WHITE PAPER:
ELLIPTIC CURVE CRYPTOGRAPHY (ECC)
CERTIFICATES PERFORMANCE ANALYSIS

White Paper

Elliptic Curve Cryptography (ECC) Certificates Performance Analysis

Symantec™

Norton™ SECURED
powered by Symantec

Elliptic Curve Cryptography (ECC) Certificates Performance Analysis

CONTENTS

Authors . . . 3

Introduction . . . 3

- Purpose . . . 3
- Audience . . . 3

Why ECC Certificates? . . . 3

- Algorithm Agility . . . 3
- Security . . . 4
- Scalability and Performance . . . 4
- Compliance, Guidelines . . . 4

SSL/TLS Overview . . . 4

- SSL/TLS Basics . . . 4
- SSL Cipher Suites . . . 5
- How to Choose the Cipher Suites for your Web Server . . . 6
- Session Reuse and Why Does it Matter . . . 6
- SSL Handshake Explained . . . 7
- Security Bits/Security Strength Equivalence . . . 12

Test Methodology . . . 12

- Client Applications . . . 13
- Web Servers . . . 13
- Amazon EC2 Environment, AMIs and Instance Types . . . 14
- Test Strategy . . . 15

Tests and Analysis . . . 16

- Server Performance Metrics (SEQUENTIAL) . . . 16
- Client: Desktop Response Time Versus Throughput Metrics (DRVT) . . . 17
- Client: Mobile Response Time and Throughput Metrics (MRTT) . . . 21

ECC Ubiquity . . . 23

References . . . 25

Appendix . . . 25

- Figures . . . 25
- Tables . . . 25

Authors
Ajay Kumar, Antony Jerome, Gaurav Khanna, Hari Veladanda, Hoa Ly, Ning Chai, Rick Andrews - May 2013

Introduction

Purpose
The purpose of this exercise is to provide useful documentation on Elliptic Curve Cryptography (ECC) based SSL/TLS certificates with an emphasis on comparison with the ubiquitous RSA based certificates. The primary driver of this exercise and the presentation of the accumulated data coincides with Symantec being the first CA to support certificates based on ECC algorithms.

Audience
The audience of this document is the CIO, CISO, Administrator - either Web Server, Network or IT, any actor responsible for enabling security and administering security applications and those with stakes in performance, scalability and capacity planning.

The audience is by no means, restricted. We try to provide an in depth appraisal of ECC versus RSA and its implications on hardware resources along with conceptual and empirical study of these two algorithms usage in SSL/TLS.

Why ECC Certificates?
Symantec is the first commercial Certificate Authority to sell certificates based on ECC (Elliptical Curve Cryptography) algorithms. This next-generation algorithm provides stronger security and better server utilization than current standard encryption methods, but requires shorter key lengths. The result is increased protection and a better customer experience.

Many other players in the IT security technology space are looking at ECC, and starting to build it into their future development cycles. While at the time of this publication RSA is still the standard for SSL/TLS encryption, given the changes in root availability, guideline directives to change key sizes, and the improved performance, it is clear that ECC is going to be a major factor in securing the security ecosystem for years to come.

Algorithm Agility
Algorithm Agility is the practice of having multiple certificates available for installation. Especially given the new guidelines to migrate from 1024-bit keys to 2048-bit keys as of 1/1/2014[1], businesses need have the ability to choose the right algorithm options that fit their needs. In parallel with the adjustment to the minimum key size by NIST, the US Government has issued and adopted guidelines for alternative algorithms for encryption and signing adding Elliptic Curve Cryptography (ECC) and Digital Signature Algorithms (DSA)[2].

[1] Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths
[2] FIPS: Digital Signature Standard (DSS)

Any organization should be able to choose between certificates that provide protection based on the algorithm that suits their environment: RSA, ECC, or DSA. This agility allows business owners to provide a broader array of encryption options for different circumstances, infrastructure, and customer or partner groups.

The design of Transport Layer Security (TLS – formerly Secure Sockets Layer or SSL) allows different algorithms to work either alone or side by side. With algorithmic agility, organizations can choose the public key algorithm, or combination of algorithms, that work best for their online presence and infrastructure.

Security

While key lengths for current encryption methods using RSA increase exponentially as security levels increase, ECC key lengths increase linearly. For example, 128-bit security requires a 3,072-bit RSA key, but only a 256-bit ECC key. Increasing to 256-bit security requires a 15,360- bit RSA key, but only a 512-bit ECC key[3]. The previously mentioned NIST guidelines stay abreast with the need for increasing security. With such a favorable security per bit ratio, it is anticipated that ECC will be the focus of planning for IT and their supplemental Security systems.

Scalability and Performance

Websites using ECC need fewer server processing cycles, allowing for more simultaneous SSL/TLS connections and faster page loading.

Compliance, Guidelines

The ECC algorithm is endorsed by the NSA (National Security Agency), and is compliant with the NIST 800-131A guidelines[4]. The curves used by Symantec for ECC certificates are among those listed as acceptable for NSA Suite B[5].

SSL/TLS Overview

SSL/TLS Basics

SSL and its successor, TLS, are security protocols that enable privacy, data integrity[6] between two communicating applications. For the remainder of this document, we would use the terms SSL and TLS interchangeably unless explicitly differentiated.

