Dr. Gaurav Varshney, IIT Jammu

TLS Cipher Suites

TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384

Let's break that down.

- TLS simply indicates the protocol;
- ECDHE signifies the key exchange algorithm;
- ECDSA signifies the authentication algorithm;
- AES_256_CBC indicates the bulk encryption algorithm; and
- SHA384 indicates the MAC algorithm.

TLS_CHACHA20_POLY1305_SHA256 (0x1303)	Forward Secrecy
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)	Forward Secrecy
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)	Forward Secrecy
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)	Forward Secrecy
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)	Forward Secrecy
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)	Forward Secrecy

https://www.jscape.com/blog/cipher-suites

Dr. Gaurav Varshney, IIT Jammu

RSA

problem is difficult. In true cryptographic fashion, the factoring problem on which RSA rests is hard because lots of smart people have looked at it, and apparently nobody has found an efficient solution.

To generate an RSA public and private key pair, choose two large prime numbers p and q and form their product $N = pq$. Next, choose e relatively prime to the product $(p-1)(q-1)$. Finally, find the multiplicative inverse of e modulo $(p-1)(q-1)$ and denote this inverse as d. At this point, we have N, which is the product of the two primes p and q, as well as e and d, which satisfy $ed = 1 \bmod (p-1)(q-1)$. Now forget the factors p and q.

The number N is the *modulus*, and e is the *encryption exponent* while d is the *decryption exponent*. The RSA key pair consists of

$$\text{Public key: } (N, e)$$

and

$$\text{Private key: } d.$$

In RSA, encryption and decryption are accomplished via modular exponentiation. To encrypt with RSA, we treat the plaintext message M as a number and raise it to the power e, modulo N, that is,

$$C = M^e \bmod N.$$

To decrypt C, modular exponentiation using the decryption exponent d does the trick, that is,

$$M = C^d \bmod N.$$

It's probably not obvious that RSA decryption actually works—we'll prove that it does shortly. Assume for a moment that RSA does work. Now, if Trudy can factor the modulus N (which is public) she will obtain p and q. Then she can use the other public value e to easily find the private value d since $ed = 1 \bmod (p-1)(q-1)$ and finding modular inverses is computationally easy. In other words, factoring the modulus enables Trudy to recover the private key, which breaks RSA. However, it is not known whether factoring is the only way to break RSA.

Does RSA really work? Given $C = M^e \bmod N$, we must show that

$$M = C^d \bmod N = M^{ed} \bmod N. \tag{4.2}$$

To do so, we need the following standard result from number theory [43]:

Euler's Theorem: If x is relatively prime to n then $x^{\phi(n)} = 1 \bmod n$

Recall that e and d were chosen so that

$$ed = 1 \bmod (p-1)(q-1).$$

Furthermore, $N = pq$, which implies

$$\phi(N) = (p-1)(q-1)$$

(see the Appendix if you are not familiar with the ϕ function). These two facts together imply that

$$ed - 1 = k\phi(N)$$

for some integer k. We don't need to know the precise value of k.

Now we have all of the necessary pieces of the puzzle to verify that RSA decryption works. Observe that

$$\begin{aligned} C^d &= M^{ed} = M^{(ed-1)+1} = M \cdot M^{ed-1} \\ &= M \cdot M^{k\phi(N)} = M \cdot 1^k = M \bmod N. \end{aligned} \tag{4.3}$$

In the first line of equation (4.3) we simply added zero to the exponent and in the second line we used Euler's Theorem to eliminate the ominous-looking $M^{\phi(N)}$ term. This confirms that the RSA decryption exponent does, in fact, decrypt the ciphertext C. Of course, the game was rigged since e and d were chosen so that Euler's Theorem would make everything come out as desired in the end. That's just the way mathematicians do things.

4.3.1 Textbook RSA Example

Let's consider a simple RSA example. To generate, say, Alice's keypair, we'll select the two "large" primes $p = 11$ and $q = 3$. Then the modulus is $N = pq = 33$ and $(p-1)(q-1) = 20$. Next, we choose the encryption exponent $e = 3$, which is, as required, relatively prime to $(p-1)(q-1)$. We then compute the corresponding decryption exponent, which in this case is $d = 7$, since $ed = 3 \cdot 7 = 1 \bmod 20$. Now, we have

$$\text{Alice's public key: } (N, e) = (33, 3)$$

and

$$\text{Alice's private key: } d = 7.$$

As usual, Alice's public key is public but only Alice has access to her private key.

