Analysing TLS traffic

Tufa Alexandru Daniel 1628927

1 December 2017

1 Introduction

The Transport Layer Security, TLS, and its predecessor, Secure Sockets Layer, SSL, are two protocols that offer the possibility to communicate in a secure way over a computer network. The TLS protocol is primarily used to provide privacy and data integrity between two communicating computer applications. Before creating this secure communication, a handshaking protocol has to be done between the two entities that, for simplicity, we'll call client and server, where the client is the application that wants to start the communication with the other entity, the server. To fully understand this handshaking protocol we'll analyse it through a packet sniffer, Wireshak [1], between a client, Google Chrome browser, and a server, the Wikipedia[1] web page. Let's see an overview of this protocol.

2 Handshaking protocol

The first to start the protocol is the client with a "Client Hello" message that presents information such as the SSL/TSL version, a nonce or challenge in order to calculate the MasterSecret, the CipherSuites supported by the client and, optionally, the data compression methods supported. The server responds with a "Server Hello" message that contains the CipherSuite chosen by the server from the list provided by the client, the session ID and it's own nonce or challenge. To authenticate itself the server also sends its digital certificate and, if it requires a digital certificate for client authentication, also sends a "Client Certificate Request". The client verifies the server's digital certificate and sends a random byte string encrypted with the server's public key. This, together with the previous random numbers

[1]https://www.wikipedia.org/

used, will generate the MasterSecret that will allow both parties to communicate securely. If the server sent a "Client Certificate Request" then the client sends a random byte string encrypted with the client's private key together with the client's digital certificate. If this has happened the server verifies the certificate. In the end the client sends the server a "finished" message that is encrypted with the secret key, indicating that the client part of the handshake is complete. After the same has been done by the server, they can start communicating through messages that are symmetrically encrypted with the shared secret key.

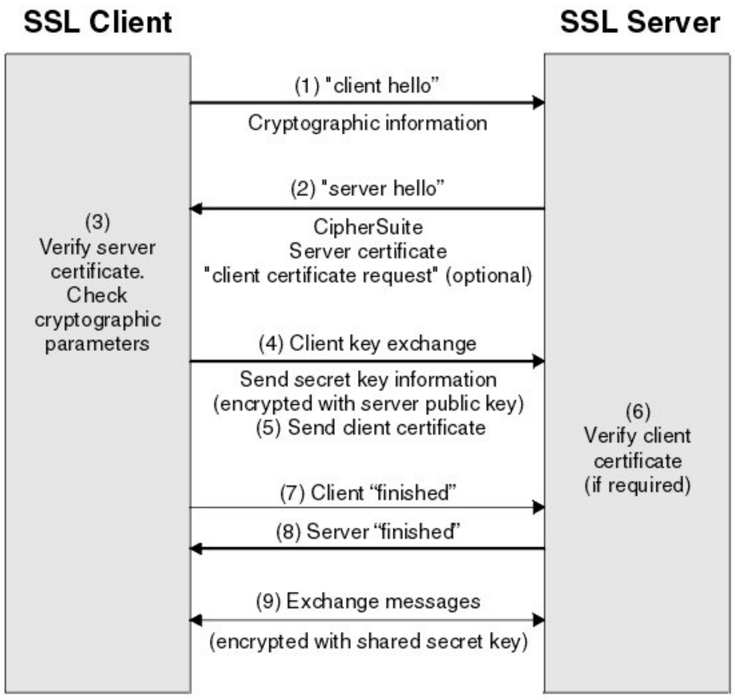

SSL Client
SSL Server
(1) "client hello"
Cryptographic information
(2) "server hello"
CipherSuite
Server certificate
"client certificate request" (optional)
(3) Verify server certificate. Check cryptographic parameters
(4) Client key exchange
Send secret key information
(encrypted with server public key)
(5) Send client certificate
(6) Verify client certificate (if required)
(7) Client "finished"
(8) Server "finished"
(9) Exchange messages
(encrypted with shared secret key)

