SharkFest '17 Europe

SSL/TLS Decryption

uncovering secrets

Wednesday November 8th, 2017

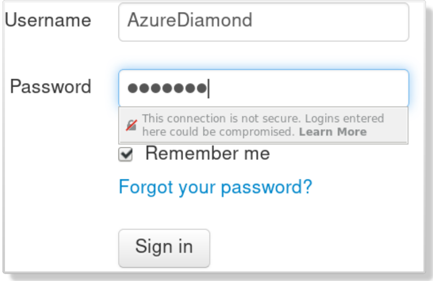
Peter Wu
Wireshark Core Developer
peter@lekensteyn.nl

About me

- Wireshark contributor since 2013, core developer since 2015.
- Areas of interest: TLS, Lua, security, . . .
- Developed a VoIP product based on WebRTC.
- Cloudflare crypto intern.

Secrets

- Things that people care about: pictures, videos, documents, email conversations, passwords, . . .
- Application Data: cookies, API keys, Request URI, User Agent, form data, response body, . . .
- How to keep these safe when sending it over the internet or over your local Wi-Fi network?



Username AzureDiamond

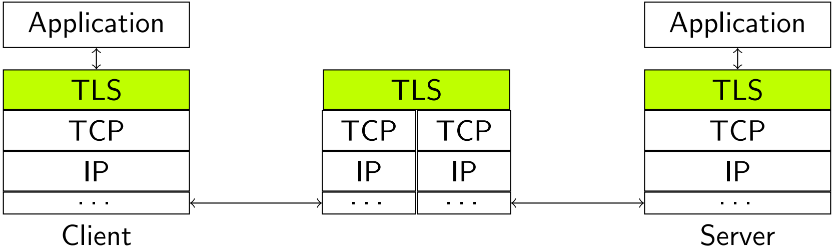
Password ●●●●●●●

This connection is not secure. Logins entered here could be compromised. **Learn More**

☑ Remember me

Transport Layer Security (TLS)

- Provides secure communication channel between two endpoints (client and server).
- Network protocol with two components:
  - Handshake Protocol: exchange capabilities, establish trust and establish keys.
  - Record Protocol: carries messages and protects application data fragments.



Secure Sockets Layer (SSL) versus Transport Layer Security (TLS)

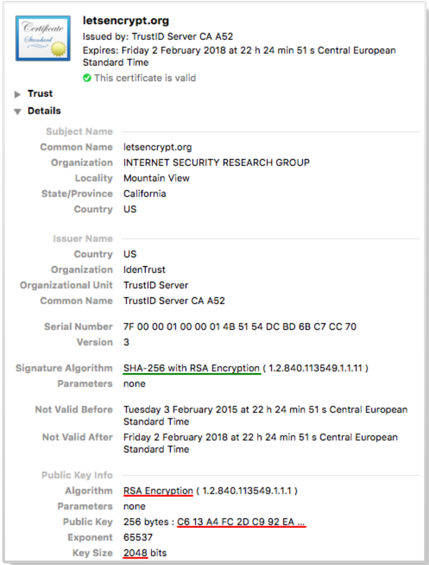
- SSLv3: old (RFC 6101, 1996) and deprecated (RFC 7568, 2015). Do not use it!
- TLS 1.0 (RFC 2246, 1999), 1.1 (RFC 4346, 2006), 1.2 (RFC 5246, 2008).
- Changes:
  - New versions are generally fixing weaknesses due to new attacks.
  - TLS 1.0 (RFC 3546, 2003) and up allow for extensions, like Server Name Indication (SNI) to support virtual hosts.
  - TLS 1.2: new authenticated encryption with additional data (AEAD) mode.
- "SSL" term still stuck, e.g. "SSL certificate", "SSL library" and field names in Wireshark (e.g. `ssl.record.content_type`).
- Mail protocols: *TLS* often refers to *STARTTLS* while *SSL* directly starts with the handshake.

