, Finished
47	17:24:23,428416	192.168.1.131	8628	104.21.81.6	443	TCP	54	8628 → 443 [FIN, ACK] Seq=582 Ack=1994 Win=65792 Len=0
48	17:24:23,428515	192.168.1.131	8629	104.21.81.6	443	HTTP2	146	Magic, SETTINGS[0], WINDOW_UPDATE[0]
49	17:24:23,428722	192.168.1.131	8629	104.21.81.6	443	HTTP2	538	HEADERS[1]; GET /2020/02/04/tech-tip-simple-python3-https-server/
50	17:24:23,432928	103.231.98.194	443	192.168.1.131	8288	TCP	60	443 → 8288 [ACK] Seq=1 Ack=2 Win=4057 Len=0
51	17:24:23,432928	103.231.98.194	443	192.168.1.131	8288	TCP	60	443 → 8288 [FIN, ACK] Seq=1 Ack=2 Win=4057 Len=0
52	17:24:23,432977	192.168.1.131	8288	103.231.98.194	443	TCP	54	8288 → 443 [ACK] Seq=2 Ack=2 Win=252 Len=0
53	17:24:23,433171	103.231.98.194	443	192.168.1.131	8290	TCP	60	443 → 8290 [ACK] Seq=1 Ack=2 Win=4049 Len=0
54	17:24:23,435981	103.231.98.194	443	192.168.1.131	8290	TCP	60	443 → 8290 [FIN, ACK] Seq=1 Ack=2 Win=4049 Len=0
0120	60 74 00 00 00 73 4d 6f 7a 69 6e 6c 61 2f 35 2e nt...Mo zilla/5.							
0130	30 20 28 57 69 6e 64 6f 77 73 20 4e 54 20 31 30 0 (Windo ws NT 10							
0140	2e 30 3b 20 57 69 6e 36 34 3b 20 78 36 34 29 20 .; Win6 4; x64)							
0150	41 70 70 6c 65 57 65 62 4b 69 74 2f 35 33 37 2e AppleWeb Kit/537.							
0160	33 36 20 28 48 54 4d 4c 2c 20 6c 69 6b 65 20 36 (KHTML L, like							
0170	47 65 63 6b 6f 29 43 68 72 6f 6d 65 2f 38 38 Gecko) C hrome/88							
0180	2e 30 3e 34 33 32 34 2e 31 35 30 53 61 66 61 ,0.4324. 150 Safa							
0190	72 69 2f 35 33 37 2e 33 36 00 00 00 06 61 63 63 ri/537.3 6....acc							

```
> Header: path: /2020/02/04/tech-tip-simple-python3-https-server/
> Header: sec-ch-ua: "Chromium";v="88", "Google Chrome";v="88", "Not A Brand";v="99"
> Header: sec-ch-ua-mobile: ?0
> Header: upgrade-insecure-requests: 1
> Header: user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.150 Safari/537.36
> Header: accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
> Header: sec-fetch-site: none
> Header: sec-fetch-mode: navigate
> Header: sec-fetch-user: ?1
> Header: sec-fetch-dest: document
> Header: accept-encoding: gzip, deflate, br
> Header: accept-language: en-US,en;q=0.9
```

Using RSA private key

This method works in case where we own or get the RSA private key. It is applicable also for TLS decryption of different protocol than HTTP.

One of the limitations of using this method is that only SSL3, TLS1.0 -1.2 must be used. There is no support for TLS version 1.3. Another limitation is that (EC)DHE key exchange algorithm specified in the cipher suites must not be used because of principle how (EC)DHE key exchange algorithms work.

According to Wireshark docs only in these cases RSA private key method could be used:

- The cipher suite selected by the server is not using (EC)DHE.
- The protocol version is SSLv3, (D)TLS 1.0-1.2. It does not work with TLS 1.3.
- The private key matches the server certificate. It does not work with the client certificate, nor the Certificate Authority (CA) certificate.
- The session has not been resumed. The handshake must include the ClientKeyExchange handshake message.

So if we own the web server RSA private key and can manage to use supported version of TLS + not using cipher suites implementing (EC)DHE key exchange algorithm, we can successfully decrypt the TLS traffic.

These restrictions we can configure on the client side (web browser client) for example: to use only TLS 1.2 and cipher suites not using (EC)DHE key exchange algorithm but after that it all depends on server side if it enables to perform TLS handshake using these specified parameters.

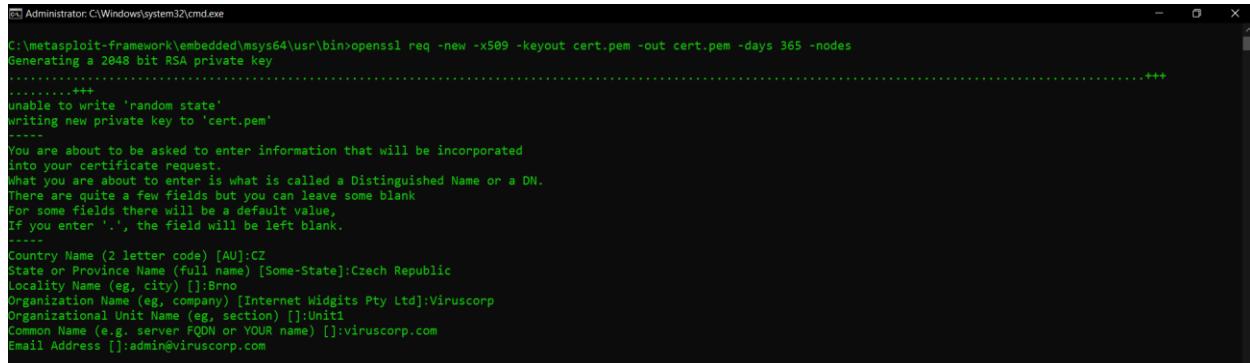
I will be focusing on how we can configure the server side to enable only TLS1.2 and cipher suites not using (EC)DHE key exchange algorithm. For this example, I created simple Python web server with our specified parameters which will be covered later.

First of all, we have to generate our RSA private key + Certificate for our web server. We will use “openssl”. We can download it here: [\[openssl\]](#).

Basically, Certificate contains RSA public key + another data for CERT verification. RSA private key is self-explanatory.

Generating Certificate + RSA private key in openssl: “openssl req -new -x509 -keyout cert.pem -out cert.pem -days 365 –nodes”

Fill the CERT info as you wish.



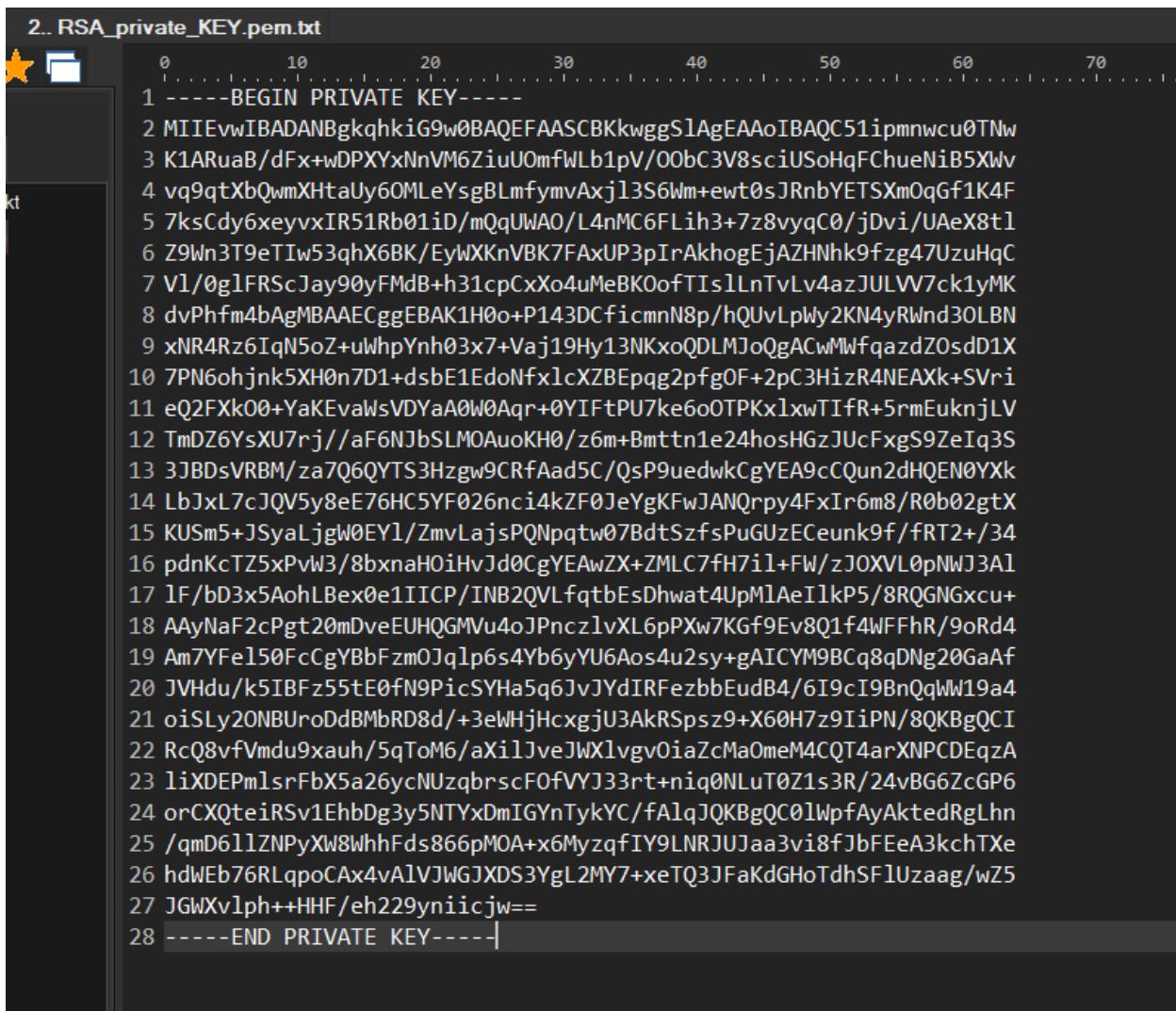
```
Administrator: C:\Windows\system32\cmd.exe
C:\metasploit-framework\embedded\msys64\usr\bin>openssl req -new -x509 -keyout cert.pem -out cert.pem -days 365 -nodes
Generating a 2048 bit RSA private key
.....+++
unable to write 'random state'
writing new private key to 'cert.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CZ
State or Province Name (full name) [Some-State]:Czech Republic
Locality Name (eg, city) []:Brno
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Viruscorp
Organizational Unit Name (eg, section) []:Unit
Common Name (e.g. server FQDN or YOUR name) []:viruscorp.com
Email Address []:admin@viruscorp.com
```