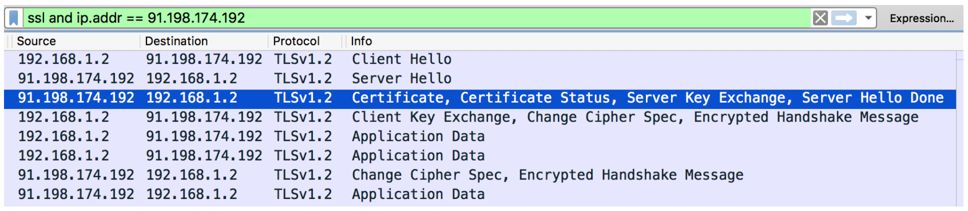
Figure 1: Handshaking Protocol

3 Analysis of the traffic

To make our analysis easier we have used two filters for Wireshark:

- ssl
- ip.addr == 91.198.174.192

So that only the packets concerning the SSL/TLS protocol and with IP destination or source from Wikipedia will appear.



```
ssl and ip.addr == 91.198.174.192                                          Expression...
Source          Destination     Protocol  Info
192.168.1.2     91.198.174.192  TLSv1.2   Client Hello
91.198.174.192  192.168.1.2     TLSv1.2   Server Hello
91.198.174.192  192.168.1.2     TLSv1.2   Certificate, Certificate Status, Server Key Exchange, Server Hello Done
192.168.1.2     91.198.174.192  TLSv1.2   Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
192.168.1.2     91.198.174.192  TLSv1.2   Application Data
192.168.1.2     91.198.174.192  TLSv1.2   Application Data
91.198.174.192  192.168.1.2     TLSv1.2   Change Cipher Spec, Encrypted Handshake Message
91.198.174.192  192.168.1.2     TLSv1.2   Application Data
```

Figure 2: Wireshark packets

Let's analyse them one by one.

3.1 Client Hello

This is the first message in our TLS handshake. We can see the version of the TLS protocol that is the highest supported by the client, in particular TLS 1.2. The client *Random* number is the one used to calculate the Master secret. We can also observe the list of cipher suites supported by the client. For example the first one is

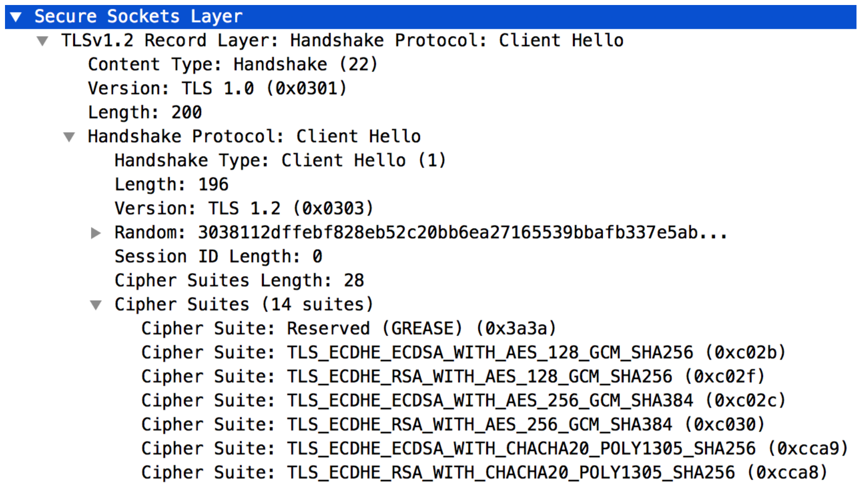
$$TLS\_ECDHE\_ECDS\_WITH\_AES\_128\_GCM\_SHA256$$

which indicates that the client wants to use as key exchange algorithm the Ellipctic Curve Diffie-Hellman Exchange, for the Digital Signature algorithm it prefers the Elliptic Curve Digital Signature Algorithm, for the bulk encryption algorithm the client prefers AES-128 with Galois/Counter Mode mode for authentication and finally SHA-256 as the cryptographic hash in the Message Authentication Code. The client also, by including an extension of type "status_request", indicates that it wants a certificate status message which contains the OCSP[2] response.