“Secure” communication channel

- Symmetric-key algorithms: encrypt/decrypt bulk (application) data using a single (secret) *symmetric key*. Examples: AES, 3DES, RC4.
- How to create such a secret? For example, AES-256 needs a 256-bit key.
- Public-key cryptography: a (secret) *private key* and a related *public key*.
  - Mathematically hard to compute private key from public key.
  - Encrypt data with *public key*, decrypt with *private key*.
  - Limitation: maximum data size for RSA is equal to modulus size, 2048-4096 bits.
  - Idea: generate a random *premaster secret* and encrypt it with the *public RSA key*.
- Where to retrieve this RSA public key from?

Certificates and trust

- Public key is embedded in a X.509 certificate.
- How can this certificate be trusted?
- A Certificate Authority (issuer) signs the certificate with its private key.
- Public-key cryptography: use a private (secret) key and a public key with small data.
  - Compress data using a hash function. Examples: SHA256, SHA1, MD5.
  - Sign hash with private key, verify with public key. Examples: RSA, ECDSA.
- Root CAs are self-signed and installed by the OS vendor or local admin (Group Policy, etc.).

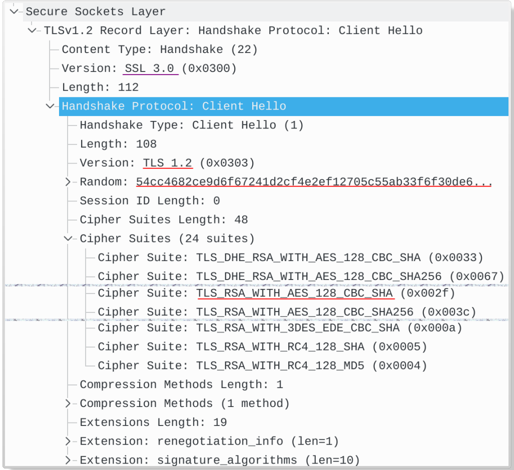


letsencrypt.org
Issued by: TrustID Server CA A52
Expires: Friday 2 February 2018 at 22 h 24 min 51 s Central European Standard Time
✓ This certificate is valid

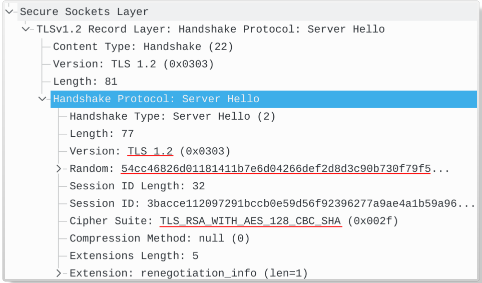
Subject Name	
Common Name	letsencrypt.org
Organization	INTERNET SECURITY RESEARCH GROUP
Locality	Mountain View
State/Province	California
Country	US
Issuer Name	
Country	US
Organization	IdenTrust
Organizational Unit	TrustID Server
Common Name	TrustID Server CA A52
Serial Number	7F 00 00 01 00 00 01 4B 51 54 DC BD 6B C7 CC 70
Version	3
Signature Algorithm	SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)
Parameters	none
Not Valid Before	Tuesday 3 February 2015 at 22 h 24 min 51 s Central European Standard Time
Not Valid After	Friday 2 February 2018 at 22 h 24 min 51 s Central European Standard Time
Public Key Info	
Algorithm	RSA Encryption (1.2.840.113549.1.1.1)
Parameters	none
Public Key	256 bytes : C6 13 A4 FC 2D C9 92 EA ...
Exponent	65537
Key Size	2048 bits

TLS handshake with RSA key exchange method

- Client Hello advertises supported parameters, Server Hello decides.
- Server picked RSA key exchange: TLS_**RSA**_WITH_AES_128_CBC_SHA.



```
Secure Sockets Layer
  TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: SSL 3.0 (0x0300)
    Length: 112
    Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 108
      Version: TLS 1.2 (0x0303)
      Random: 54cc4682ce9d6f67241d2cf4e2ef12705c55ab33f6f30de6...
      Session ID Length: 0
      Cipher Suites Length: 48
      Cipher Suites (24 suites)
        Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
        Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (0x0067)
        Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
        Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003c)
        Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
        Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
        Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
      Compression Methods Length: 1
      Compression Methods (1 method)
      Extensions Length: 19
      Extension: renegotiation_info (len=1)
      Extension: signature_algorithms (len=10)
```