Now suppose Bob wants to send Alice a message M. Further, suppose that as a number, the message is $M = 15$. Bob looks up Alice's public key $(N, e) = (33, 3)$ and computes the ciphertext as

$$C = M^e \bmod N = 15^3 = 3375 = 9 \bmod 33,$$

which he then sends to Alice.

TLS Handshake: RSA based SSL Handshake

Dr. Gaurav Varshney, IIT Jammu

Speeding up Secure Web Transactions Using Elliptic Curve Cryptography.
https://www.researchgate.net/publication/221655528_Speeding_up_Secure_Web_Transactions_Using_Elliptic_Curve_Cryptography

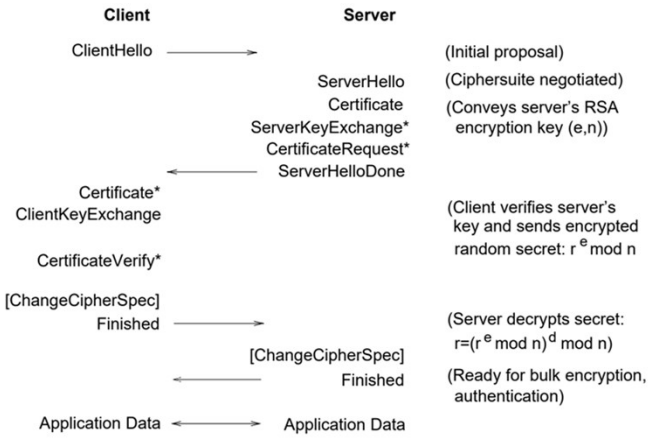


Figure 1. RSA-based SSL handshake.

TLS- ECDHE

Dr. Gaurav Varshney, IIT Jammu

Also known as: secp256r1 prime256v1

$$y^2 \equiv x^3 + ax + b$$

Parameters

Name	Value
p	0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff
a	0xffffffff00000001000000000000000000000000fffffffffffffffffffffffc
b	0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b
G	(0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296, 0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5)

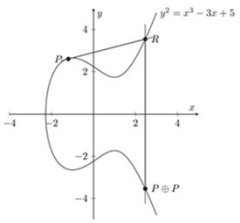


FIGURE 1. Adding a point to itself.

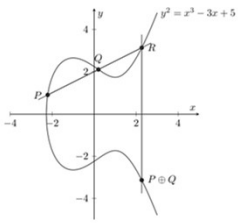


FIGURE 2. Adding two distinct points.

https://neuromancer.sk/std/nist/P-256

4.5.2 ECC Diffie-Hellman

Now that we can do addition on elliptic curves, let's consider the ECC version of the Diffie-Hellman key exchange. The public information consists of a curve and a point on the curve. We'll select the curve

$$y^2 = x^3 + 11x + b \pmod{167}, \tag{4.7}$$

leaving b to be determined momentarily. Next, we select any point (x, y) and determine b so that this point lies on the resulting curve. In this case, we'll choose, say, $(x, y) = (2, 7)$. Then substituting $x = 2$ and $y = 7$ into equation (4.7) we find $b = 19$. The public information is

$$\text{Public: } y^2 = x^3 + 11x + 19 \pmod{167} \text{ and the point } (2, 7). \tag{4.8}$$

Alice and Bob each must randomly select their own secret multipliers.[8] Suppose Alice selects $A = 15$ and Bob selects $B = 22$. Then Alice computes

$$A(2, 7) = 15(2, 7) = (102, 88),$$

where all arithmetic is done on the curve in equation (4.8). Alice sends her computed result to Bob. Bob computes

$$B(2, 7) = 22(2, 7) = (9, 43),$$

which he sends to Alice. Now Alice multiplies the value she received from Bob by her secret multiplier A, that is,