3.2 Server Hello

With this message the server informs the client about the highest TLS protocol version supported which is also supported by the client itself and will be the version ultimately used for the communication. It also provides its own Random number that will be used to generate the Master secret, the Session

[2]The Online Certificate Status Protocol is an Internet protocol used for obtaining the revocation status of an X.509 digital certificate.



```
▼ Secure Sockets Layer
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 200
    ▼ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 196
        Version: TLS 1.2 (0x0303)
      ▶ Random: 3038112dffebf828eb52c20bb6ea27165539bbafb337e5ab...
        Session ID Length: 0
        Cipher Suites Length: 28
      ▼ Cipher Suites (14 suites)
          Cipher Suite: Reserved (GREASE) (0x3a3a)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
          Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
```

Figure 3: Client Hello

ID generated to identify the current session and the chosen strongest cipher suite that both server and client support. For this connection the chosen one is

$$TLS\_ECDHE\_ECDSA\_WITH\_CHACHA20\_POLY1305\_SHA256$$

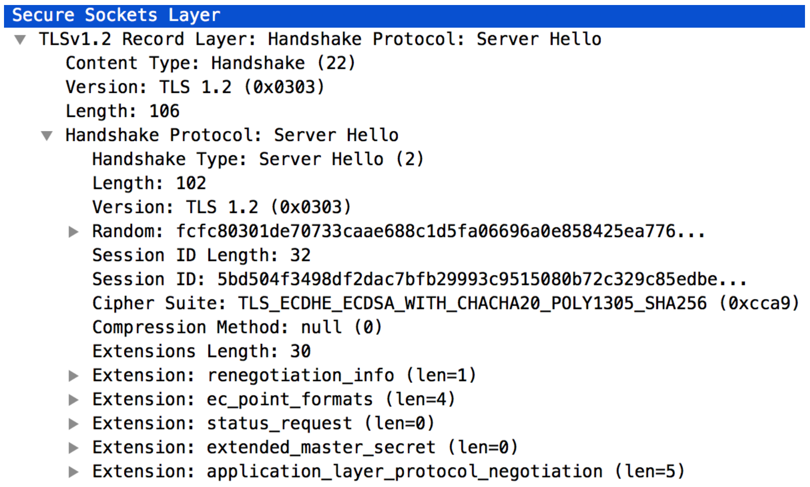
that indicates that the following parameters will be used: the Elliptic Curve Diffie-Hellman key change together with Elliptic Curve Digital Signature Algorithm with ChaCha [2] as stream cipher together with Poly1305 [3] as message authentication code and SHA-256 as hashing function.

3.3 Certificate, Certificate Status, Server Key Exchange, Server Hello Done

This packet is divide into 4 parts.

