Advantage of using Elliptic curve cryptography in SSL/TLS

Benjamin Clement Sebastian(benjaminclementsebastian@umail.ucsb.edu)
& Ugur Alpay Cenar(ucenar@umail.ucsb.edu)

***Abstract*—Mobile and wireless devices is experiencing an explosive growth. These devices have strict Power, CPU power, memory, bandwidth and latency constraints, which makes it important to have an efficient cryptosystem. It is also important for the web performance in general to have an efficient cryptosystem. Today is RSA an widely used public-key cryptosystem.The problem with RSA is that it requires large key sizes which can lead to lower bandwidth and higher CPU usage. Elliptic Curve Cryptography (ECC) is emerging as an attractive public-key cryptosystem. It offers equivalent security level with smaller key sizes, which can lead to faster computation and lower power usage.**
SSL/TLS is the most widely used security protocol on the web, so more efficient SSL/TLS will have a significant impact on the web performance. This paper compares those two cryptosystems to see if ECC gives significant advantage over RSA regarding to performance when implemented in SSL/TLS protocol.

I. Introduction

The use of Internet has grown exponentially the last decade and it will continue to grow. With the rapid deployment of online applications like online banking and stock trading there is a need for not only to have the most secure transmission of data, but also the fastest available. These days it has been a growing trend to use mobile and wireless platform as Internet hosts, which has battery, power and CPU limitations. Today the most common protocol for secure transmission of data through the web is SSL/TLS, and is using public key cryptography to derive symmetric keys and then use symmetric key cryptography to ensure confidentiality. RSA is the most commonly used public key cryptosystem used in SSL/TLS. As cryptoanalysis is getting better and more advanced, it requires that both symmetric and public key size to grow in size. The bigger the key size, the more computer resources must be used. Since 2011, NIST has recommended to use key-size of 2048 bit for RSA algorithm and recommends to increase the key-size to 3072 bit after 2030 for maximum security. Elliptic Curve Cryptography(ECC) is emerging as an attractive alternative to RSA. ECC offers equivalent security with smaller key sizes resulting in faster computations and lower power consumption. This paper describes and compares those two cryptosystems implemented in SSL/TLS on the web.

II. SSL/TLS overview

Secure Sockets Layer (SSL) and it successor Transport Layer Security (TLS) are security protocols that enables privacy between two communicating applications. This is protocol is mainly used in HTTP, but are also utilized for other protocols like FTP and SMTP. ECC was first included in this protocol with release of TLS 1.0. In rest of this paper the phrase SSL are going to be used for both TLS and SSL, because both are frequently referred to as SSL in public.
Two communicating applications needs to use the same master key for encryption and decryption. This can be agreed by SSL handshake.
The Handshake protocol goes like:

Algorithm A: SSL handshake

1. **Client:** ClientHello message to the server. [1]
2. **Server:** Selects cipher suite [2], which can be (ECDH-ECDSA-AES256-SHA) [3] and sends it back as ServerHello message.
3. **Server:** Sends ServerCertificate message which contains ECDH public-key signed with ECDSA to protect against Man-In-The-Middle attack.
4. **Client:** Validates the signature and sends back ClientKeyExchange message that contains client ECDH public-key
5. Both use their own private key to arrive at shared master secret and all further messages will be encrypted.

The SSL handshake can be computational expensive in cases where a mobile client is present. In some cases the same master secret can be used in several sessions. This can avoid unnecessary handshakes. The RSA cryptosystem is the most widely used public key cryptography algorithm in the world. In SSL the RSA algorithm is used in the handshake between a server and the client and is used to exchange the private key, for confidentiality, and digital signatures, for authentication.

III. Overview RSA

RSA derives its security from the difficulty of integer factorization of large integers which are the product of two large prime numbers. The key generation is as follows, the two prime numbers are first generated using Rabin-Miller primality test algorithm. The product of those two primes is the modulus n, and is used by both the public and private key.The totient $\phi(n) = (p-1)(q-1)$ is computed. A number e is selected such that $1 < e < \phi(n)$. n and e comprise the public key. The

[1]Includes: client supported SSL protocol version and supported cipher suites.

[2]Both client and server have to support the cipher suite

[3]Cipher suite tells what kind of encryption protocol are going to be used. In this case ECDH are used for key-exchange, ECDSA for signature, AES256 for bulk-encryption and SHA for message authentication

private key d is computed such that $d * e \ mod \ \phi(n) = 1$, and by using the extended Euclidean algorithm d can be found. So after the symmetric key has been generated it can be encrypted using the RSA public key and transmitted over to the recipient and decrypted with the corresponding private key. So now both parties have the private key. The other use of RSA in the SSL handshake is digital signatures and it works as follows. The sender creates a hash of the message, encrypt it with the RSA private key of the sender. Then the recipient can verify that the message has not been altered with by decrypting the hash value with the sender's public key. If the value matched the hash of the original message, then it has not been altered with.