The SSL ecosystem was primarily developed for HTTP applications but that does not preclude it and it is utilized for other protocols as well such as FTP, SMTP etc. The SSL ecosystem along with the two parties comprises the Certification Authorities such as Symantec, SSL bindings for the particular protocol such as HTTPS or FTPS, other ancillary protocols such as OCSP (Online Certificate Status Protocol)[7] and CRLs to name a few. All the major browser clients, other mobile or desktop clients and web servers implement the SSL protocol. We have utilized two of the most popular web servers: Apache Web Server[8] and Internet Information Services[9] ; for analyzing SSL made possible by RSA and ECC based certificates. There is an abundance of resources that aid in understanding the protocol in

[3] The ca se for Elliptic Curve Cryptography
[4] Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths
[5] NSA Suite B Cryptography
[6] TLS 1.2, TLS 1.1, TLS 1.0
[7] Online Certificate Status Protocol
[8] Apache Web Server
[9] IIS

any amount of depth and some of the material is referenced at the end as well. However we will delineate certain facets of the SSL protocol relevant to the exercise such as:

1. We begin by describing cipher suites: what are they and the role they play.
2. Session Reuse and why does it matter?
3. The SSL Handshake with an emphasis on enabling forward secrecy[10],

Utmost care has been taken to ensure the accuracy and veracity of the contents of this document. If you have any questions on these aspects or feel the information is incorrect, please contact us at hari_veladanda@symantec.com

SSL Cipher Suites

During a SSL handshake (covered in the next section), the client sends a set of cipher suites it supports. The server will select a preferred cipher suite from the list for the subsequent steps.

Cipher Suites are represented as 2 byte constants and specify the server authentication algorithm, the key exchange algorithm, the bulk encryption algorithm and the digest (message integrity) algorithm[11].

For instance, for illustrative purposes only:

0xC0,0x0A - ECDHE-ECDSA-AES256-SHA SSLv3 Kx=ECDH Au=ECDSA Enc=AES(256) Mac=SHA1

In the above example:

1. the 2 byte identifier is “0xC0,0x0A”,
2. The server authentication algorithm is “ECDSA” (Elliptic Curve DSA),
3. The key exchange algorithm is ephemeral “ECDH” (Ephemeral Elliptic Curve DH)
4. The bulk encryption algorithm is “AES”
5. The MAC is “SHA1”

The cipher suite selected by the server during the SSL handshake depends on the type of web server certificate, RSA or ECC, the client SSL protocol version, and the cryptographic algorithms support by the both sides. A selection of a cipher suite has a profound impact on server performance numbers and has particular security implications as well.

[10] SSL/TLS and Perfect Forward Secrecy by Vincent Bernat
[11] Refer to Eric Rescorla’s “SSL and TLS”

How to Choose the Cipher Suites for your Web Server

While specifying a list of cipher suites for your web server, it is recommended that data is collected to answer the following questions:

1. What are the types of clients that connect to your server? For example: types of browsers and their versions, desktop and mobile devices etc.
2. What is the SSL/TLS version that is commonly supported by all the myriad clients and is it secure?
3. The security level of data – how confidential is it; will it need to be protected years down the line? For instance, certain transactions need to be secured and kept secured for an indefinite period of time whereas others do not.
4. Are the recommendations and outlines as published as in NSA Suite B and/or are the best practices for selecting a cipher suite for the web server being followed?
5. Are there any known vulnerabilities for a particular suite and are they patched? If the vulnerability arises on a client then the mitigation requires either disabling support for that client on the server or choosing another suite.
6. Are the algorithms in the cipher suite supported by your server and the clients?

In this section the terms “client” and “web browser” are interchangeable.

As of writing, TLS protocol version 1.0 is ubiquitous in support among the various browsers[12]. Although there are reported vulnerabilities in this version but the newer versions such as TLS v1.1 and TLS v1.2 are not widely supported. Proper configuration is required to eliminate these vulnerabilities and needless to say that is paramount in achieving the aims of SSL.

We recommend that the cipher suite that is chosen by your organization has the characteristics elucidated in this section and we reiterate that utmost care has to be taken to ensure that the server software is patched as per all known vulnerabilities. For instance, if there is vulnerability in OpenSSL (that powers Apache and Nginx SSL/TLS) that comes to the fore then it has to be acted upon and patched.

Session Reuse and Why Does it Matter

The complete SSL Handshake process can be very expensive especially in cases of mobile clients with comparatively lower hardware specifications as compared to that of a desktop. Session resumption allows savings in CPU and network roundtrips to secure a connection based on a “master secret” that has been agreed upon in a prior handshake. This results in an abbreviated handshake with fewer round trips between the client and the server and elimination of the CPU cycles that are required for the public key cryptography steps required to generate the “master secret” at either end.

[12] TLS Survey by Tom Ritter

To enable session resumption, the server such as an Apache Web Server, can be configured to host the session information per client or the client can cache the same. The latter approach is explained in RFC 5077[13]. Older clients require that the server cache the session information[14].

Session resumption benefits are especially conspicuous on high latency and lower hardware specifications such as mobile devices.