$$A(9, 43) = 15(9, 43) = (131, 140).$$

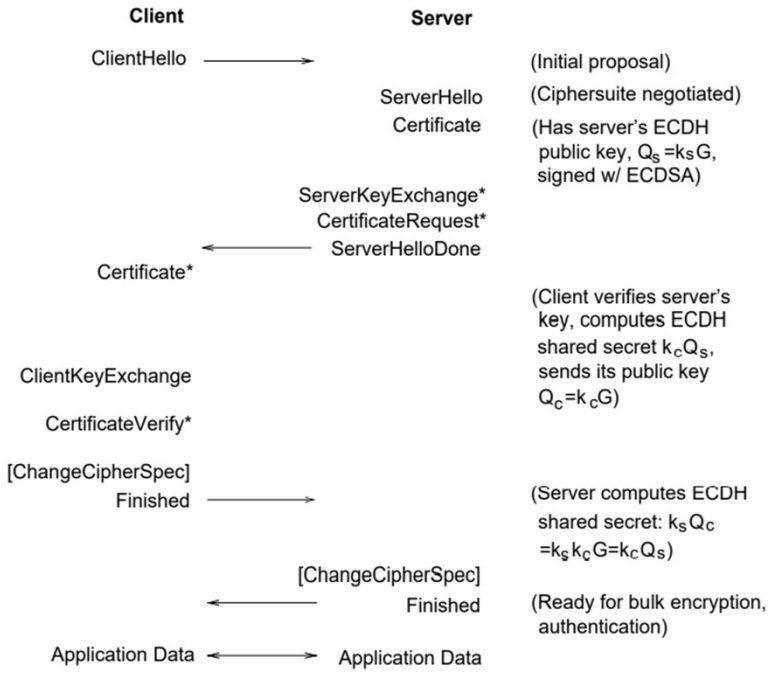
Similarly, Bob computes

$$B(102, 88) = 22(102, 88) = (131, 140)$$

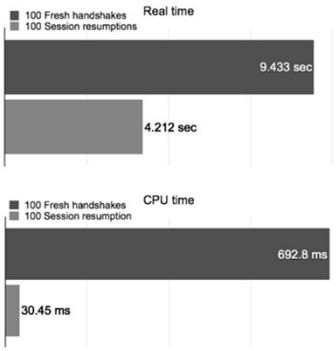
and Alice and Bob have established a shared secret, suitable for use as a symmetric key. Note that this elliptic curve version of Diffie-Hellman works

https://math.uchicago.edu/~may/REU2020/REUPapers/Shevchuk.pdf

TLS Handshake: ECC based SSL Handshake



TLS Handshake: Session Resumption: Using Session IDS



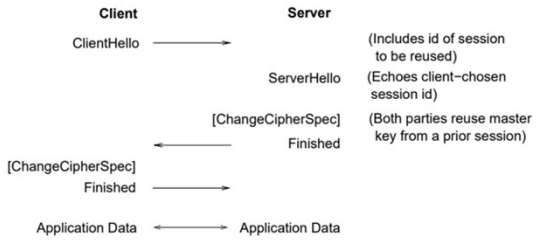


Figure 3. Abbreviated SSL handshake.

Session Identifiers

In RFC 5246 TLS 1.2 Appendix-F.1.4, states:

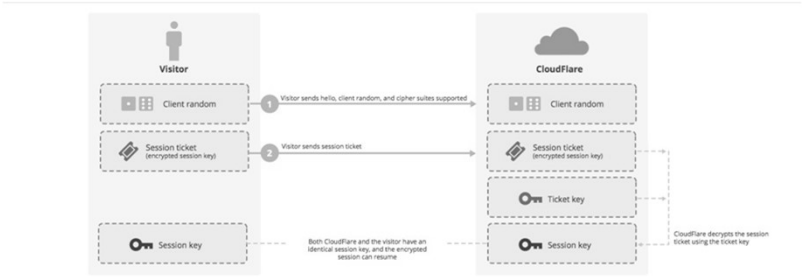
*When a connection is established by resuming a session, **new** ClientHello.random and ServerHello.random values are hashed with the session's Master Secret. Provided that the Master Secret has not been compromised and that the secure hash operations used to produce the encryption keys and MAC keys are secure, the connection should be secure and effectively independent from previous connections.*

Attackers cannot use known encryption keys or MAC secrets to compromise the Master Secret without breaking the secure hash operations.

https://blog.cloudflare.com/tls-session-resumption-full-speed-and-secure/

TLS Handshake: Session Resumption: Using Session Tickets

Dr. Gaurav Varshney, IIT Jammu



The idea is simple: outsource session storage to clients. A session ticket is a blob of a session key and associated information encrypted by a key which is only known by the server. The ticket is sent by the server at the end of the TLS handshake. Clients supporting session tickets will cache the ticket along with the current session key information. Later the client includes the session ticket in the handshake message to indicate it wishes to resume the earlier session The server on the other end will be able to decrypt this ticket, recover the session key and resume the session.