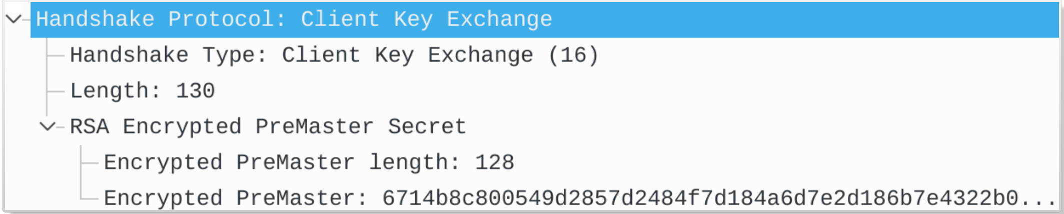
```
Secure Sockets Layer
  TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 81
    Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 77
      Version: TLS 1.2 (0x0303)
      Random: 54cc46826d01181411b7e6d04266def2d8d3c90b730f79f5...
      Session ID Length: 32
      Session ID: 3bacce112097291bccb0e59d56f92396277a9ae4a1b59a96...
      Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
      Compression Method: null (0)
      Extensions Length: 5
      Extension: renegotiation_info (len=1)
```

+ Certificate (with RSA public key)

+ ServerHelloDone

TLS handshake with RSA key exchange method - ClientKeyExchange

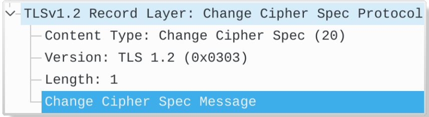
- Client receives Server Hello, knows protocol version and cipher suite.
- Client generates a new random 48-byte **premaster secret**, encrypts it using the *public key* from the Certificate and sends the result to the server in a *ClientKeyExchange* message.
- Using the private RSA key, server (or anyone else!) decrypts the premaster secret.



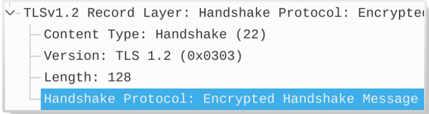
```
Handshake Protocol: Client Key Exchange
    Handshake Type: Client Key Exchange (16)
    Length: 130
    RSA Encrypted PreMaster Secret
        Encrypted PreMaster length: 128
        Encrypted PreMaster: 6714b8c800549d2857d2484f7d184a6d7e2d186b7e4322b0...
```

TLS handshake with RSA key exchange method - Finishing up

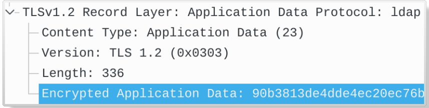
- Both sides calculate the 48-byte **master secret** based on the Client Random, Server Random and the premaster secret.
- Both sides derive symmetric keys from this master secret, send the *ChangeCipherSpec* message to start record protection.
- Finally they both finish the Handshake protocol by sending a *Finished* Handshake message over the encrypted record layer.
- Now the actual encrypted *Application Data* can be sent and received.



```
TLSv1.2 Record Layer: Change Cipher Spec Protocol
    Content Type: Change Cipher Spec (20)
    Version: TLS 1.2 (0x0303)
    Length: 1
    Change Cipher Spec Message
```



```
TLSv1.2 Record Layer: Handshake Protocol: Encrypte
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 128
    Handshake Protocol: Encrypted Handshake Message
```



```
TLSv1.2 Record Layer: Application Data Protocol: ldap
    Content Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 336
    Encrypted Application Data: 90b3813de4dde4ec20ec76b
```

Handshake overview

```
Client                                      Server

ClientHello         -------->
                                       ServerHello
                                      Certificate*
                                ServerKeyExchange*
                    <--------      ServerHelloDone
ClientKeyExchange
[ChangeCipherSpec]
Finished            -------->
                                [ChangeCipherSpec]
                    <--------             Finished
Application Data    <------->     Application Data
```