SSL Handshake Explained

This section describes briefly what is involved at each steps of a successful SSL handshake at a very high level; note that client authentication is omitted.

The SSL Handshake

In this section we will elucidate the SSL Handshake protocol in context of Forward Secrecy.

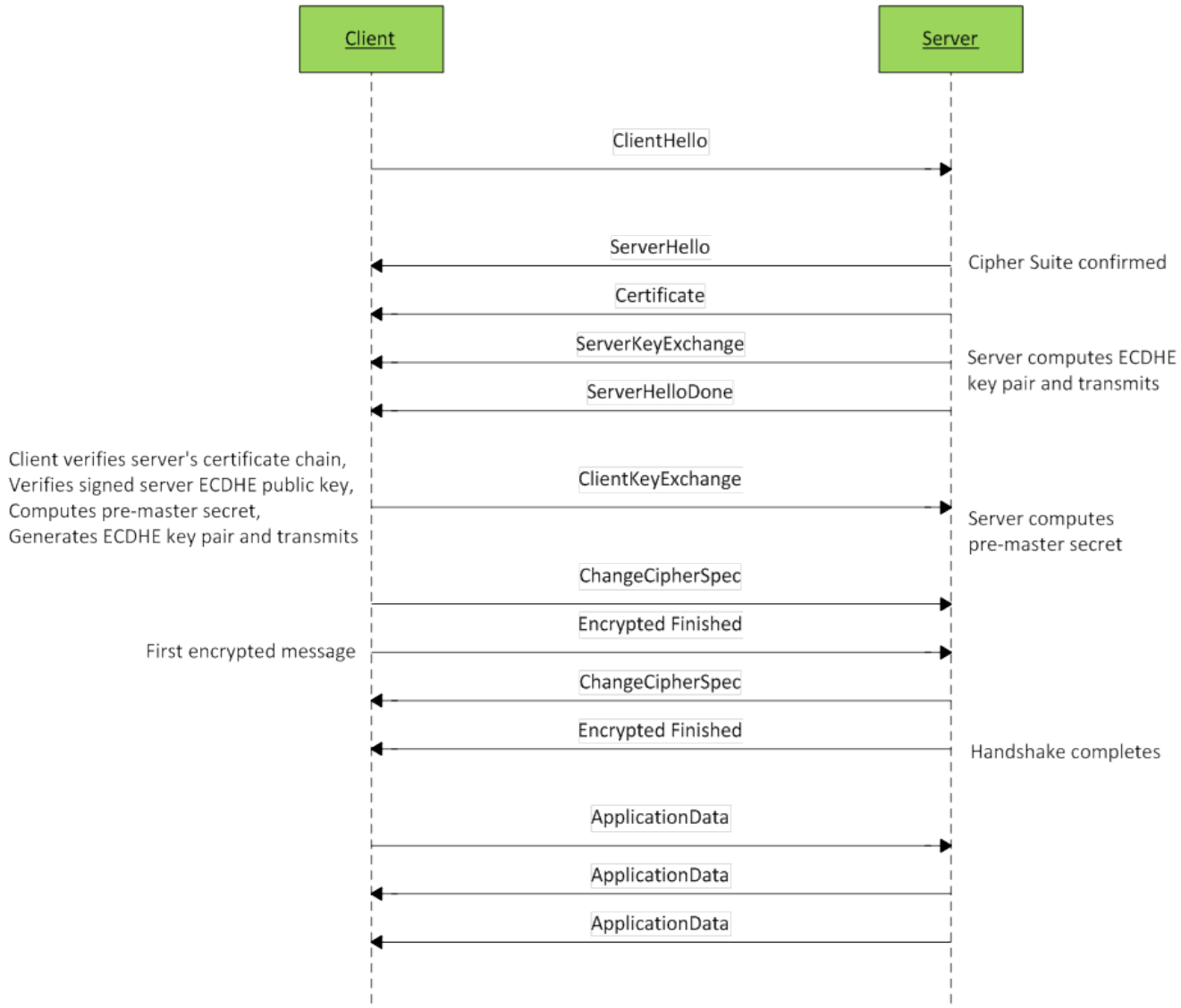

Figure 1: Ephemeral Handshake

[13] TLS Session Resumption without Server-Side State
[14] Overclocking SSL

Steps:

1. To initiate a SSL session, the client sends a **ClientHello** message to the server. The message consists of the client supported SSL protocol version, a set of cipher suites it supports in descending preferred order, a stream of random bytes, a set of extensions etc. One of the extensions is the elliptic_curve extension[15], which specifies a list of Elliptic curves the client can support. Another extension, ec_point_formats, specifies if the client can support point compression format.
2. Upon receiving the ClientHello request, the server selects a cipher suite it supports from among the set sent by the client. For the purpose of this exercise, the server when configured with a RSA certificate or an ECC certificate and with a digitalSignature flag in its Key Usage extension, then the selected cipher suite was ECDHE-RSA-WITH-AES256-SHA or ECDHE-ECDSA-AES256-SHA.