IV. Elliptic curve cryptography (ECC) background

The strength of every public-key cryptosystem is that it is an easy mathematical problem if you have the private key. But to crack the system the cracker needs to solve the discrete logarithm problem which can be very difficult mathematical problem to solve. The difficulty of the problem, which determines the security of the cryptosystem, is determined by the key size. The problem is that larger key size means more computation, which makes the system slower and the system needs to use more power in total. The advantage of ECC is that it needs smaller key-size to get equivalent security compared to traditional cryptosystems like RSA. Cryptosystems like RSA can be attacked in sub-exponential time while it takes exponential time to attack ECC. In the table below you can see the key-sizes that gives equivalent security.

TABLE I
COMPARABLE KEY SIZES

Symmetric (bits)	ECC (bits)	RSA(bits)
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

Elliptic curve cryptography(ECC) was suggested independently by Neal Koblitz and Victor S. Miller in 1985. ECC has been widely used cryptographic method since 2004. ECC operates over group of points on a elliptic curve, unlike RSA that operates over integer fields. Elliptic curve is a plane curve over a finite field, which have points that is satisfying the equation $y^2 + a_1xy + a_3y = x^3 + a_2x + a_6$ including the point at ∞.
What makes ECC secure is the Elliptic curve discrete logarithm problem (ECDLP). If you have given points P and $Q = [k]P$, then it is hard to find the integer k. From the user side, k can be chosen randomly, and $Q = [k]P$ can be calculated using scalar point multiplication, which can be performed by using combination of point-addition and point-multiplication. But calculating k from $Q = [k]P$ is infeasible, which makes ECC secure.

V. Elliptic curve diffie-hellman (ECDH)

In SSL the ECC curve $y^2 = x^3 + ax + b$ is usually used. This curve is of the simplified Weierstrass form.
To use ECC in cryptography, both sides of the communication have to agree on public domain parameters. Domain parameters for an elliptic curve, describes elliptic curve E defined over a finite field $\mathbb{F}_q$ with base point $P \in E(\mathbb{F}_q)$ with order n. The domain parameters consists of [4]

- Finite field $\mathbb{F}_q$ where q is the field order (usually prime number).
- Two coefficients $a, b \in \mathbb{F}_q$ that defines the equation for the elliptic curve E over $\mathbb{F}_q$.
- A point $P = (x, y) \in E(\mathbb{F}_q)$ with the order of n such that $nP = 0$,this point is also called the base point
- the cofactor $h = \frac{1}{n}E(\mathbb{F}_q)$. Where it is important that n is large and h is small to be secure against Pohlig-hellman and Pollar-rho attacks.

These combined makes the public domain parameters (q, a, b, P, n, h).

Elliptic curve Diffie Hellman key exchange (ECDH) and Elliptic Curve Digital Signature Algorithm (ECDSA) is two important algorithms used in SSL.
To make an Elliptic curve Diffie Hellman key exchange (ECDH), both parts (Alice and Bob) must use same public domain parameters (q, a, b, P, n, h) that is agreed upon.
Algorithm B: ECDH

1 Alice and Bob agree on domain parameters with elliptic curve group ε of order q and a primitive element $P \in \varepsilon$ which is public
2 Alice selects integer $k_a \in [2, q-1]$ and computes $Q = [k_a]P$ and sends it to Bob
3 Bob selects integer $k_b \in [2, q-1]$ and computes $R = [k_b]P$ and sends it to Alice
4 Alice receives R and computes $S = [k_a]R$
5 Bob receives Q and computes $S = [k_b]Q$

- Then: $S = [k_a]R = [k_a][k_b]P = [k_b]Q = [k_b][k_a]P = [k_a * k_b \ mod \ q]P$

Illustration of the algorithm can be seen on figure 1.
The private key is held private and only the owner of the private key knows the private key. So no other that Alice can determine Alice's private key, unless someone can solve ECDLP. And no one but Alice and Bob knows the shared secret key, unless someone can solve Elliptic curve Diffie-hellmann problem.