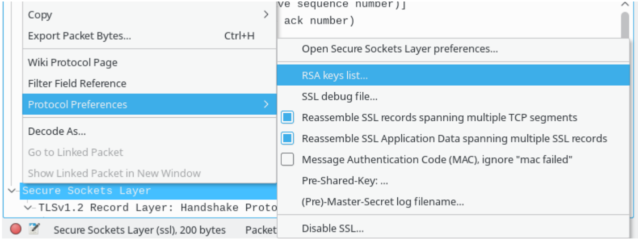
Simplified TLS handshake (adapted from RFC 5246 (TLS 1.2))

Plaintext please

- Server administrators can check application logs.
- Web browsers provide developer tools.
- What if the information is not logged?
- What if you want to know what this third-party Android app is doing?
- What if the application under investigation is poorly documented?
- What if you want to debug your new HTTP/2 feature?
- Solution: packet capture plus SSL/TLS secrets!

Decryption using private RSA server key

Configure Wireshark with RSA private key file[1]:



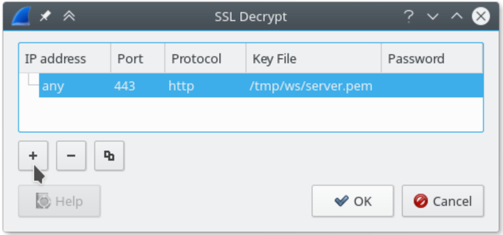
```
ve sequence number)]
ack number)
Secure Sockets Layer
TLSv1.2 Record Layer: Handshake Proto
```

☑ Reassemble SSL records spanning multiple TCP segments

☑ Reassemble SSL Application Data spanning multiple SSL records

☐ Message Authentication Code (MAC), ignore "mac failed"

Secure Sockets Layer (ssl), 200 bytes Packet



IP address	Port	Protocol	Key File	Password
any	443	http	/tmp/ws/server.pem	

- IP address is unused and ignored. Port + Protocol can be empty. These three fields will be removed in future.
- Specify PEM-encoded key file or PKCS#12 Key File + Password.

[1] See `https://wiki.wireshark.org/SSL#Preference_Settings`

Limitations of RSA private key

- Clients usually do not have access to the RSA key, only server operators can use it.
- In case of mutual authentication (client certificates), the private key is only used for *signing*. The client private RSA key cannot decrypt.
- Encrypted premaster secret is not sent with resumed sessions.

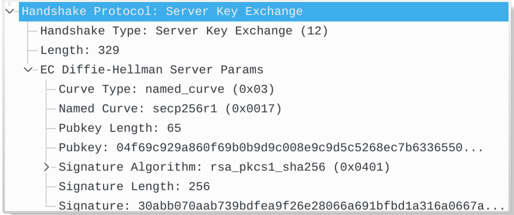
```
Client                                                  Server

ClientHello                   -------->
                                                   ServerHello
                                            [ChangeCipherSpec]
                              <--------               Finished
[ChangeCipherSpec]
Finished                      -------->
Application Data              <------->       Application Data
```

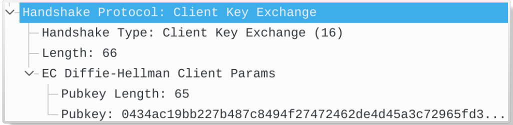
Message flow for an abbreviated handshake (RFC 5246, Figure 2)

Ephemeral (Elliptic Curve) Diffie-Hellman (ECDHE)