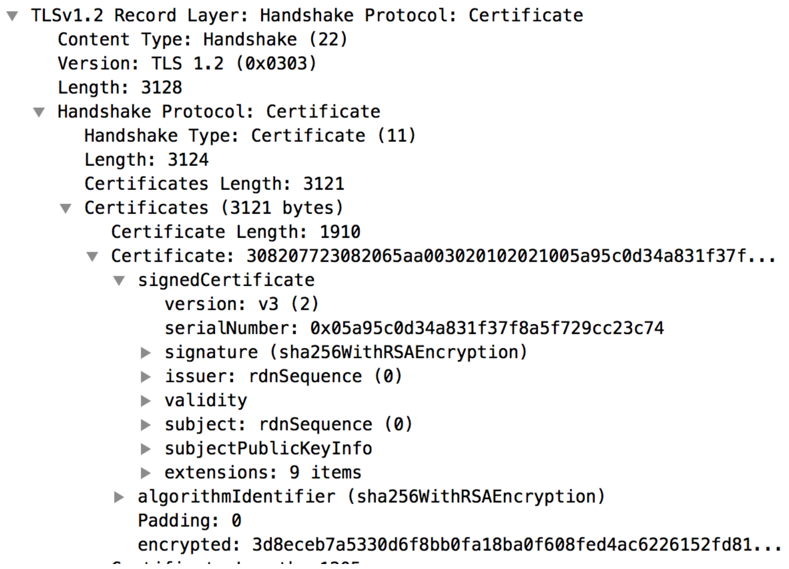
3.3.1 Certificate

In the first part the X.509 certificate is sent according to its standard. We can observe all its attributes such as version, serialNumber, signature, issuer, validity and subject. The server, together with its own certificate, sends the certificate of the issuer, the Certification Authority and we can see that it's called DigiCert High Assurance.



```
Secure Sockets Layer
▼ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 106
  ▼ Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 102
      Version: TLS 1.2 (0x0303)
    ▶ Random: fcfc80301de70733caae688c1d5fa06696a0e858425ea776...
      Session ID Length: 32
      Session ID: 5bd504f3498df2dac7bfb29993c9515080b72c329c85edbe...
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
      Compression Method: null (0)
      Extensions Length: 30
    ▶ Extension: renegotiation_info (len=1)
    ▶ Extension: ec_point_formats (len=4)
    ▶ Extension: status_request (len=0)
    ▶ Extension: extended_master_secret (len=0)
    ▶ Extension: application_layer_protocol_negotiation (len=5)
```

Figure 4: Server Hello



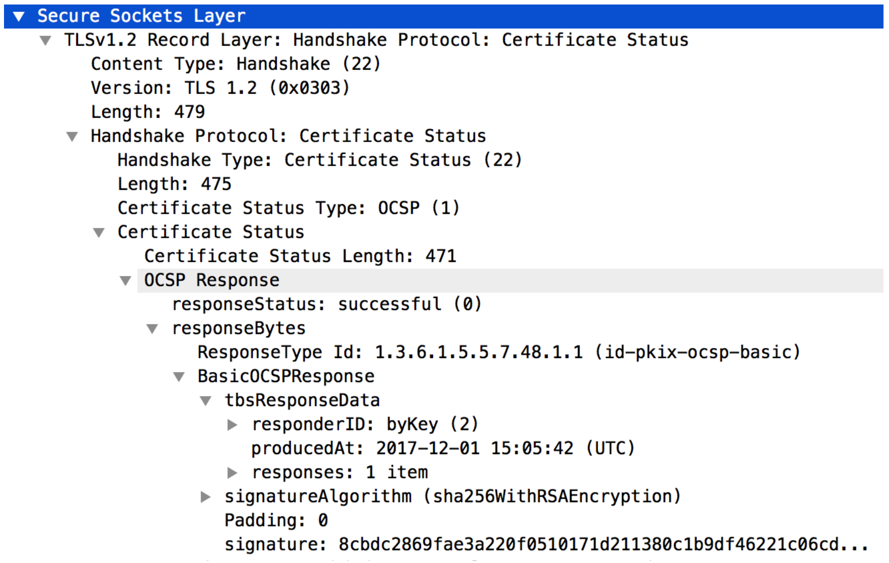
```
▼ TLSv1.2 Record Layer: Handshake Protocol: Certificate
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 3128
  ▼ Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 3124
      Certificates Length: 3121
    ▼ Certificates (3121 bytes)
        Certificate Length: 1910
      ▼ Certificate: 308207723082065aa003020102021005a95c0d34a831f37f...
        ▼ signedCertificate
            version: v3 (2)
            serialNumber: 0x05a95c0d34a831f37f8a5f729cc23c74
          ▶ signature (sha256WithRSAEncryption)
          ▶ issuer: rdnSequence (0)
          ▶ validity
          ▶ subject: rdnSequence (0)
          ▶ subjectPublicKeyInfo
          ▶ extensions: 9 items
        ▶ algorithmIdentifier (sha256WithRSAEncryption)
          Padding: 0
          encrypted: 3d8eceb7a5330d6f8bb0fa18ba0f608fed4ac6226152fd81...
```