Our newly created cert.pem is containing both certificate and RSA private key as we can see in the picture below.

```
0          10          20          30          40          50          60          70          80          90
1 -----BEGIN PRIVATE KEY-----
2 MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBKkwwggS1AgEAAoIBAQCS1ipmnwcu0TNw
3 K1ARuaB/dFx+wDPXYxNnVM6ZiuU0mfWLb1pV/00bC3V8sciUSoHqFChueNiB5XWv
4 vq9qtXbQwmXhtaUy60MLEYsgBLmfymvAxjl3S6Wm+ewt0sJRnbYETSXm0qGf1K4F
5 7ksCdy6xeyvxIR51Rb01iD/mQqUWA0/L4nMC6FLih3+7zvyqC0/jDvi/UAeX8t1
6 Z9Wn3T9eTIw53qhX6BK/EyWXKnVBK7FAxUP3pIrAkhogEjAZHNhk9fzg47UzuHqC
7 Vl/0glFRScJay90yFMdB+h31cpCxXo4uMeBK0ofTIs1LnTvLv4azJULVV7ck1yMK
8 dvPhfm4bAgMBAECggEBAK1H0o+P143DCfcimnN8p/hQuvLpWy2KN4yRWnd3OLBN
9 xNR4Rz6IqN5oZ+uLhpYnh03x7+Vaj19Hy13NKxoQDLMJoQgACwMWfqazdZ0sdD1X
10 7PN6ohjnk5XH0n7D1+dsbE1EdoNxlcXZBEpqg2pfgOF+2pC3HizR4NEAXk+SVri
11 eQ2FXk00+YaKEvaWsVDYsA0W0Aqr+0YIFtPU7ke6o0TPKx1xwTIfr+5rmEuknjLV
12 TmDZ6YsXU7rj//aF6NJbSLMOAuoKH0/z6m+Bmttn1e24hosHGzJUcFxgS9ZeIq3S
13 3JBDsVRBM/za7Q6QYTS3Hzgw9CRFaad5C/QsP9uedwkCgYEAE9cCQun2dHQEN0YXk
14 LbJxL7cJQV5y8eE76HC5YF026nci4kZF0JeYgKFwJANQrpy4FxIr6m8/R0b02gtX
15 KUSm5+JSyaLjgW0EY1/ZmvLajsPQNpqtw07BdtSzfsPuGUzECeunk9f/fRT2+/34
16 pdnKtTZ5xPvW3/8bxnaH0iHvJd0CgYEAwZX+ZMLC7fH7i1+Fw/zJOXVL0pNWJ3Al
17 1F/bD3x5AoLBex0e1IICP/INB2QVLfqtbEsDhwat4UpM1AeIlkP5/8RQGNGxcu+
18 AAyNaF2cPgt20mDveEUHQGMVu4oJPncz1vXL6pPXw7KGf9Ev8Q1f4WFfH/R/9oRd4
19 Am7YFe150FcCgYBbFzm0Jqlp6s4Yb6yUy6Aos4u2sy+gAIcYM9BCq8qDNg20GaAf
20 JVHdu/k5IBFz55tE0fn9PicSYHa5q6JvJYdIRFezbBudB4/6I9cI9BnQqW19a4
21 oiSlY20NBUr0DdBmBRD8d/+3eWhjHcxgjU3AkRSpsz9+X60H7z9i1PN/8QKBgQCI
22 RcQ8vfVmdu9xauh/5qToM6/aXilJveJWXlvgvOiaZcMaOmeM4CQT4arXNPcDEqzA
23 liXDEPmlsrFbX5a26ycNUzqbrscF0fVYJ33rt+niq0NLuT0Z1s3R/24vBG6ZcGP6
24 orCXQteiRSv1Ebhdg3y5NTYxDmIGYnTykYC/fAlqJQKBgQC01WpfAyAktedRgLhn
25 /qmD611ZNPyXW8WhhFds866pM0A+x6MyzqfIY9LNRLJUJaa3vi8fJbFEeA3kchTXe
26 hdWEb76RLqpoCAx4vAlVJWGJXDS3YgL2MY7+xeTQ3JFaKdGhoTdhSF1Uzaag/wZ5
27 JGWXvlph++HHF/eh229yniicjw==
28 -----END PRIVATE KEY-----
29 -----BEGIN CERTIFICATE-----
30 MIID/zCCAuegAwIBAgIJIAIswlWPT35q6MA0GCSqGSIb3DQEBCwUAMIGVMQswCQYD
31 VQQGEwJDWjEXMBUGA1UECAwOQ3p1Y2ggUmVwdJlsalWmxDTALBgNVBAcMBEJybml8x
32 EjaQBgNVBAoMCVZpcnVzY29ycDEOMAwGA1UECwxFVW5pdDExFjAUBgNVBAMMDXZp
33 cnVzY29ycC5jb20xIjAgBpkhkiG9w0BCQEW2FkbWluQHZpcnVzY29ycC5jb20w
34 HhcNMjEwMjE2MTcxMjUyWhcNMjIwMjE2MTcxMjUyWjCB1TELMAkGA1UEBhMCQ1ox
35 FzAVBgNVBAgMDkN6ZWNoIFJlcHVibGljMQ0wCwYDVQQHARCCm5vMRIwEAYDVQQK
36 DALWaXJ1c2NvcnAxDjAMBgNVBAsMBVVuaXQxMRYwFAYDVQQDDA12aXJ1c2NvcnAu
37 Y29tMSIwIAYJKoZIhvNAQkBFhNhZG1pbkB2aXJ1c2NvcnAuY29tMIIBIjANBkgq
38 hkiG9w0BAQEFAAOCAQ8AMIIICgKCAQEAdYqZp8HLtEzcCtQEbmfg3RcfsAz12MT
39 Z1T0mYrlDpn1i29aVfzjmwt1fLHI1EqB6hQobnjYgeV1r76varV20MJ1x7WlMu jj
40 C3mLIAS5n8prwMY5d0ulpvnsLdLCUZ22BE015jqhn9SuBe5LAncusXsr8SEedUlw9
```

We will extract the RSA private key from cert.pem which we will be using later in Wireshark. The cert.pem file with both RSA private key and certificate is used in our Python server implementation later on.