- Decryption using RSA private key not possible with cipher suites like TLS_EC**DHE**_ECDSA_WITH_AES_128_GCM_SHA256 and TLS_EC**DHE**_RSA_WITH_AES_128_GCM_SHA256.
- Although it has *RSA* in its name, it is not used for encryption, but signing.
- Instead it uses *Diffie-Hellman* to establish a shared secret (the **premaster secret**) based on *ephemeral* secrets (different secrets for every session).
- Server chooses a group/curve, generates private value and its related public value and sends it to the client. Client uses same group/curve and also generates a pair.
- Computationally hard to find the private value given the public one.



```
Handshake Protocol: Server Key Exchange
    Handshake Type: Server Key Exchange (12)
    Length: 329
    EC Diffie-Hellman Server Params
        Curve Type: named_curve (0x03)
        Named Curve: secp256r1 (0x0017)
        Pubkey Length: 65
        Pubkey: 04f69c929a860f69b0b9d9c008e9c9d5c5268ec7b6336550...
        Signature Algorithm: rsa_pkcs1_sha256 (0x0401)
        Signature Length: 256
        Signature: 30abb070aab739bdfea9f26e28066a691bfbd1a316a0667a...
```



```
Handshake Protocol: Client Key Exchange
    Handshake Type: Client Key Exchange (16)
    Length: 66
    EC Diffie-Hellman Client Params
        Pubkey Length: 65
        Pubkey: 0434ac19bb227b487c8494f27472462de4d45a3c72965fd3...
```

TLS secrets summary

- Any of these can be used for decryption with passive captures:
  - premaster secret: RSA-encrypted or output from DH key exchange
  - Master secret: derived from premaster secret and handshake context. Also used for session resumption.
  - Symmetric encryption key for record encryption.
  - RSA private key file (for RSA key exchange, covered before).
- So how to use the master secrets?

SSL key log file

- Text file with master secrets[2].
- Works for any cipher, including RSA and DHE.
- Clients can use this too!
- Set environment variable `SSLKEYLOGFILE` before starting Firefox or Chrome. The variable is only read during startup, so restart if necessary.
- Format: `CLIENT_RANDOM` *<Client Hello Random> <master secret>*.

```
# SSL/TLS secrets log file, generated by NSS
CLIENT_RANDOM 5f4dad779789bc5142cacf54f5dafba0a06235640796f40048ce4d0d1df63ad8 a4d69a3fa4222d6b6f2492e66dca2b1fc4e2bc143df849ad45eff9f
CLIENT_RANDOM c2407d5ba931798e3a35f775725fb3e5aefcb5804bb50271fe3bd5fb19c90061 e419759e7b44f766df6defe6b656eda3d430754044773b6fc0a91eb
CLIENT_RANDOM abec6cf83ea1dcb135b21fd94bc0120dd6a37dca0fcd96efd8989d05c51cc3ab 5b4d525dfe3168132d388881033633c2aba99346c25ae8163f2191f
CLIENT_RANDOM dffe2c85a7d6f3c3ec34ba52ea710f0f1649e58afa02f9824d983ea74f07900e fdb58d49482f876f200ce680b9d6987434e3aca54d203fc57cc5888
CLIENT_RANDOM fbf40ada961093cd917fba97bfffe7c4b0bbf57a0cf90626dee417d3d12b3755 6b4e313d6be9316c42f47ddd3ceeef9743825bd3c3bb25ec9ac73c9
CLIENT_RANDOM 2b8184f7642df4bb5979ad9a623690b08f392deb94fdb64b00d7dc78b711638b dfdbe9f4d6949eea02489eb39b2c8d7770c12928becaf0ac1e34edf
CLIENT_RANDOM 7e4340c76c720d39c98e761697be0f32e1c79c6c04ade05a3f29325ac9cae612 1dfe402b85560048ae278b78febe83ee1640785b969c328d94a785a
```

[2] File format at https://developer.mozilla.org/NSS_Key_Log_Format

Using SSL key log file in Wireshark

- Configure file in Wireshark preferences: Edit → Preferences; Protocols → SSL; (Pre-)Master Secret log filename.
- Key log file is also read during a live capture. And if the file is removed and a new file is written, the new key log file is automatically read.
- Caveat: key log is read while processing ChangeCipherSpec. If key is written too late, trigger a redissection (e.g. change a preference or (Un)ignore a packet).