Figure 5: Certificate

3.3.2 Certificate Status

The Certificate Status message validates whether the server's X.509 digital certificate is revoked or not and this is ascertained by contacting a designated OCSP. This is done because the client has asked for it with the

”status_request” extension.

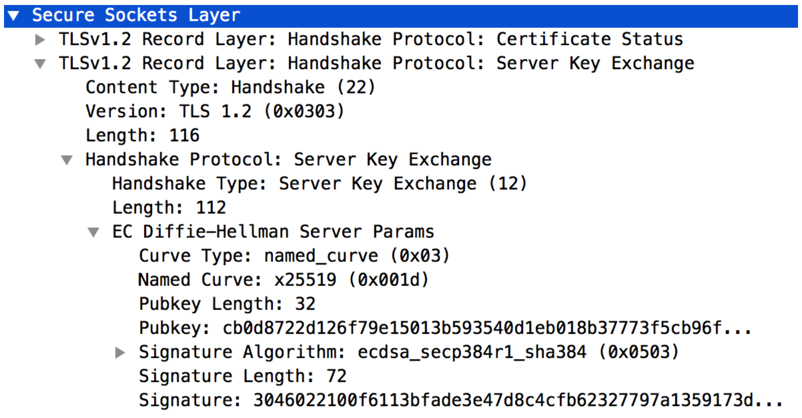


```
▼ Secure Sockets Layer
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Certificate Status
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 479
    ▼ Handshake Protocol: Certificate Status
        Handshake Type: Certificate Status (22)
        Length: 475
        Certificate Status Type: OCSP (1)
      ▼ Certificate Status
          Certificate Status Length: 471
        ▼ OCSP Response
            responseStatus: successful (0)
          ▼ responseBytes
              ResponseType Id: 1.3.6.1.5.5.7.48.1.1 (id-pkix-ocsp-basic)
            ▼ BasicOCSPResponse
              ▼ tbsResponseData
                ▶ responderID: byKey (2)
                  producedAt: 2017-12-01 15:05:42 (UTC)
                ▶ responses: 1 item
              ▶ signatureAlgorithm (sha256WithRSAEncryption)
                Padding: 0
                signature: 8cbdc2869fae3a220f0510171d211380c1b9df46221c06cd...
```

Figure 6: Certificate Status

3.4 Server Key Exchange

This message is used to provide the public key that will be used during the Key Exchange.

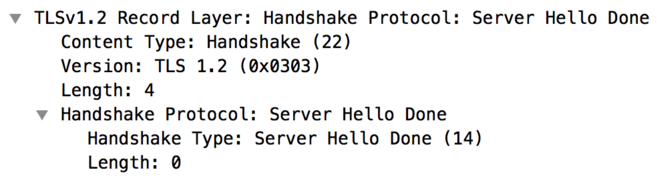


```
▼ Secure Sockets Layer
  ▶ TLSv1.2 Record Layer: Handshake Protocol: Certificate Status
  ▼ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 116
    ▼ Handshake Protocol: Server Key Exchange
        Handshake Type: Server Key Exchange (12)
        Length: 112
      ▼ EC Diffie-Hellman Server Params
          Curve Type: named_curve (0x03)
          Named Curve: x25519 (0x001d)
          Pubkey Length: 32
          Pubkey: cb0d8722d126f79e15013b593540d1eb018b37773f5cb96f...
        ▶ Signature Algorithm: ecdsa_secp384r1_sha384 (0x0503)
          Signature Length: 72
          Signature: 3046022100f6113bfade3e47d8c4cfb62327797a1359173d...
```

Figure 7: Client Key Exchange

3.4.1 Server Hello Done

Finally the server communicates to the client that it has done and awaits the client's response.