   The server response consists of multiple SSL messages:
   The server sends a **ServerHello** message, which consists of the server supported SSL version, the selected cipher suite, a random value generated by the server, and a list of extensions. If the selected cipher suite is for ECC, then extensions will contain a list of supported ec-point-formats. Both the client and the server generated random numbers are saved later for master secret generation.

   This is followed by a **Certificate** message, which the server uses to convey its server certificate chain to the client. The certificate chain selected by the server reflects the server authentication portion of the chosen cipher suite. This chain is used by the client for the server authentication purposes.

   Since the server chooses ECDHE as the key exchange algorithm, it will generate an ephemeral ECDHE key pair based on a server selected EC domain parameter. To guard against man-in-the-middle attack, the ephemeral ECDH public key and the domain parameters are signed by the server's private key (which corresponds to the public key in the server certificate). The ephemeral ECDH public key, domain parameters, and the signature block are sent to the client through the **ServerKeyExchange** message.

   To conclude the cipher negotiation step, the server sends a **ServerHelloDone**.
3. The session cipher suite is established once the client receives the ServerHello message. The client validates the server certificate, based on the certificate chain it received from the server Certificate message, chaining up to a trusted root certificate. To guard against man-in-the-middle attacks, a SSL client application should consider the following as a part of the certificate chain validation: a) Certificate status at each certificate level in the hierarchy must be checked against CRL or OCSP to ensure that the target certificate has not

[15] ECC Cipher Suites for TLS

been revoked; b) the certificate's validity period should also be checked to make sure that none of the certificates has expired; and c) the web server domain name in the server certificate, certificate extensions, etc should be validated as well. Note that the domain name in the server certificate may contain a wildcard character, which must be taken into account when validating the certificate.

Once the server certificate is validated, the client uses the public key in the server certificate to validate the signature in the ServerKeyExchange message. If the signature is validated, the client generates an ECDH ephemeral key pair from the EC domain parameter sent in the ServerKeyExchange message. The client ECDH ephemeral public key is sent to the server in a **ClientKeyExchange**.

To complete the ClientKeyExchange, the client performs an ECDH operation to generate the premaster secret. The premaster secret, client and server generated random bytes, and an identifier label will be used by both the client and the server to generate the same master secret independently. The master secret is then used to generate all cryptography keys: bulk data encryption keys, MAC keys, IV if using CBC encryption mode.

After a successful key exchange operation, the client will send a **ChangeCipherSpec** message to inform the server that key exchange has been completed successfully. From this point on, all subsequent messages will be encrypted and authenticated. To guard against a man-in-the-middle attack, a final **EncryptedHandshakeMessage** is sent to the server. The message contents contain the MAC of all handshake messages which have been sent and received, using the negotiated encryption and authentication parameters.

4. When the server receives the ClientKeyExchange message, it performs an ECDH operation to generate the same premaster secret. It goes through the same process as described in step 3 above to independently generate the same set of cryptographic keys and IV. A ChangeCipherSpec message is sent to the client, followed by an EncryptedHandshakeMessage.

   The SSL handshake process is done once both sides verify the EncryptedHandshakeMessage successfully.
5. All subsequent data transmissions are encrypted and authenticated. A MAC of the TLS header and data block are created. The TLS header, data block, encryption padding (if exists), MAC, etc. are encrypted using the bulk data encryption key before it's sent through the wire.

In our tests, we have consciously made a decision to utilize ephemeral ECDH as the key exchange algorithm and AES as the bulk encryption algorithm. The following reasons, in addition to those of performance and security, are provided for this decision:

1. Ephemeral ECDH provides for perfect forward secrecy. It also has speed advantages as compared to EDH (Ephemeral Diffie-Hellman).
2. Since we are drawing a comparison between RSA and ECC based certificates, the authentication algorithms are RSA and ECDSA respectively. ECDSA is a component in NSA's Suite B and is a recommended authentication method as per NIST[16].
3. The bulk encryption algorithm is AES256 and AES is a standard adopted by NIST and NSA (Suite B).

Please note that TLS protocol version 1[17] is the basis of all testing. This is the ubiquitous version as of writing this article.[18]

Based on the considerations so outlined, we arrive at the cipher suites that were utilized for our tests:

1. ECDHE-ECDSA-AES256-SHA (0xC0,0x0A)
2. ECDHE-RSA-AES256-SHA (0xC0,0x14)

Our tests do not include OCSP or CRL checks and client authentication is not factored in.

The following table specifies the public-key cryptographic operations in the handshake based on these cipher suites.

Table 1: Public Key Cryptographic Operations

	Client	Server
RSA	RSA_{verify} + RSA_{verify} + $ECDHE_{op}$	$ECDHE_{op}$ + RSA_{sign}
ECDSA	$ECDSA_{verify}$ + $ECDSA_{verify}$ + $ECDHE_{op}$	$ECDHE_{op}$ + $ECDSA_{sign}$

Here, an ECDHEop implies the generation of the shared premaster secret and RSA_{verify} and $ECDSA_{verify}$ are different operations to verify the Server's certificate as well as verify the signed ECDHE key from the server (this signing by the server is denoted by RSA_{sign} and $ECDSA_{sign}$ respectively). There is a cost of generation of the key-pair for ECDHE at both sides as well (not shown in the table).