Application and library support

- Any application built using NSS and GnuTLS enable key logging via the `SSLKEYLOGFILE` environment variable.
- Applications using OpenSSL 1.1.1 or BoringSSL d28f59c27bac (2015-11-19) can be configured to dump keys:

```
void SSL_CTX_set_keylog_callback(SSL_CTX *ctx,
    void (*cb)(const SSL *ssl, const char *line));
```

- cURL supports many TLS backends, including NSS, GnuTLS and OpenSSL. Key logging with OpenSSL/BoringSSL is possible with curl 7.56.0[3]

[3] Requires a build time option, see `https://curl.haxx.se/bug/?i=1866`

Key log with OpenSSL 1.1.0 and before

- Why: many applications (including Python) use OpenSSL.
- Problem: older OpenSSL versions have no key log callback.
- Solution: intercept library calls using a debugger or an interposing library (LD_PRELOAD) and dump keys[4].
- Example with OpenSSL 1.1.0f using an intercepting library[5]:

```
$ export SSLKEYLOGFILE=some.keys LD_PRELOAD=./libsslkeylog.so
$ curl https://example.com
...
$ cat some.keys
CLIENT_RANDOM 12E0F5085A89004291A679ABE8EE1508193878AB9E909745CA032212FCA24B89 148AF5875F83
```

[4] https://security.stackexchange.com/q/80158/2630

[5] https://git.lekensteyn.nl/peter/wireshark-notes/tree/src

Unsupported applications for SSLKEYLOGFILE

- Windows native TLS library is Secure Channel (SChannel). Pending feature request for Microsoft Edge browser: [6]
- Extracting secrets from SChannel is not impossible (but neither easy) though[7]
- Apple macOS applications use SecureTransport, also not supported.

[6] https://wpdev.uservoice.com/forums/257854-microsoft-edge-developer/suggestions/16310230-ssl-key-logging-aka-sslkeylogfile

[7] https://www.blackhat.com/docs/us-16/materials/us-16-Kambic-Cunning-With-CNG-Soliciting-Secrets-From-SChannel.pdf

Alternative ways to get the secret

- Force RSA key exchange (disable forward-secret cipher suites).
- Setup a fake CA and force traffic through a proxy like mitmproxy[8], OWASP Zap, Fiddler or Burp Suite.
- All of these methods can be detected by the client. Certificate pinning can also defeat the custom CA method.
- The proxy interception method may also weaken security[9].
- If you are really serious about a passive, nearly undetectable attack from a hypervisor, see the TeLeScope experiment[10].

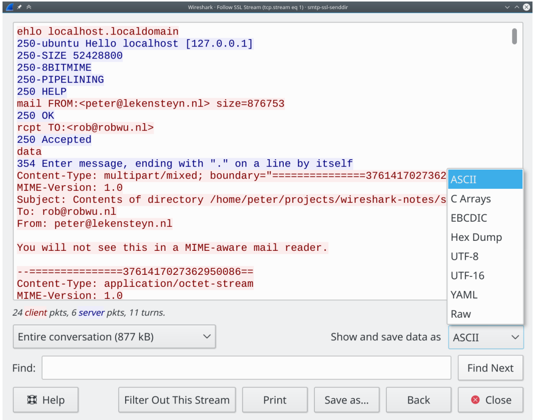
[8] http://docs.mitmproxy.org/en/stable/dev/sslkeylogfile.html

[9] Durumeric et. al., The Security Impact of HTTPS Interception, https://jhalderm.com/pub/papers/interception-ndss17.pdf

[10] https://conference.hitb.org/hitbsecconf2016ams/sessions/telescope-peering-into-the-depths-of-tls-traffic-in-real-time/

Feature: Follow SSL Stream

- Display the contents of the decrypted application data.
- Right-click in the packet list or details view, *Follow* → *SSL Stream*.
- Great for text-based protocols like SMTP. For binary data, try the *Hex Dump* option.
- Click on data to jump to related packet (in packet list). Note that display filter can hide packet, clear the filter to avoid that.