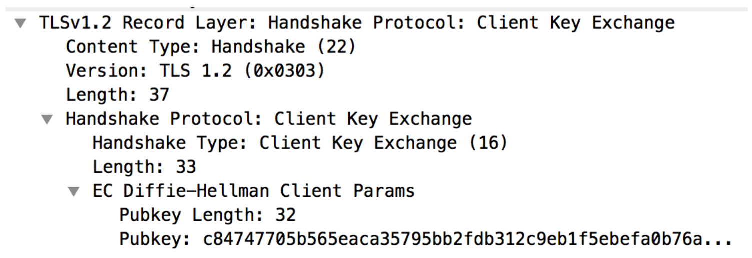
```
▼ TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 4
  ▼ Handshake Protocol: Server Hello Done
      Handshake Type: Server Hello Done (14)
      Length: 0
```

Figure 8: Server Hello Done

3.5 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message

3.5.1 Client Key Exchange

With this packet the client provides the Pre-master secret in the Client Key Exchange message. It' a random number generated by the client and encrypted with the server public key. This, together with the client and server random number, is used to create the master secret.

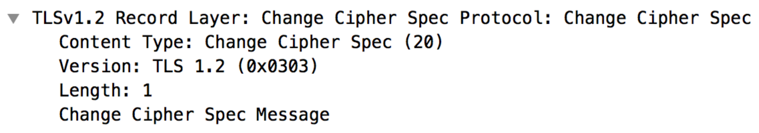


```
▼ TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 37
  ▼ Handshake Protocol: Client Key Exchange
      Handshake Type: Client Key Exchange (16)
      Length: 33
    ▼ EC Diffie-Hellman Client Params
        Pubkey Length: 32
        Pubkey: c84747705b565eaca35795bb2fdb312c9eb1f5ebefa0b76a...
```

Figure 9: Client Key Exchange

3.5.2 Client Change Cipher Spec

With the Change Cipher Spec message the client notifies the server that all the future message will be encrypted using the algorithm and keys that were just negotiated.

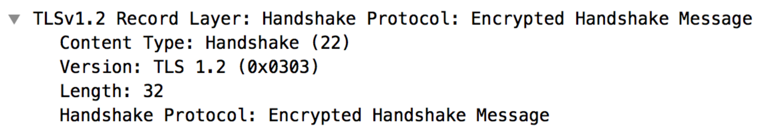


```
▼ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
    Content Type: Change Cipher Spec (20)
    Version: TLS 1.2 (0x0303)
    Length: 1
    Change Cipher Spec Message
```

Figure 10: Client Change Cipher Spec

3.5.3 Client Encrypted Handshake Message

Ultimately the Encrypted Handshake Message notifies the server that the TLS handshake is completed for the client and this message contains a hash of all the messages exchanged previously.



```
▼ TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 32
    Handshake Protocol: Encrypted Handshake Message
```

Figure 11: Client Encrypted Handshake Message

3.6 Change Cipher Spec, Encrypted Handshake Message

The final packet of the handshake protocol is used by the server to inform the client that the messages will be encrypted with the established algorithms and keys. The server also sends an Encrypted Handshake Message similar to the one of the client to inform him that the protocol has also ended for his side.
From this point onwords the two parties will communicate using a secure connection.

References

[1] Wireshark - network protocol analyser. https://www.wireshark.org/.

[2] Chacha, a variant of salsa20. https://cr.yp.to/chacha/chacha-20080128.pdfl.

[3] The poly1305-aes message-authentication code. http://cr.yp.to/mac/poly1305-20050329.pdfl.