[16] FIPS: Digital Signature Standard (DSS)
[17] TLS Protocol Version 1.0
[18] TLS Survey by Tom Ritter

The certificates utilized in our tests have the following hierarchies:

Table 2: Certificate Hierarchies

RSA:
1. RSA 2048 root CA, RSA 2048 intermediate CA and RSA 2048 end-entity certificate. We will use the term "RSA-2048" to refer to this configuration of the certificate hierarchy. 2. RSA 3072 root CA, RSA 3072 intermediate CA and RSA 3072 end-entity certificate. We will use the term "RSA-3072" to refer to this configuration of the certificate hierarchy.
ECC:
1. ECC P-384 root CA, ECC P-256 intermediate CA and ECC P-256 end-entity certificate. We will use the term "ECC-256" to refer to this configuration of the certificate hierarchy.

Consequently - as per the certification hierarchy above - during the certificate verification process at the client there would be one more verify operation (not shown in the table above) and that would be the root CA verifying the signature of the intermediate CA.

The following table demarcates the OpenSSL 1.0.1c "speed"[19] numbers for RSA 2048 bits, ECC P-246 and P-384 bits. And OpenSSL is the enabler for SSL communications for Apache Web Server – one of the two web servers (the other being IIS) used to gather numbers for this exercise on the Linux OS. One can draw an inference based on the time requirements in the "sign" column and data in the Public Key Cryptographic Operations in an SSL Handshake table that ECC has an inherent advantage on the server where the signing takes place. For instance, for a XLarge server instance, ECC P-256 bit sign operation is approximately 7% of RSA 2048 sign operation. However for a verify operation that occurs on the client, ECC P-256 is 566% of RSA 2048.

Table 3: OpenSSL 1.0.1c Speed Numbers with 64 bit ECC Optimizations

Certificate type	XLarge (c1.xlarge)				Medium (c1.medium)			
	Sign	Verify	Sign/s	Verify/s	Sign	Verify	Sign/s	Verify/s
RSA 2048 bits	0.002860s	0.000090s	349.7	11092.7	0.002925s	0.000092s	341.9	10863.7
256 bit ECDSA (nistp256)	0.0002s	0.0005s	4656.1	1848.7	0.0002s	0.0006s	4492.4	1773.6
384 bit ECDSA (nistp384)	0.0004s	0.0020s	2341.2	487.9	0.0004s	0.0021s	2269.4	470.2

[19] OpenSSL speed manual page

Security Bits/Security Strength Equivalence

The following table specifies the comparable key sizes for symmetric and asymmetric cryptosystems based on equivalent security strengths. For instance: a 256 bit ECC key provides equivalent security as compared to a 3072 bit RSA key. And also note that the root CA for the ECC-256 end-entity certificate has a key size of 384 bits.

Table 4: Comparable Key Sizes[20]

Symmetric Key Size (bits)	RSA, DSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	**3072**	**256**
192	7680	384
256	15360	521

Test Methodology

The methodology that was followed is explained in the following sections. We planned to determine the relative differences in a SSL handshake between a client and a server configured with either a RSA or an ECC based server certificate. The primary difference that is articulated herein is that due to the public-key cryptographic algorithms (as in RSA or ECC). Consequently, we kept the symmetric bulk encryption algorithm and the digest algorithm to be similar in each of the two cipher suites associated with these two algorithms. Please note the choice of ephemeral ECDH in the key exchange was due to the forward secrecy that it provides and we do see a popular move towards this as well.

Another aim of the exercise was to present a realistic assessment of the SSL handshake and therefore we consciously made certain decisions including running the tests with TLS v1.0 (due to that being supported across the wide variety of browsers and servers), running the servers under varying degrees of load, allowing fetches (HTTP GETS) of resources of varying sizes and 68% session reuse[21] (2 out of 3 handshakes reuse the previous SSL state). We also ran the tests in an environment and model that is gaining in popularity and is easily available to enable the reader to repeat the tests: Amazon EC2.

A “complete handshake” includes certificate chain verification and this in turn encompasses the process in which the trusted root verifies the intermediate that in turn verifies the end-entity certificate. This verification is built-in the applications that are used in the test. The complete handshake also includes the operations that were specified in the table on Public Key Cryptographic Operations as well.

[20] The case for Elliptic Curve Cryptography
[21] Revisiting SSL: A Large Scale Study of the Internet’s Most Trusted Protocol

Client Applications

We have utilized two different client applications for the test:

1. JMeter 2.8[22] : The JMeter application was utilized to put the web server under varying degrees of load and was configured with a ThreadGroup containing a HTTPRequest sampler that utilizes HttpClient4 and HTTPS to retrieve different sized files through a HTTP GET operation. The program utilizes the JSSE framework and the chain verification process within the ambit of it. We updated the source to include support for specifying a cipher suite, configuring it to disable session reuse so each request is a complete SSL handshake and enabling parameterized session reuse value as well. For instance: the program can be configured to enable 68% session reuse (2 out of 3 handshakes are abbreviated). The JMeter application was also updated to enable chain verification by including a call to the underlying TrustManager implementation to verify the server's certificate. We also disabled Nagle's Algorithm to simulate browsers and to eliminate the wait on the client to transmit a 1 byte ChangeCipherSpec message. For the remainder of this document we would refer to an instance running JMeter as a desktop client.
2. Android program: This program was created utilizing the Android SDK and was deployed to a Samsung Galaxy SIII mobile device with LTE. The program had similar specifications as JMeter as in the Nagle's algorithm was also disabled, it is programmed to choose a cipher suite, disable or enable session reuse and enable chain verification.