Extracted RSA private key:



```
2.. RSA_private_KEY.pem.txt
0      10     20     30     40     50     60     70
1 -----BEGIN PRIVATE KEY-----
2 MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBKkwggSlAgEAAoIBAQC51ipmnwcu0TNw
3 K1ARuaB/dFx+wDPXYxNnVM6ZiuU0mfLb1pV/00bC3V8sciUSoHqFChueNiB5XWv
4 vq9qtXbQwmXHtaUy60MLEYsgBLmfymvAxj13S6Wm+ewt0sJRnbYETSXm0qGf1K4F
5 7ksCdy6xeyvxIR51Rb01iD/mQqUWA0/L4nMC6FLih3+7z8vyqC0/jDvi/UAeXt1
6 Z9Wn3T9eTIw53qhX6BK/EylWXKnVBK7FAxUP3pIrAkhogEjAZHNhk9fzg47UzuHqC
7 V1/0g1FRScJay90yFMdB+h31cpCxXo4uMeBK0oFTIs1LnTvLv4azJULVV7ck1yMK
8 dvPhfm4bAgMBAECggEBAK1H0o+P143DCfcmnN8p/hQUvLpWy2KN4yRWnd30LBN
9 xNR4Rz6IqN5oZ+uWhpYnh03x7+Vaj19Hy13NKxoQDLMJoQgACwMWfqazdZ0sdD1X
10 7PN6ohjnk5XH0n7D1+dsbE1EdoNfxlcXZBEpqg2pfgOF+2pC3HizR4NEAXk+SVri
11 eQ2FXk00+YaKEvaWsVDYaa0W0Aqr+0YIFTPU7ke6o0TPKx1xwTIfR+5rmEuknjLV
12 TmDZ6YsXU7rj//aF6NJbSLMOAuOKh0/z6m+Bmttn1e24hosHGzJUcFwgS9ZeIq3S
13 3JBDsVRBM/za7Q6QYTS3Hzgw9CRfAad5C/QsP9uedwkCgYEA9cCQun2dHQEN0YXk
14 LbJxL7cJQV5y8eE76HC5YF026nci4kZF0JeYgKFwJANQrpY4FxIr6m8/R0b02gtX
15 KUSm5+JSyaLjgW0EYl/ZmvLajsPQNpqtw07BdtSzfsPuGUzECeunk9f/fRT2+/34
16 pdnKcTZ5xPvW3/8bxnaH0iHvJd0CgYEAwZX+ZMLC7fH7il+FW/zJOXL0pNWJ3Al
17 lF/bD3x5AoLBex0e1IICP/INB2QVLfqtbEsDhwat4UpM1AeI1kP5/8RQGNGxcu+
18 AAyNaF2cPgt20mDveEUHQGMVu4oJPnczlvXL6pPxw7KGf9Ev8Q1f4WFFhR/9oRd4
19 Am7YFe150FcCgYBbFzm0Jqlp6s4Yb6yYU6Aos4u2sy+gAICYM9BCq8qDNg20GaAf
20 JVHdu/k5IBFz55tE0fN9PicSYHa5q6JvJYdIRFezbBudB4/6I9cI9BnQqlWW19a4
21 oiSLy20NBUr0DdBmRD8d/+3eWHjHcxgjU3AkRSpsz9+X60H7z9IiPN/8QKBgQCI
22 RcQ8vfVmdu9xauh/5qToM6/aXilJveJWx1vgv0iaZcMaOmeM4CQT4arXNPcDEqzA
23 liXDEPmlsrFbX5a26ycNUzqbrscF0fVYJ33rt+niq0NLuT0Z1s3R/24vBG6ZcGP6
24 orCXQteiRSv1EhbDg3y5NTYxDmIGYnTykYC/fAlqJQKBgQC01WpfAyAktedRgLhn
25 /qmD611ZNPyXW8WhhFds866pMOA+x6MyzqfIY9LNRJUJaa3vi8fJbFEeA3kchTXe
26 hdWEb76RLqpoCAx4vA1VJWGXJDS3YgL2MY7+xeTQ3JFaKdGHoTdhSF1Uzaag/wZ5
27 JGWXvlph++HHF/eh229ynicjw==
28 -----END PRIVATE KEY-----
```

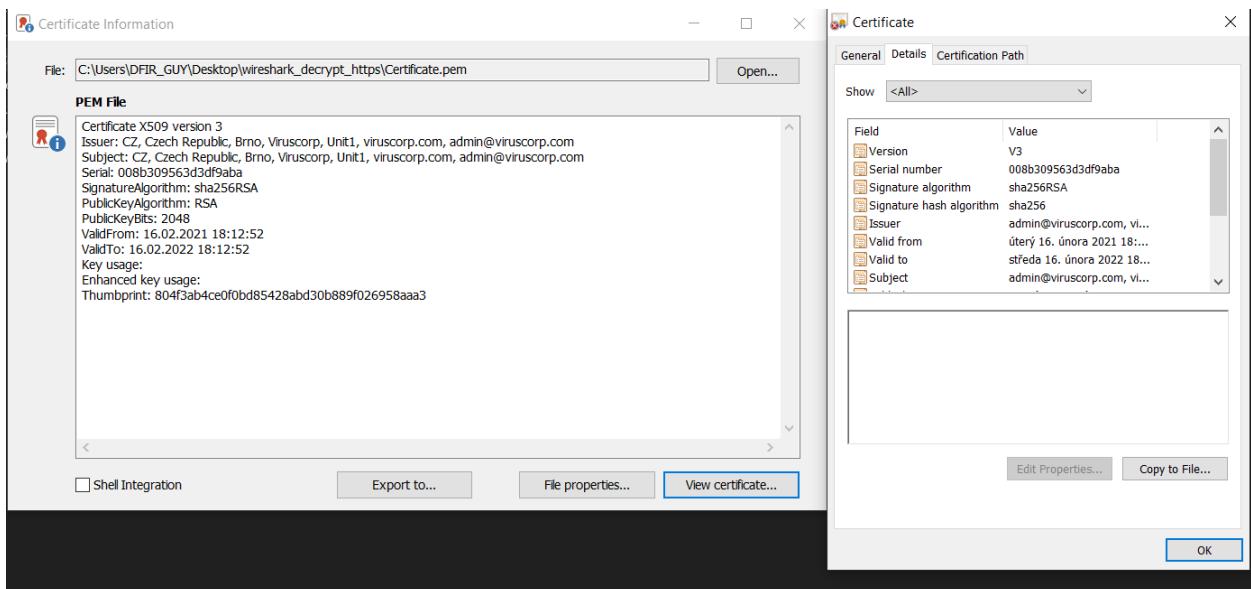
We can also extract the certificate from cert.pem, save it as certificate.pem and check containing info we filled during its creation before. We can check these information in tool like [\[certinfo\]](#).

```

0          10         20         30         40         50         60         70         80
-----BEGIN CERTIFICATE-----
2 MIID/zCCAuegAwIBAgIJIAIswlWPT35q6MA0GCSqGSIb3DQEBCwUAMIGVMQswCQYD
3 VQQGEwJDWjEXMBUGA1UECAw0Q3p1Y2ggUmVwdWJsaWmxDTALBgNVBAcMBEJybm8x
4 EjAQBgNVBAoMCVZpcnVzY29ycDEOMAwGA1UECwwFVW5pdDExFjAUBgNVBAMMDXZp
5 cnVzY29ycC5jb20xIjAgBgkqhkiG9w0BCQEWEIFkbWluQHZpcnVzY29ycC5jb20w
6 HhcNMjEwMjE2MTcxMjUyWhcNMjIwMjE2MTcxMjUyWjCB1TELMAkGA1UEBhMCQ1ox
7 FzAVBgNVBAgMDkN6ZWNoIFJlcHVibG1jMQ0wCwYDVQQHDARCc5vMRIwEAYDVQQK
8 DALWaXJ1c2NvcnAxDjAMBgNVBAwMBVVuaXQxMRYwFAYDVQQDDA12aXJ1c2NvcnAu
9 Y29tMSIwIAYJKoZIhvNAQkBFhNhZG1pbkB2aXJ1c2NvcnAuY29tMIIBIjANBgkq
10 hkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAdYqZp8HLtEzcCtQEbmfgf3RcfsAz12MT
11 Z1T0mYr1Dpn1i29aVfzjmwt1fLHI1EqB6hQobnjYgeV1r76varV20Mjlx7W1Mujj
12 C3mLIAS5n8prwMY5d0ulpvnsLdLCUZ22BE015jqhn9SuBe5LAncusXsr8SEedUW9
13 NYg/5kK1FgDvy+JzAuhS4od/u8/L8qgtP4w74v1AH1/LZwfVp90/XkyM0d6oV+gS
14 vxM1lyp1QSuxQMVD96SKwJIaIBIwGRzYZPX84001M7h6g1Zf9IJRUUnCwsvdMhTH
15 Qfod9XKQsV60LjhGsjqH0yLJS507y7+GsyVC1Ve3JNcjCnbz4X5uGwIDAQABo1Aw
16 TjAdBgNVHQ4EFgQUQLEJdDk5RFaoX2ct7c/pj4np5vAwHwYDVR0jBBgwFoAUQLEJ
17 dDk5RFaoX2ct7c/pj4np5vAwDAYDVR0TBauAwEB/zANBgkqhkiG9w0BAQsFAAOc
18 AQEAtF1PjKoz3/J910hxUPtrW5FoHOYCmftDX9msTx8YIqCYkitbmNIv0Qujb5Z3
19 uQ10yHR6phi4NuTnXOCsKQc8po4CGaVK9tVC9NqnQ01S40Yb2JyvTd8PGty+E2II
20 7MnnihbUwtv8GqlWUPJz0oBfjdtFSWT4grwVol5NgDEZUZdowNxtCyaZ3RaviMx+c
21 Eq7S3jFY02/MiId4G3A7T1c5EtB4h4Ko/k0ovUY7M+w6bF6CVetHRrERuz+10LM7
22 u21TTGj1USk82q7KLp3LPYjekvFBHN6LNXMgB/R/CNFCEJpxFrQHjDwfLK2uq+o
23 FbYuONDv25jHPFcM0pD/QXAQlg==
24 -----END CERTIFICATE-----

```

Using Certinfo:

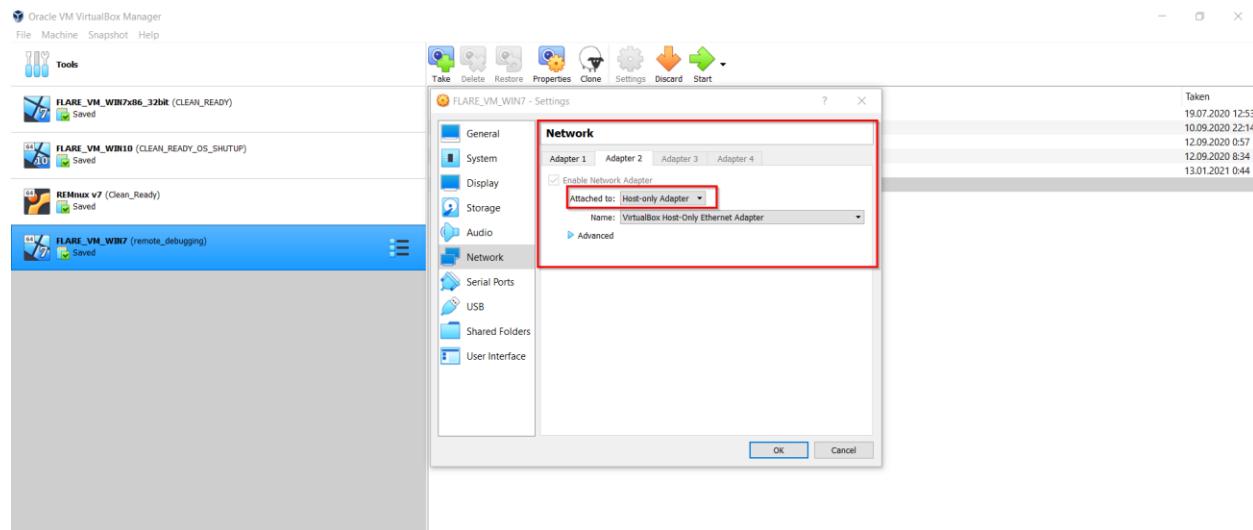


In real scenario/case, we will already have some web server private key and captured network traffic data but here, we will be implementing our own python web server with our previously created certificate + RSA private key.

Let's start to run some VM where we have installed python3+. For virtualization platform I will be using VirtualBox and as Guest OS – Windows 7.

Let's configure our VM network setting in VirtualBox:

Choose VM and go to Settings → Network and enable Network adapter (Host-only Adapter) – this settings will create Network interface only between our Host and Guest VM machine.



Start our VM and check the IP address on our newly created network interface (Host-only Adapter). In my case the IP “192.168.56.2” is used.

```
C:\Users\INFERNO\Desktop>ipconfig
```

```
Windows IP Configuration
```

```
Ethernet adapter Local Area Connection 2:
```

```
Connection-specific DNS Suffix . . . . . fe80::fdeb:536f:c94e:e7b2%15
Link-local IPv6 Address . . . . . 192.168.56.2
IPv4 Address . . . . . 192.168.56.2
Subnet Mask . . . . . 255.255.255.0
Default Gateway . . . . .
```

```
Ethernet adapter Npcap Loopback Adapter:
```

```
Connection-specific DNS Suffix . . . . . fe80::eddf:3ecf:6e61:4d25%13
Link-local IPv6 Address . . . . . 169.254.77.37
Autoconfiguration IPv4 Address . . . . . 169.254.77.37
Subnet Mask . . . . . 255.255.0.0
Default Gateway . . . . .
```

```
Ethernet adapter Local Area Connection:
```

```
Connection-specific DNS Suffix . . . . . fe80::f149:4696:15bd:6174%11
Link-local IPv6 Address . . . . . 10.0.2.15
IPv4 Address . . . . . 10.0.2.15
Subnet Mask . . . . . 255.255.255.0
Default Gateway . . . . . 10.0.2.2
```

```
Tunnel adapter isatap.{7F2E8261-B763-4A33-8154-B7ECDD000C73}:
```

```
Media State . . . . . Media disconnected
Connection-specific DNS Suffix . . . . .
```

```
Tunnel adapter isatap.{9489E21D-B7A4-4904-8AFC-7DE9310DB243}:
```

```
Media State . . . . . Media disconnected
Connection-specific DNS Suffix . . . . .
```

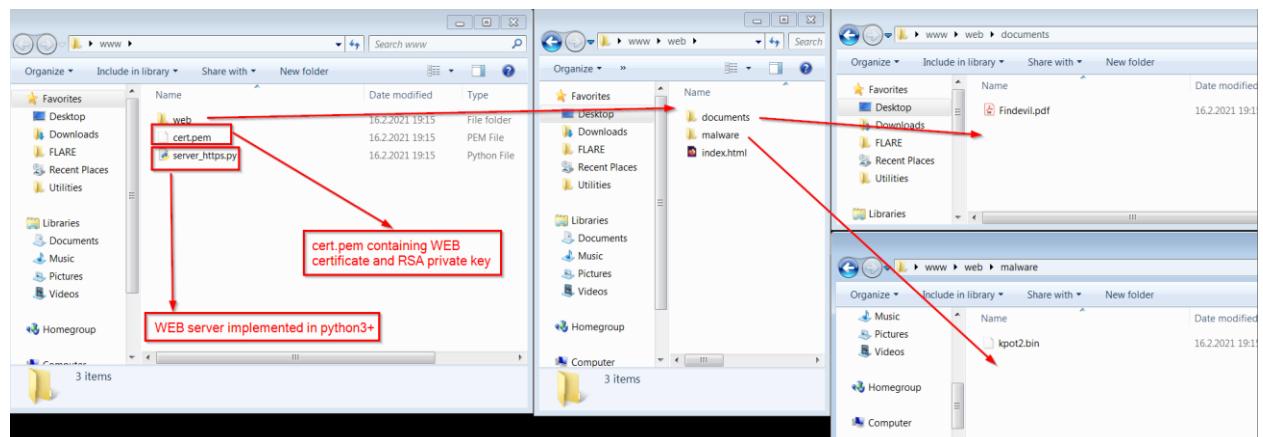
```
Tunnel adapter isatap.{CA791937-459D-4EBD-987C-AACBFFEE4ABE5}:
```

```
Media State . . . . . Media disconnected
Connection-specific DNS Suffix . . . . .
```

```
C:\Users\INFERNO\Desktop>
```

Now we will copy our previously created cert.pem containing both certificate and RSA private key to the VM.

We also implement the web server in python, some optional folders/files and some example html web content (Download here: [\[www content\]](#))



The web server implementation in Python3+ is in the picture below and you can download this example from: [\[HTTPS WEB SERVER.py\]](#)

```

1 import http.server, ssl, os
2
3 web_dir = os.path.join(os.path.dirname(__file__), 'web') ← Setting our ROOT web server folder to
4 os.chdir(web_dir) ← folder "web". Python web server script is in
5
6 server_address = ('192.168.56.2', 443) ← parent folder "www"
7 httpd = http.server.HTTPServer(server_address, http.server.SimpleHTTPRequestHandler)
8 httpd.socket = ssl.wrap_socket(
9     httpd.socket,
10    server_side=True,
11    certfile='C:\\Users\\INFERNO\\Desktop\\www\\cert.pem',
12    ssl_version=ssl.PROTOCOL_TLSv1_2, ← Path to our previously created cert.pem
13    ciphers='RSA+AESGCM:RSA+AES:RSA+HIGH') ← containing certificate and RSA private key
14
15 httpd.serve_forever() ← Only TLS version 1.2 will be used
16

```

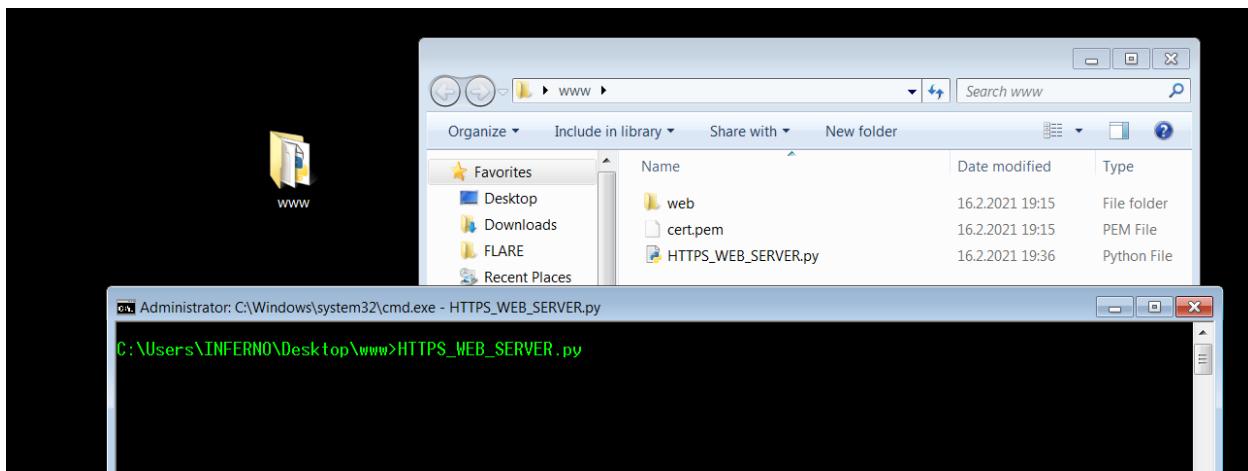
Our web server IP address is "192.168.56.2" and port "443"

Here we set the restriction that only cipher suites NOT implementing (EC)DHE key exchange algorithm will be used.