Wireshark · Follow SSL Stream (tcp.stream eq 1) · smtp-ssl-senddir

```
ehlo localhost.localdomain
250-ubuntu Hello localhost [127.0.0.1]
250-SIZE 52428800
250-8BITMIME
250-PIPELINING
250 HELP
mail FROM:<peter@lekensteyn.nl> size=876753
250 OK
rcpt TO:<rob@robwu.nl>
250 Accepted
data
354 Enter message, ending with "." on a line by itself
Content-Type: multipart/mixed; boundary="===============3761417027362
MIME-Version: 1.0
Subject: Contents of directory /home/peter/projects/wireshark-notes/s
To: rob@robwu.nl
From: peter@lekensteyn.nl

You will not see this in a MIME-aware mail reader.

--===============3761417027362950086==
Content-Type: application/octet-stream
MIME-Version: 1.0
```

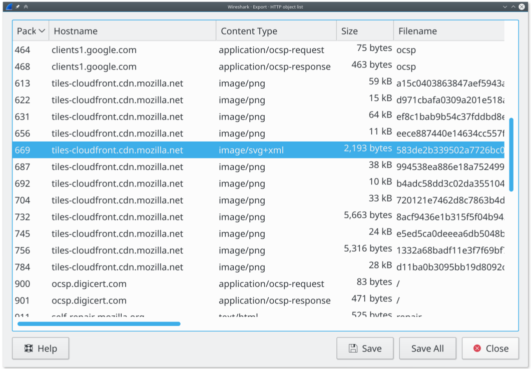
24 client pkts, 6 server pkts, 11 turns.

Entire conversation (877 kB)

Show and save data as: ASCII

Feature: Export HTTP Objects

- After decryption is enabled, HTTP payloads within TLS (HTTPS) can be exported.
- *File* → *Export Objects* → *HTTP...*
- Click on an item to select it in the packet list.
- Note: does not cover HTTP2 or QUIC (yet?) as of Wireshark 2.4.



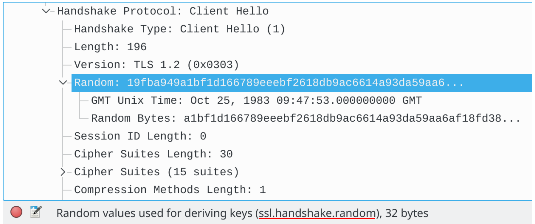
Pack	Hostname	Content Type	Size	Filename
464	clients1.google.com	application/ocsp-request	75 bytes	ocsp
468	clients1.google.com	application/ocsp-response	463 bytes	ocsp
613	tiles-cloudfront.cdn.mozilla.net	image/png	59 kB	a15c0403863847aef5943a
622	tiles-cloudfront.cdn.mozilla.net	image/png	15 kB	d971cbafa0309a201e518a
631	tiles-cloudfront.cdn.mozilla.net	image/png	64 kB	ef8c1bab9b54c37fddbd8e
656	tiles-cloudfront.cdn.mozilla.net	image/png	11 kB	eece887440e14634cc557f
669	tiles-cloudfront.cdn.mozilla.net	image/svg+xml	2,193 bytes	583de2b339502a7726bc0
687	tiles-cloudfront.cdn.mozilla.net	image/png	38 kB	994538ea886e18a752499
692	tiles-cloudfront.cdn.mozilla.net	image/png	10 kB	b4adc58dd3c02da355104
704	tiles-cloudfront.cdn.mozilla.net	image/png	33 kB	720121e7462d8c7863b4d
732	tiles-cloudfront.cdn.mozilla.net	image/png	5,663 bytes	8acf9436e1b315f5f04b94
745	tiles-cloudfront.cdn.mozilla.net	image/png	24 kB	e5ed5ca0deeea6db5048b
756	tiles-cloudfront.cdn.mozilla.net	image/png	5,316 bytes	1332a68badf11e3f7f69bf
784	tiles-cloudfront.cdn.mozilla.net	image/png	28 kB	d11ba0b3095bb19d8092
900	ocsp.digicert.com	application/ocsp-request	83 bytes	/
901	ocsp.digicert.com	application/ocsp-response	471 bytes	/
911	self-repair.mozilla.org	text/html	525 bytes	repair

Feature: Export SSL Session Keys

- Suppose you have a capture which is decrypted using a RSA private key file. How to allow others to decrypt data without handing over your RSA private key file?
- *File* → *Export SSL Session Keys...*
- Generates a key log file which can be used instead of the private RSA key file.
- Note: currently contains all keys. Remove lines which are not needed (match by the second field, the Random field from Client Hello).