Both these programs disabled OCSP and CRL checks as well as any hostname validation.

Web Servers

The web server be it Apache or IIS was configured with the corresponding certificate for which the test was to be run. Since the Apache web server running on a Linux instance is dependent on OpenSSL for providing HTTPS support, we enabled the ECC optimizations[23] in this version of OpenSSL.

The Apache web server version 2.4.3 was deployed on a RedHat Enterprise Linux 6.3 AMI (Amazon Machine Image)[24]. The reason we chose this version for Apache was that ECC as an authentication mechanism is available in the 2.4.x codebase without it needing to be patched in. This image had been updated to use OpenSSL v1.0.1c with the ECC optimization patches as previously mentioned. Although this version of Apache allows configuration of multiple certificates simultaneously, the certificates were configured one at a time.

The IIS web server v8 was utilized for the Windows platform tests and it was deployed on a Win 2012 RTM Standard Edition image[25]. The certificates were configured through bindings to port 443 one at a time as well.

[22] Apache JMeter
[23] 64-bit ECC Optimizations
[24] Red Hat AMI ID ami-cc5af9a5 (x86_64)
[25] Windows AMI ID ami-7663f01f (x86_64)

The desktop client was also built on the same image as the Apache web server and its runtime environment was JDK 7u10. The JMeter application was setup to trust the roots of the certificate hierarchies. For instance: the ECC P-384 root, the RSA 2048 root and the RSA 3072 root.

Files of different sizes were retrieved while running the tests. The desktop client was setup to retrieve files of varying sizes:

1. 0K => empty file, this essentially implies a zero size payload not including the HTTP response packet with the headers. A payload of essentially size 0 bytes
2. 200K
3. 1200K

The android program retrieved a 90K file.

From information sourced from HttpArchive[26] and others[27], it was determined that the size of the page (on the wire) is around 1.2M and an average browser creates 6 connections to a host[28] and consequently that has determined the size of the files utilized in the tests. It is of course, a pertinent question as to why regular web pages of these transfer sizes not created. We utilized a monolithic text page instead of html and its constituents for keeping the experiment simple and the variables limited.

Amazon EC2 Environment, AMIs and Instance Types

The Amazon EC2 environment was setup that included a VPC to host the test setup with all of the instances including the desktop clients and the servers. The tenancy of the instances was specified to be “dedicated” and CloudWatch monitoring was configured to be “detailed”. CloudWatch was thereafter the source for the “Average CPU Utilization”, “Max Network In” and “Max Network Out” statistics. The instance types[29] were selected based on certain criteria and that is collected below.

1. High-CPU Medium Instance: Almost all servers were of this type unless otherwise stated. The primary reason is to saturate the server in terms of CPU Utilization before saturating the network pipe or any other resource. Also an ancillary benefit is that the numbers of clients that are required to do that are reduced as well.
2. High-CPU Extra Large Instance: All the desktop clients were of this type. We made sure that the client was not a throttling factor by involving a multiple number of clients in tests that were to saturate the server.

[26] HttpArchive
[27] Web Performance Today
[28] BrowserScope
[29] Amazon EC2 Instance Types

We have seen a number of companies moving on or having some presence on cloud computing platforms - Amazon EC2 and other cloud computing platforms have become popular. We selected EC2 to be the platform to run our experiments on also due to easy availability of an environment for our readers to repeat the tests if wished. An underlying goal of the entire exercise was to capture realistic conditions such as the servers running under different loads with numerous clients; for tests that included the mobile device, the tests were run on the ATT LTE network.

The specification for the instance types and their usage is in the following table. For the remainder of this document, we would refer to "c1.xlarge" as XLarge and "c1.medium" as Medium.

Table 5: Amazon AMIs and Instance Types

	Linux / Win	OS version	Instance Type	Apache / IIS version	JMeter / Client
Server	Linux	Red Hat Enterprise Linux Server release 6.3 64 bit	High-CPU Extra Large Instance (c1.xlarge) And High-CPU Medium Instance (c1.medium)	Apache 2.4.3	-
Server	Win	Windows Server 2012 64 bit	High-CPU Extra Large Instance (c1.xlarge) And High-CPU Medium Instance (c1.medium)	IIS 8	-
Client – Desktop	Linux	Red Hat Enterprise Linux Server release 6.3 64 bit	High-CPU Extra Large Instance (c1.xlarge)	-	JMeter 2.8 with updates
Client - Mobile	Android	JellyBean aka Android 4.1.1	Samsung Galaxy S III	-	Android program created for the tests

Test Strategy

The strategy encompasses configuration of the server (be it IIS or Apache) with a certificate and the appropriate intermediates. There are three certificates utilized in the tests: one for ECC and two for RSA and the server was configured consecutively with each. We ran three kinds of experiments through the desktop clients:

1. **Server Performance Metrics (Sequential):** This test specified sequential invocations through a single client and subsequent retrieval of the statistics at the SSL layer using the SSLDump[30] utility. All the tests involved packet capture (tcpdump on Linux, WireShark on Win) and that was utilized to extract SSL messages through the SSLDump utility and extracted the time associated with the server transactions across a large number of iterations. We gathered the "total server time" that included the time printed for the server messages and we excluded any SSL client messages. However we limited our parsing to the output generated by SSLDump. The resulting SSL traffic captured by SSLDump utility was parsed utilized a python script. The server transactions include steps that originate at the server in the SSL handshake (Server Hello, Certificates… Finished) and the actual payload (application_data). There was zero session reuse in this experiment. The server was very lightly loaded due to a singular client being the source of sequential invocations to the server.
2. **Client: Desktop Response Time versus Throughput Metrics (DRVT):** A test of this type loaded the server by running the same transaction concurrently through multiple clients and gathering latency (response time) and throughput at the client. Data points in the test were gathered by generation of incrementally increasing levels of load on the server till the server was saturated in terms of the CPU resource. The statistics gathered were at 0 % and 68% session reuse. An instance of a transaction encompasses loading the server to a particular CPU and resource utilization, waiting for the throughput and response time to stabilize and then collecting the statistics. The steps were repeated at incremental levels of server resource utilization.
3. **Client: Mobile Response Time and Throughput Metrics (MRTT):** A test of this type coincided with the DRVT tests. When the DRVT tests had loaded the server to a particular utilization level, this test was initiated and results accumulated over many different runs over different server loads. The android program performed sequential invocations on the ATT LTE network and displays the average response time based on the cipher suite selected. The corresponding throughput was the actual throughput achieved when simultaneously loading the server with DRVT tests (200K GET for Apache and IIS with 0% session reuse). An example of a transaction includes utilizing the DRVT to load the server and then running the mobile application by specifying the IP address of the web server, the cipher suite, the number of iterations, the file to GET and the session reuse percentage and collecting the response time. We utilized a session reuse value of 68%.

Tests and Analysis

Server Performance Metrics (SEQUENTIAL)

The tests were performed on two large instances. Two different cipher suites as we detailed earlier with 2 different authentication algorithms are utilized in this test case:

1. For ECC: the cipher suite is ECDHE-ECDSA-AES256-SHA (0xC0,0x0A)
2. For RSA: the cipher suite is ECDHE-RSA-AES256-SHA (0xC0,0x14)

[30] SSLDump

We see that ECC-256 hierarchy outperforms RSA-2048 and RSA-3072 irrespective of the file sizes (the 90K and 200K sized files).

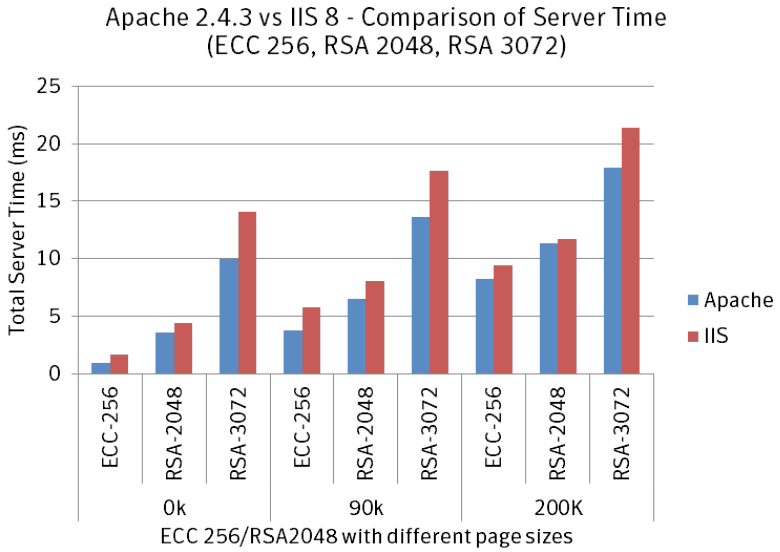

Figure 2: Server Time

Client: Desktop Response Time Versus Throughput Metrics (DRVT)
Here we capture the test strategy (on this page) that encompasses load tests. The entries in table below and the sections below explicate the tests that were run. For instance: test "0K GET with 0% reuse" was run on the a XLarge desktop Client with an XLarge server and was run for both Apache and IIS web servers with 0% session reuse that implies that each SSL handshake was a complete handshake and not an abbreviated one.

Also note that a 0K file does not specify an empty payload – there were still HTTP headers transmitted in the process.

Table 6: Test Cases and Scenarios

Test	Specifications		Web Server	Session Reuse %	File Size
	Desktop Client	Server			
0K GET with 0% reuse	XLarge	XLarge	IIS	0%	0K
			Apache		
200K GET with 0% reuse	XLarge	Medium	IIS	0%	200K
			Apache		
200K GET with 68% reuse	XLarge	Medium	IIS	68%	200K
			Apache		
1200K GET with 0% reuse	XLarge	Medium	IIS	0%	1200K
			Apache		

0K GET with 0% Reuse

With 0% session reuse and the implication of each handshake being a full handshake with the required CPU processing involved, we have seen the limits of CPU saturation on the instance running Apache and for RSA-3072 was reached at around 500 requests per second, for RSA-2048 it was around 1300 and ECC-256 proved to be resilient till 2800. The salient point is that ECC-256 was able to cope with a much higher number of transactions. Although the underlying data points, as in throughput and latency, were different for both Apache and IIS the inference drawn was the same and in favor of ECC-256. Note that ECC-256 is considered to be as secure as RSA-3072 as per the table in the "Security Bits / Security Strength equivalence" section.