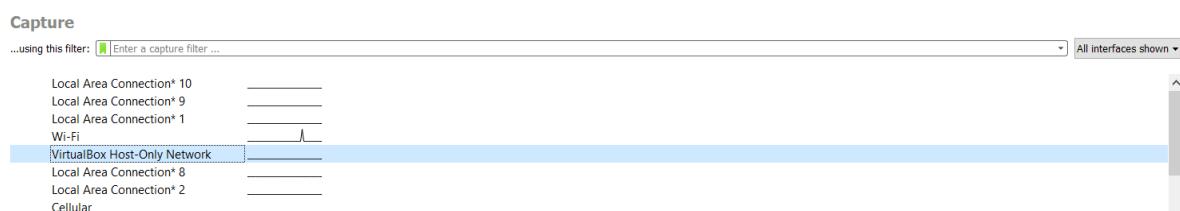
As we can see, we configured the Python web server to meet all conditions for TLS decryption via RSA private key method in Wireshark. (TLS version 1.2, only cipher suites NOT implementing (EC)DHE key exchange algorithm, cert.pem containing our RSA private key)

The used IP address is that we previously obtained from interface (Host-only Adapter).

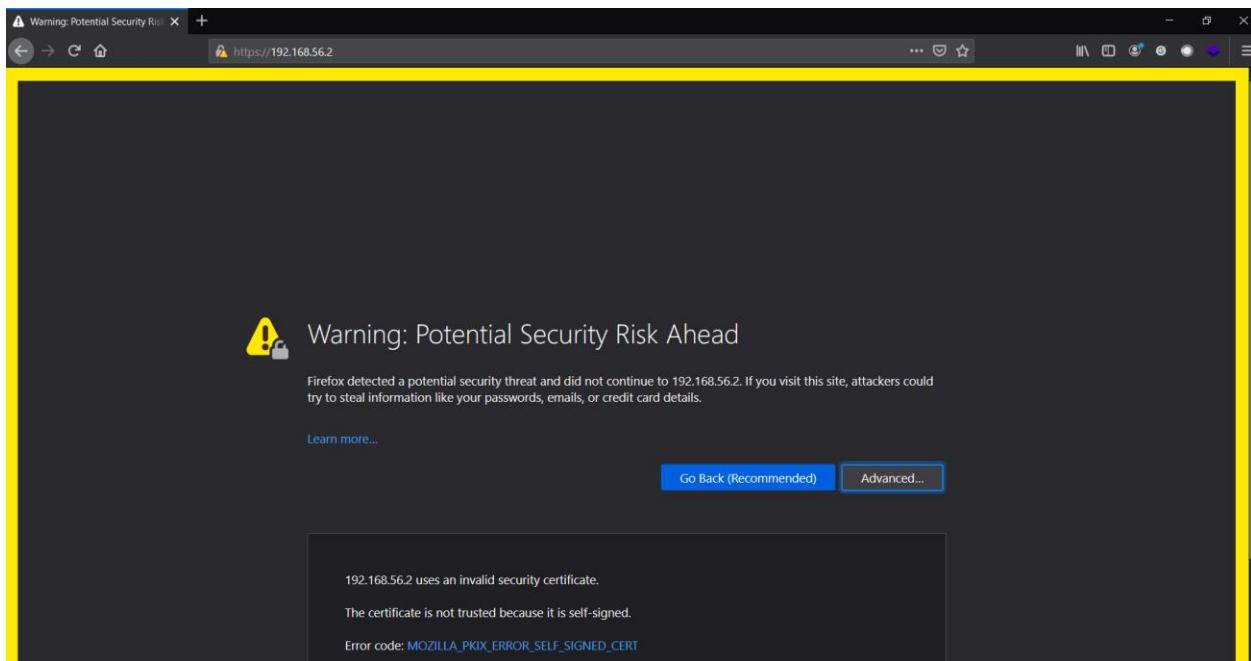
Now we can start our Python web server in VM.



Start Wireshark in Host and choose interface “VirtualBox Host-Only Network”.



Start web browser in host and navigate to hxxps[://]192.168.56.2



As we can see our certificate is self-signed and it is not trusted (cannot be verified). Let's check the certificate. We got our CERT information as we expected.

The screenshot shows the Firefox certificate details page for "viruscorp.com". The URL in the address bar is "about:certificate?cert=MIID%2fzCCAuegAwIBAgIJAlswlWPT35q6MA0GCSqGSIb3DQEBCwUAMIGVMQswCQYDVQQGEwJDWjEXMBUG/". The page displays the following certificate information:

Subject Name	
Country	CZ
State/Province	Czech Republic
Locality	Brno
Organization	Viruscorp
Organizational Unit	Unit1
Common Name	viruscorp.com
Email Address	admin@viruscorp.com

Issuer Name	
Country	CZ
State/Province	Czech Republic
Locality	Brno
Organization	Viruscorp
Organizational Unit	Unit1
Common Name	viruscorp.com
Email Address	admin@viruscorp.com

Validity	
Not Before	2/16/2021, 6:12:52 PM (Central European Standard Time)
Not After	2/16/2022, 6:12:52 PM (Central European Standard Time)

Public Key Info	
-----------------	--

Let's proceed (Accept the RISK and Continue).

We reached our previously created index.html example served via our Python web server implementation. You can download this index.html example here: [\[index.html\]](#)



This is an example. You can not see this content in **Wireshark** if you are not able to **Decrypt** TLS encrypted network traffic.

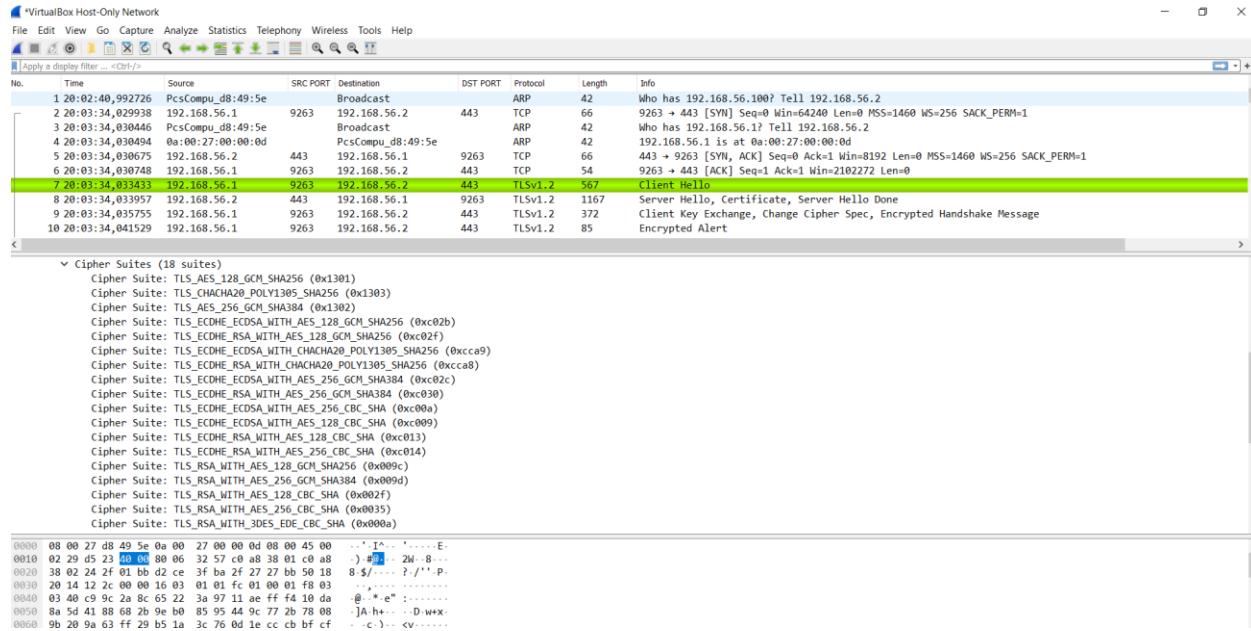
Downloads

[Kpot2 Malware](#)