VI. Elliptic curve digital signature algorithm (ECDSA)

For a SSL connection to be in place, some crucial things needs to be in order. Among them it's needed to verify that the sender is who it says it is. A digital signature allows the recipient to verify the message's authenticity. The ECDSA works as follows. Alice wants to sign a message with her private key, d_A, and Bob wants to validate the signature using Alice's public key, Q. Nobody but Alice should be able to

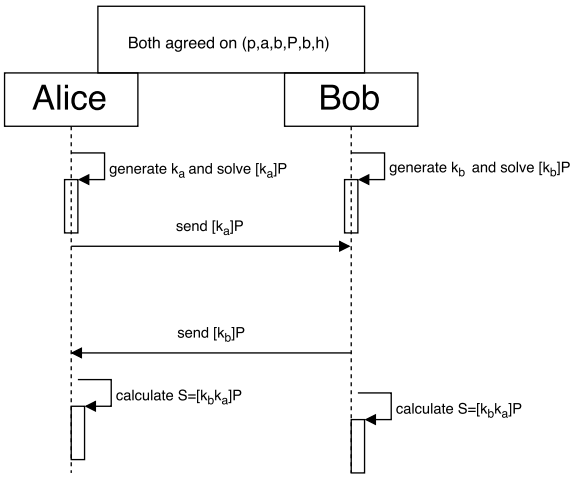

Fig. 1. Illustration of how ECDH works (S is the shared master secret)

produce valid signatures. They're using the same domain parameters. First, the message is hashed, converted to a much smaller and fixed length message, using a chosen hashing algorithm, for example Secure-Hash-Algorithm-2 (SHA-2). A secure hash has a number of properties that makes it secure, for example 1) irreversibility, nearly impossible to convert it to the original message, 2) collision resistance, chances are very small to find another message which have the same hash value as the original message, and 3) any change in message will produce a significant change in hashing value. The hashing value is denoted h(m). The algorithm works as followed [5]:

Algorithm C: ECDSA signing
Input: $\varepsilon(a, b, p)$ with order n and the primitive element $P \in \varepsilon$ with order n
Private key is random integer $d \in [2, n-2]$
Public key is a point on the curve $Q = [n]P$
Output: Signature of the message m: (s_1, s_2)

1: Generate random integer $r \in [2, n-2]$
2: Compute $[r]P = (x_1, y_1)$
3: Compute integer $s_1 = x_1 \ (mod\ n)$
4: If $s_1 = 0$, stop and go back to step 1
5: Compute $r^{-1} \ (mod\ n)$
6: Compute $s_2 = r^{-1}(h(m) + ds_1) \ (mod\ n)$
7: If $s_2 = 0$, stop and go to step 1
8: The signature on the message m is the pair of integers (s_1, s_2)

Algorithm D: ECDSA Verification
Input:the message m, and the signature (s_1, s_2)
Verifier knows the domain parameters and the public key Q
Output: Message is valid or not valid.

1: Compute $w = s_2^{-1} \ (mod\ n)$
2: Compute $u_1 = h(m)w \ (mod\ n)$
3: Compute $u_2 = s_1 w \ (mod\ n)$
4: Compute $[u_1]P \oplus [u_2]Q = (x_2, y_2)$
5: Compute integer $v = x_2 \ (mod\ n)$
6: The signature is valid if $v = s_1$

VII. Comparing Performance

To check if ECC gives significant performance improvement over RSA in SSL protocol, both have to be tested in realistic situation. There is couple of articles that have tested both cryptosystems in SSL and compared the performance to see what kind of improvement ECC gives over RSA. This paper gives short summary of the performance test made by Sun Microsystems [2] and Symantec [3].

A. Test methodology

The experiment from Sun Microsystems [2] was tested on an Apache 2.0.45 web server using OpenSSL. The server was an computer with 900MHz UltraSparc 3 processor with 2GB ram and the client was running on 7 900MHz UltraSparc 3 processors with 14 GB ram. Server and client was connected via 100Mb ethernet network.
In this test, two different cipher suites was used: RSA-RC4-SHA and ECDH-ECDSA-RC4-SHA. where each cipher suite was used with key-sizes 1024 bits and 2048 bits for RSA, 160 bits and 224 bits for ECC. From table I it is known that 1024 bits RSA gives same security as 160 bits ECC and 2048 bits RSA gives same security as 224 bits ECC.

B. Desktop as client

One of the tests run by Sun MicroSystem compared the First Response Time (FRT) as a function of Request Per Second (RPS). FRT is the delay between initial SSL handshake and when the client receives its first packet. This is experienced as the latency between the user requests for a website and seeing the first update on the browser window. Request per second is how many page requests the server gets at the same time.
This was tested with 30KB of page size with 60% session reuse which means new session for every 3 fetches. The results of the test can be seen on the figure 3. From figure 3, it can be seen that with small key sizes there is not much difference on the FRT between RSA and ECC, and the difference increase for larger key-sizes. It can also be seen that the server can handle up to 270% more requests with ECC compared to RSA.

Another test from the paper [2] was comparing average time taken by the server to fulfill HTTP fetch with public key with no session reuse. This was tested with file sizes 0KB, 10KB, 30KB and 70KB. The figure shows microbenchmark results for ECC,RSA,RC4 and SHA. From the figure 2 it can be seen that RSA use most time for every file type and this time increases exponentially with larger key-sizes. ECC reduces total processing time by 29% to 86% compared with RSA.