Feature: Display Filters

- Display filters can be used for filtering, columns and coloring rules.
- Discover by selecting a field in packet list, look in status bar.
- Recognize TCP/TLS stream in packet list: Right-click *TCP Stream Index* (`tcp.stream`) field in packet details, *Apply as Column*.
- Right-click field in packet details, *Apply/Prepare as Filter*.
- SNI in Client Hello: `ssl.handshake.extensions_server_name`
- Change in Wireshark 2.4: `ssl.handshake.random` selects full Client or Server Random instead of the just the Random Bytes field. Reason: real time is often no longer included, full bytes field is useful for matching with key log file.

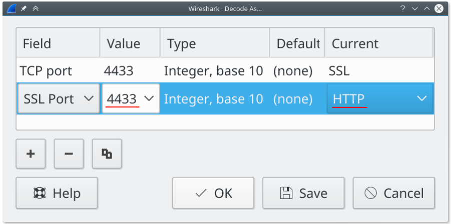


```
v Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 196
    Version: TLS 1.2 (0x0303)
  v Random: 19fba949a1bf1d166789eeebf2618db9ac6614a93da59aa6...
      GMT Unix Time: Oct 25, 1983 09:47:53.000000000 GMT
      Random Bytes: a1bf1d166789eeebf2618db9ac6614a93da59aa6af18fd38...
    Session ID Length: 0
    Cipher Suites Length: 30
  > Cipher Suites (15 suites)
    Compression Methods Length: 1
```

Random values used for deriving keys (ssl.handshake.random), 32 bytes

Feature: Decode As

- Force dissector for custom ports. Decode as SSL (TCP) or DTLS (UDP).
- Select application data protocol within SSL/TLS layer (since Wireshark 2.4).
- Example: HTTPS on non-standard TCP server port 4433.
  - Right-click TCP layer, *Decode As*. Change current protocol for **TCP Port** to *SSL*.
  - Press *OK* to apply just for now or *Save* to persist this port-to-protocol mapping.
  - Right-click SSL layer, *Decode As*. Change current protocol for **SSL Port** to *HTTP*.
- For STARTTLS protocols, select SMTP/IMAP/. . . instead of SSL for *TCP Port*.
- Tip: there are many protocols, just select the field, then use arrow keys or type the protocol name (typing *H* gives *HTTP*).



Wireshark · Decode As...

Field	Value	Type	Default	Current
TCP port	4433	Integer, base 10	(none)	SSL
SSL Port	4433	Integer, base 10	(none)	HTTP

Feature: Tshark

- Tshark: command-line tool, useful to extract information as text, especially when the query is repeated multiple times.
- Find all cipher suites as selected by the server:
  ```
  tshark -r some.pcap -Tfields -e ssl.handshake.ciphersuite -Y ssl.handshake.type==2
  ```
- List all protocol fields: `tshark -G fields`
- Configure keylogfile:
  ```
  tshark -ossl.keylog_file:firefox.keys -r firefox.pcapng.gz
  ```
- Configure RSA keyfile (fields correspond to the RSA keys dialog):
  ```
  wireshark -ouat:ssl_keys:’"","","","keys/rsasnakeoil2.key",""’
  ```
- Tshark manual: `https://www.wireshark.org/`

Future: TLS 1.3

- Replaces all previous cipher suites with new one. Dropped all old cipher suites (no more CBC, RC4, NULL, export ciphers).
- RSA key exchange is gone, all ciphers are forward secret.
- Encrypted early (0-RTT) data.
- Encrypted server extensions (like ALPN)
- Encrypted server certificate.
- Multiple derived secrets for resumption, handshake encryption, application data encryption. (Safer resumption!)
- Decryption and dissection is supported by Wireshark 2.4 (drafts 18-21 as of Wireshark 2.4.2).

Conclusions

- RSA private keys cannot be used for decryption in all cases.
- The key log method (`SSLKEYLOGFILE`) can also be used by clients and works with all cipher suites.
- TLS 1.3 debugging is even more difficult without decryption.
- Use latest Wireshark version, especially if you are doing any TLS 1.3 work.