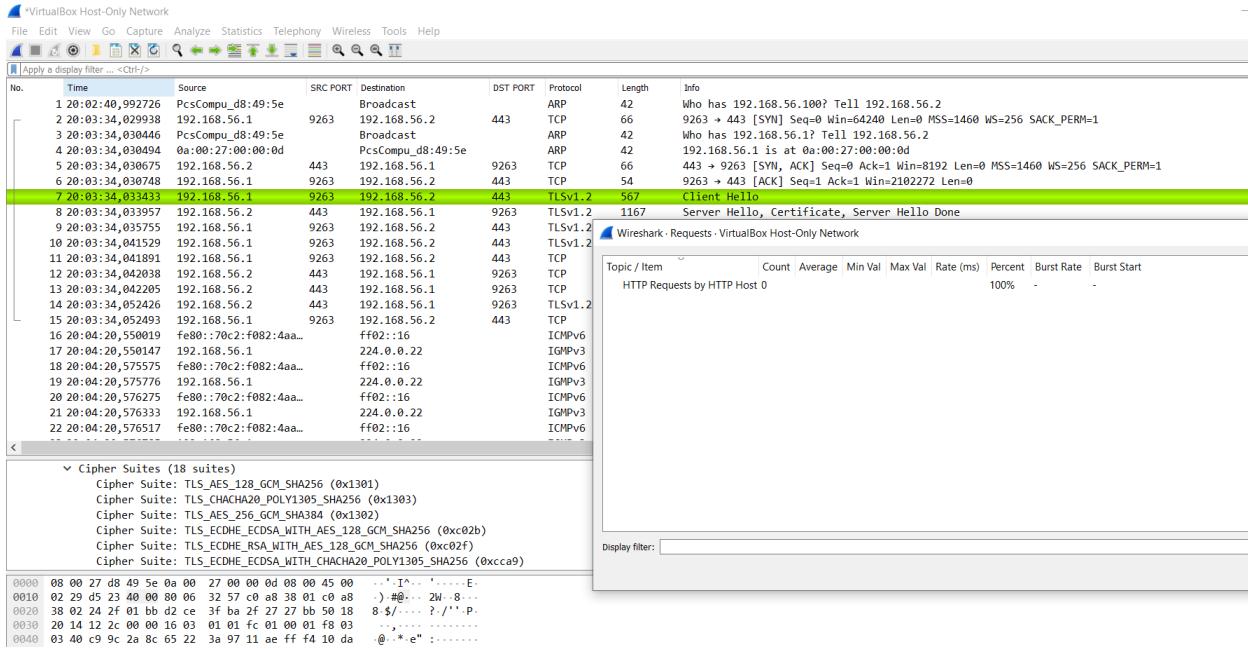
[Findevil.pdf](#)

We will download included files, exit the web browser and check the Wireshark.

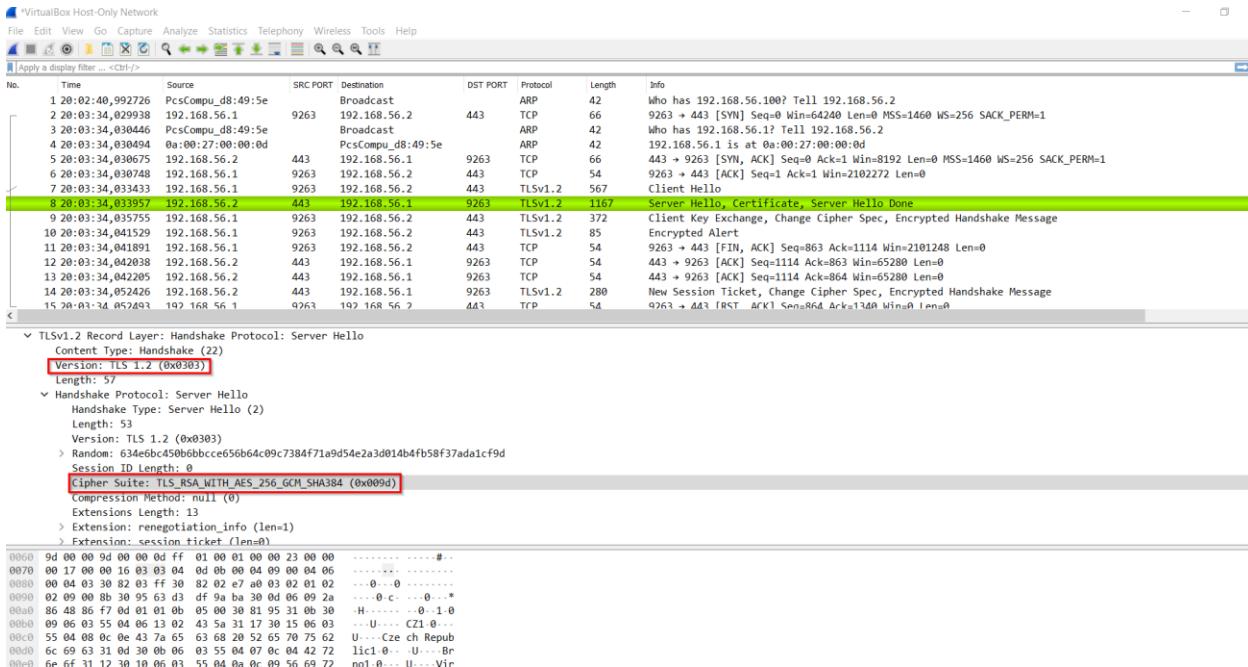
We can see in Wireshark that during TLS handshake, our web browser is offering 18 cipher suites.



There is no HTTP traffic available and all is encrypted via TLS.



We can check that our Python web server is choosing TLS version 1.2 and cipher suite "TLS_RSA_WITH_AES_256_GCM_SHA384" during TLS negotiation (Handshake) as we previously configured.



We can also check the web server certificate information.

Time	Source	Src Port	Destination	Dst Port	Protocol	Length	Info
1 20:02:40, 992726	PcsCompu_d8:49:5e		Broadcast		ARP	42	Who has 192.168.56.100? Tell 192.168.56.2
2 20:03:34, 020938	192.168.56.1	9263	192.168.56.2	443	TCP	66	9263 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
3 20:03:34, 030446	PcsCompu_d8:49:5e		Broadcast		ARP	42	Who has 192.168.56.1? Tell 192.168.56.2
4 20:03:34, 030494	0a:00:27:00:00:0d		PcsCompu_d8:49:5e		ARP	42	192.168.56.1 is at 0a:00:27:00:00:0d
5 20:03:34, 030675	192.168.56.2	443	192.168.56.1	9263	TCP	66	443 → 9263 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
6 20:03:34, 030748	192.168.56.1	9263	192.168.56.2	443	TCP	54	9263 → 443 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
7 20:03:34, 033433	192.168.56.1	9263	192.168.56.2	443	TLSv1.2	567	Client Hello
8 20:03:34, 033597	192.168.56.2	443	192.168.56.1	9263	TLSv1.2	1167	Server Hello, Certificate, Server Hello Done
9 20:03:34, 035755	192.168.56.1	9263	192.168.56.2	443	TLSv1.2	372	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
10 20:03:34, 041529	192.168.56.1	9263	192.168.56.2	443	TLSv1.2	85	Encrypted Alert
11 20:03:34, 041891	192.168.56.1	9263	192.168.56.2	443	TCP	54	9263 → 443 [FIN, ACK] Seq=863 Ack=1114 Win=2101248 Len=0
12 20:03:34, 042038	192.168.56.2	443	192.168.56.1	9263	TCP	54	443 → 9263 [ACK] Seq=1114 Ack=863 Win=65280 Len=0
13 20:03:34, 042205	192.168.56.2	443	192.168.56.1	9263	TCP	54	443 → 9263 [ACK] Seq=1114 Ack=864 Win=65280 Len=0
14 20:03:34, 052426	192.168.56.2	443	192.168.56.1	9263	TLSv1.2	280	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
15 20:03:34, 052493	192.168.56.1	9263	192.168.56.2	443	TCP	54	9263 → 443 [RST, ACK] Seq=864 Ack=1308 Win=0 Len=0

> signature (sha256WithRSAEncryption)							
> issuer: rdnSequence (0)							
> validity							
> subject: rdnSequence (0)							
> rdnSequence: 7 items (pkcs-9-at-emailAddress=admin@viruscorp.com,id-at-commonName=viruscorp.com,id-at-organizationalUnitName=Unit1,id-at-organizationName=Viruscorp,id-at-localityName=Brno...)							
> RDNSequence item: 1 item (id-at-countryName=CZ)							
> RDNSequence item: 1 item (id-at-stateOrProvinceName=Czech Republic)							
> RDNSequence item: 1 item (id-at-localityName=Brno)							
> RDNSequence item: 1 item (id-at-organizationName=Viruscorp)							
> RDNSequence item: 1 item (id-at-organizationalUnitName=Unit1)							
> RDNSequence item: 1 item (id-at-commonName=viruscorp.com)							
> RDNSequence item: 1 item (pkcs-9-at-emailAddress=admin@viruscorp.com)							
> subjectPublicKeyInfo							
> extensions: 3 items							
> algorithmIdentifier (sha256WithRSAEncryption)							
50 32 5a 30 81 95 31 0b 30 09 06 03 55 04 06 13 05						220 11-0 ..U...	
51 45 3a 31 17 03 01 55 04 06 04 43 73 03						74,0 ..U...	
52 63 69 20 52 65 70 75 62 6c 69 63 31 04 30 08 06						ch Repub lic1.0...	
53 03 55 04 07 0c 04 42 72 6e 6f 31 12 30 10 06 03						..U...By no1.0...	
54 05 55 04 0a 0c 09 56 69 72 75 73 63 6f 72 70 31 0e						..Vir uscorp1...	
55 30 0c 06 03 55 04 06 0c 05 55 6e 69 74 31 31 16						..U... Unit11...	
56 30 14 06 03 55 04 03 0c 0d 76 69 72 75 73 63 6f						..U... virusco...	
57 30 20 06 03 55 04 02 0d 76 69 72 75 73 63 6f						..H...	
58 77 70 2e 63 6f 6d 31 22 30 20 06 09 2a 86 48 86						cp.com1 ..*..H...	