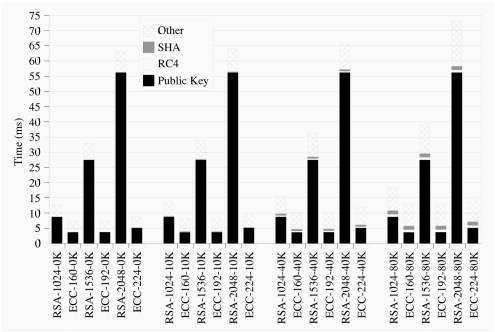

Fig. 2. Relative cost in HTTP transaction (Figure 4 in paper [2])

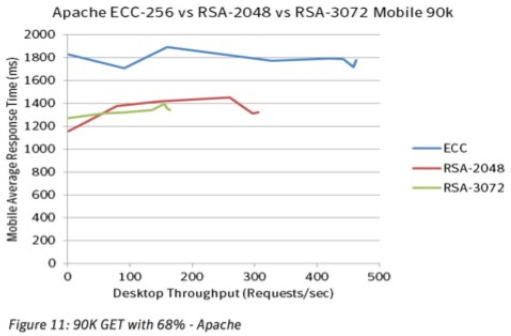

Fig. 4. Mobile response time with 90k data(lower is better) (Figure 11 in paper [3])

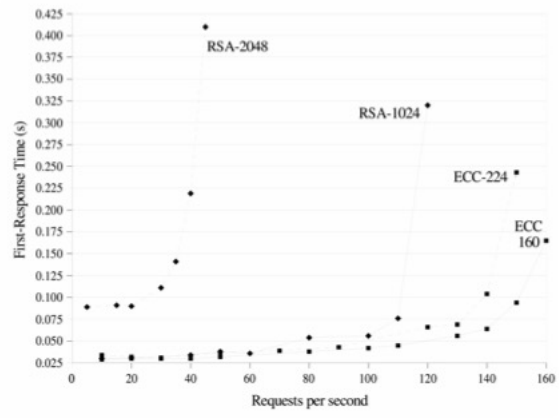

Fig. 3. Latency v/s Throughput plot for Apache web server. (Figure 5 in paper [2])

C. Mobile device as client

Symantec [3] made an similar test, but they included a test on an mobile device which was interesting. This was tested on AT&T LTE network with an android phone(Samsung galaxy s3). The cipher suites compared was ECDHE-ECDSA-AES256-SHA and ECDHE-RSA-AES256-SHA which means that both are using ECDHE as key-exchange algorithm but using different authentication algorithms. The key sizes compared was 256 bit ECC, 2048 bit RSA and 3072 bit RSA. An Apache server using OpenSSL was used in the test. The test compared Average response time as function of Request per second. It was done with 90K page size with 68% session reuse.

They ran the test several times and the result can be seen on figure 4. What is interesting is that RSA beats ECC on performance on this test for both 2048 bit and 3072 bit key-size. Symantec concludes that the public-key cryptography operations are more computationally expensive on the client using ECC-256 bit. The difference in response time is not significantly large, and the mobile hardware performance is rapidly increasing which will soon give the same performance benefits using ECC on mobile client as using it on Desktop client.

From the figure 4 it can also be seen that the server can handle more requests per second with ECC-256 compared to RSA, which is an interesting result when thinking that the amount of mobile clients are increasing which will require to handle more requests per second.

VIII. Conclusion

From the analysis above, it can be seen that ECC cipher suites give significant performance benefits over RSA used in SSL with desktop as client. But with mobile client, the ECC cipher suite gives poorer response time compared to RSA. Symantec concludes that this happens because of the hardware restrictions on the mobile client, which will not be a problem when the hardware gets better.

Today, users are more sensitive on their on-line privacy and the demand of using SSL protection for web-transactions are increasing. This trend will give broader implementation of ECC on web-transactions both on mobile and desktop clients due to the performance benefits. Popular websites like Facebook, Google, Youtube are now using ECC for encrypting transactions and more and more websites are using ECC because of its performance benefits.

References

[1] Elliptic curve Diffie–Hellman, Wikipedia https://en.wikipedia.org/wiki/Elliptic_curve_Diffie\OT1\ textendashHellman,
[2] Speeding up Secure Web Transactions using Elliptic Curve Cryptography (ECC) by SunMicrosystems, http://files.douglas.stebila.ca/files/research/papers/GSFCGE04.pdf,
[3] Elliptic Curve Cryptography (ECC) Certificates Performance Analysis by Symantec https://www.symantec.com/content/en/us/enterprise/white_papers/b-wp_ecc.pdf
[4] D. Hankerson, S. Vanstone, and A. Menezes. Guide to Elliptic Curve Cryptography, page 172, Springer 2004
[5] Slide: $13eccalgs.pdf$
[6] Slide: $09ecc.pdf$