Now is the time to decrypt TLS and obtain HTTP traffic.

According to Wireshark documentation there are 2 methods how we can perform TLS decryption via RSA private key in Wireshark:

Starting with Wireshark 3.0, a new RSA Keys dialog can be found at *Edit -> Preferences -> RSA Keys*. In this dialog, use the *Add new keyfile...* button to select a file. You will be prompted for a password if necessary. The *Add new token...* button can be used to add keys from a HSM which might require using *Add new provider...* to select select a DLL/.so file, and additional vendor-specific configuration.

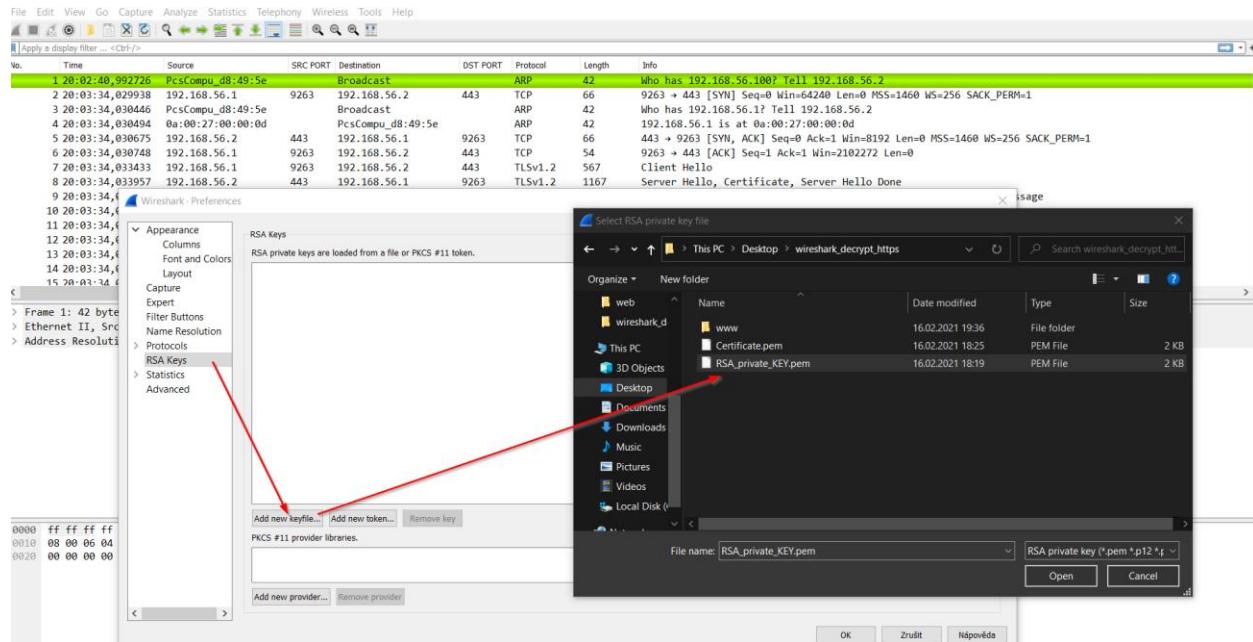
The RSA key file can either be a *PEM* format private key or a PKCS#12 keystore (typically a file with a .pfx or .p12 extension). The PKCS#12 key is a binary file, but the PEM format is a text file which looks like this:

- -----BEGIN PRIVATE KEY-----
- MIIEvglBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDReQzlKVeAK8b5
- TRcRBhSi9lYwHX8Nqc8K4HeDRvN7HiBQQP3bhUkVekdoXpRLYVuc7A8h1BLr93Qw
- ...
- KOi8FZl+jhG+p8vtpK5ZAlyp
- -----END PRIVATE KEY-----

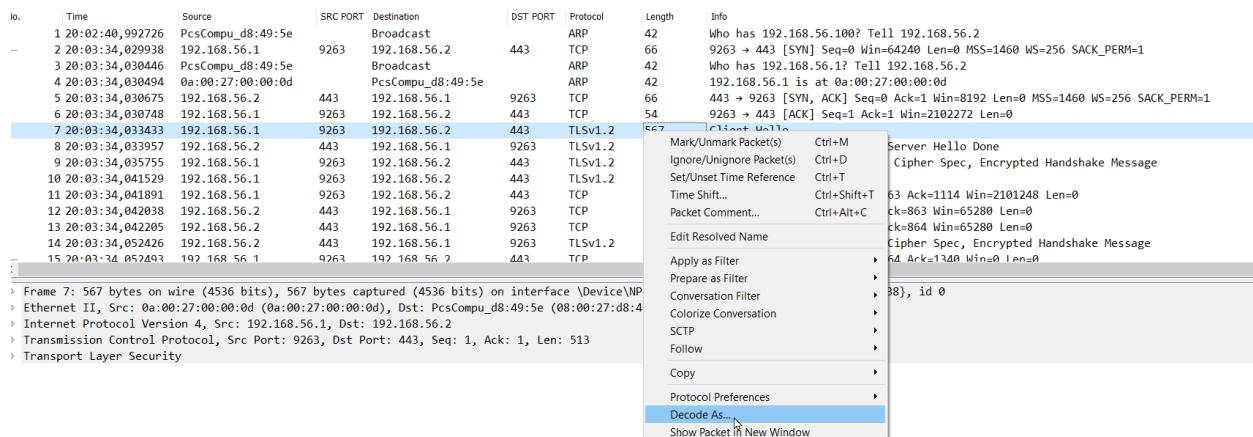
The deprecated *RSA keys list* dialog may be removed at some point. To configure keys, use the *RSA keys* dialog instead. To change the protocol for decrypted network data, right-click on a TLS packet and use *Decode As* to change the *Current* protocol for the *TLS port*. The *IP address* and *Port* fields are unused.

So navigate to Wireshark -> Edit -> Preferences -> RSA Keys

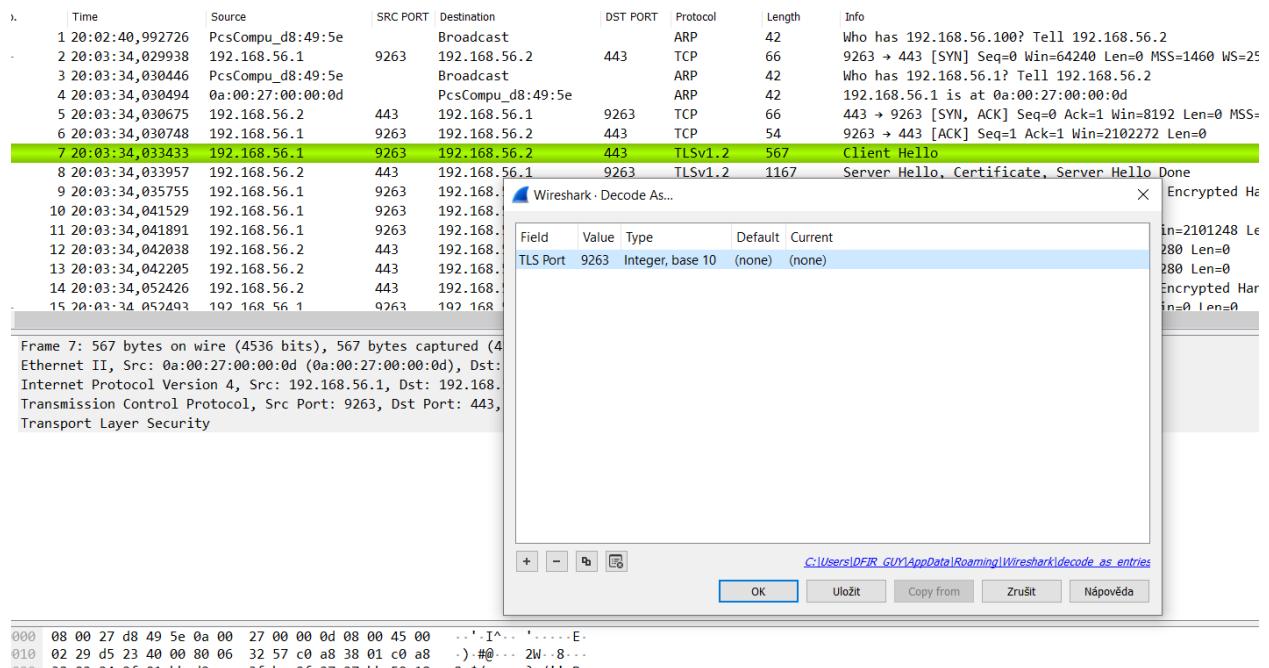
Add new key file (Our web server private key "RSA_private_KEY.pem") and confirm importing.



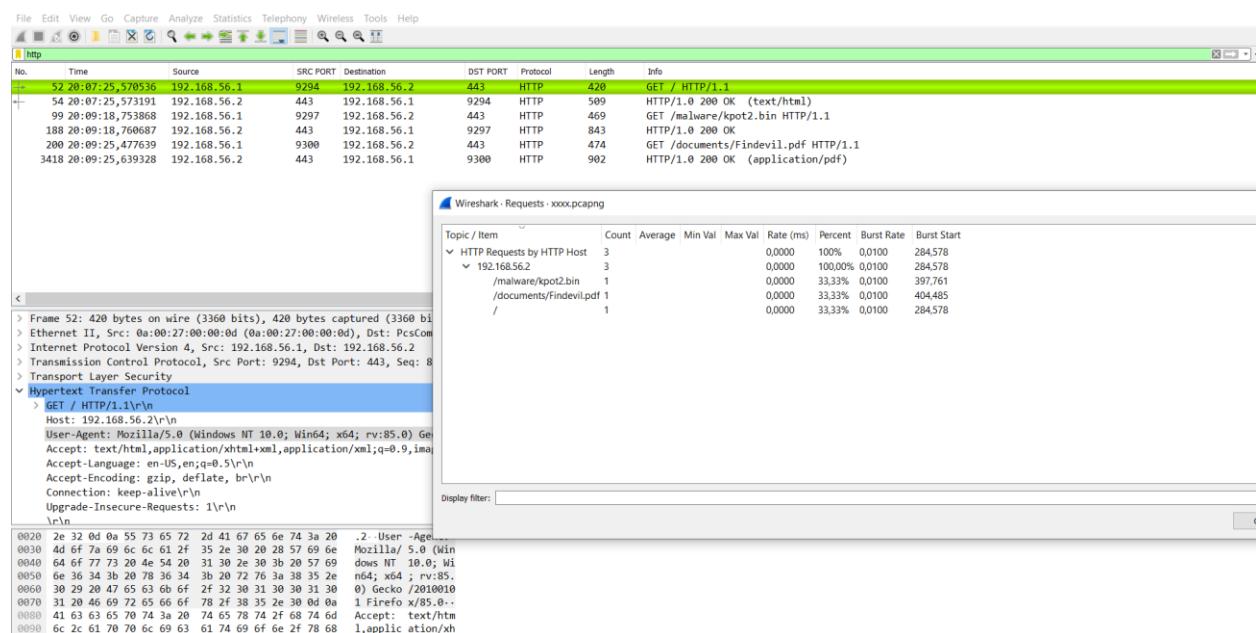
Go back to Packet list view in Wireshark and find the first packet of TLS protocol (DST IP of our web server). Mouse right click → Decode As...



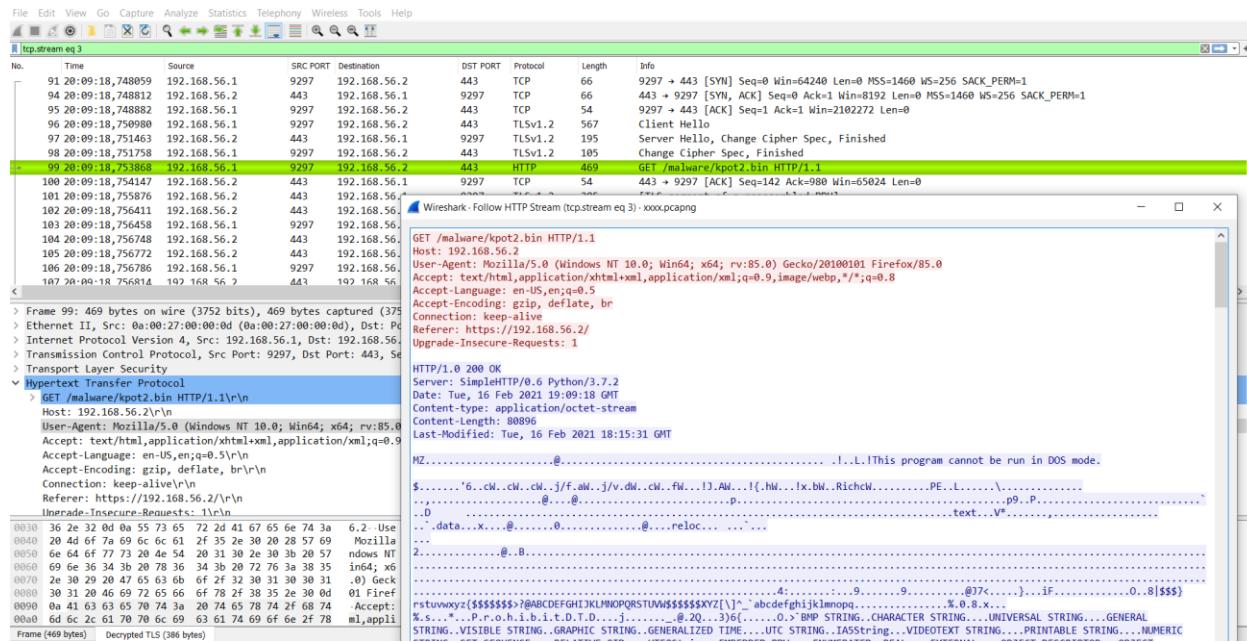
Click OK.



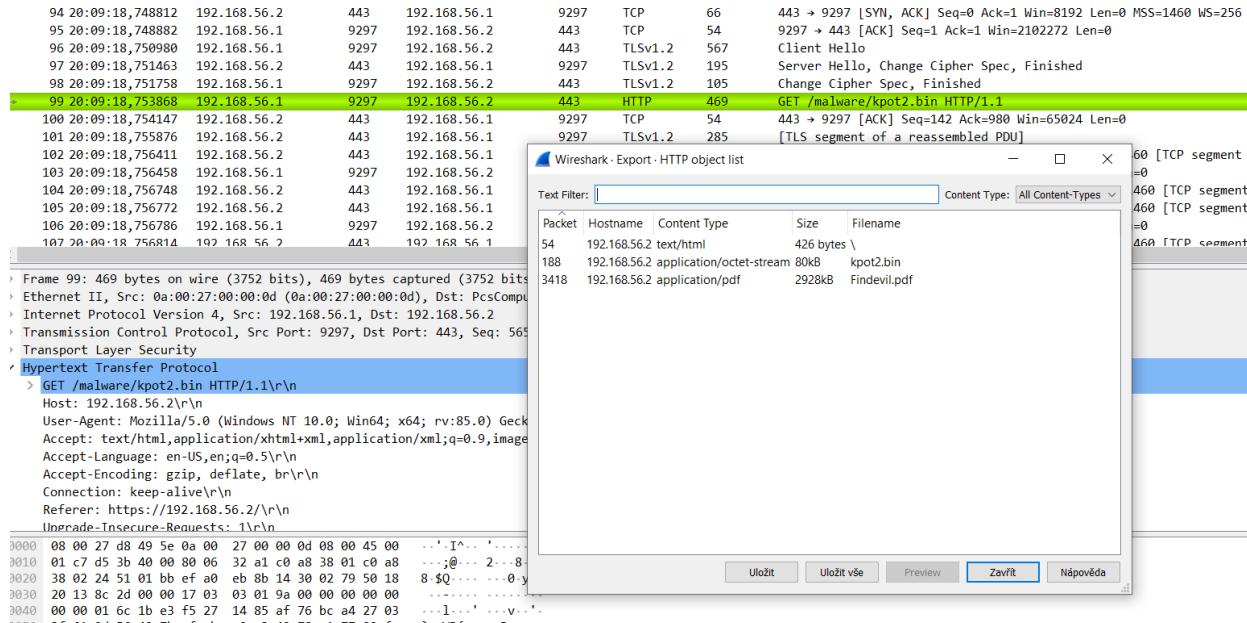
Now we can see all TLS decrypted web traffic in Wireshark.



We can see decrypted HTTP streams.



We can also export all HTTP objects as we can in normal HTTP traffic when TLS is not used.



Conclusion:

Each of method, which was shown, has its advantages and disadvantages. You can perform TLS decryption via RSA private key not only for TLS encrypted HTTP traffic but also for another protocols. In both methods, there are some restriction in use. I hope that this post will serve somebody as simple guide how one can decrypt TLS traffic in Wireshark. The main point of this was to show that even if you have the RSA private key, there are situation where it cannot be used for successful TLS decryption as mentioned in post. I would also like to thank Wireshark for its one of the best and well maintained documentation (not only for the best network protocol analyzing tool).

Useful links:

[\[https://wiki.wireshark.org/TLS\]](https://wiki.wireshark.org/TLS)

[\[https://en.wikipedia.org/wiki/Cipher_suite\]](https://en.wikipedia.org/wiki/Cipher_suite)

Author:

[\[Twitter\]](#)

[\[Github\]](#)