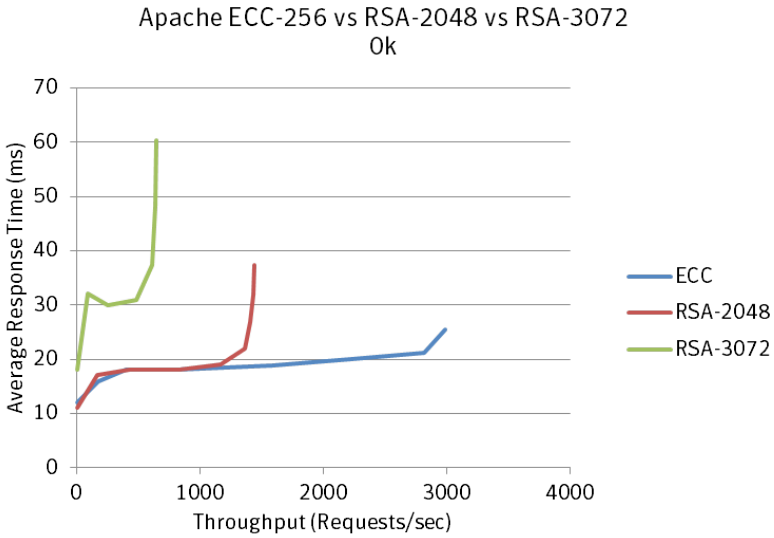

Figure 3: 0K GET with 0% reuse - Apache

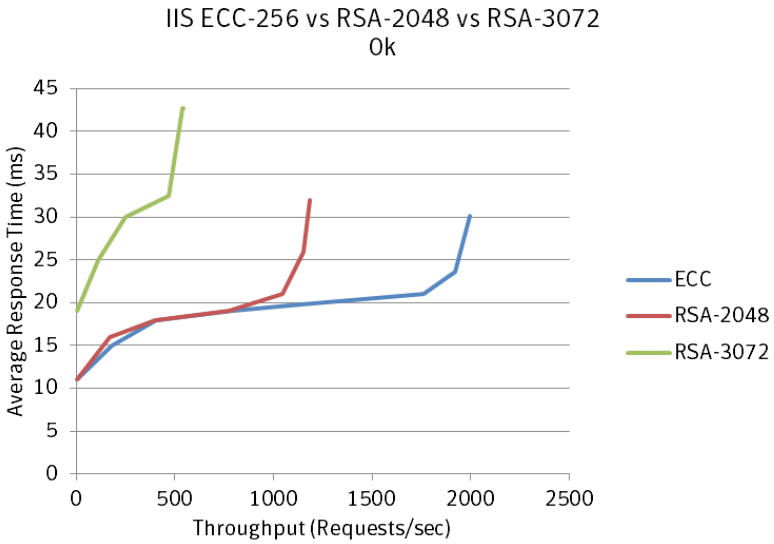

Figure 4: 0K GET with 0% reuse - IIS

200K GET with 0% reuse

The reason for hosting the web server on a Medium instance was to provide for a reduced the number of clients to be able to saturate the server in terms of CPU utilization. The results are in line with the "*0K GET with 0% reuse*".

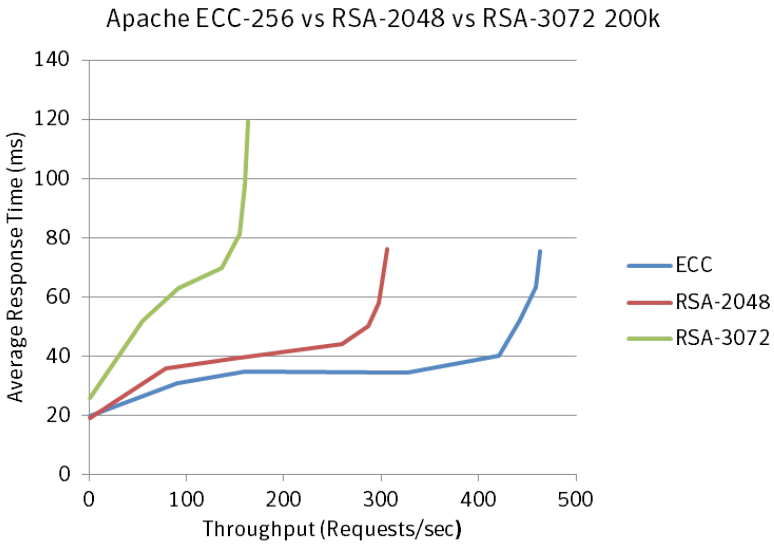

Figure 5: 200K GET with 0% reuse - Apache

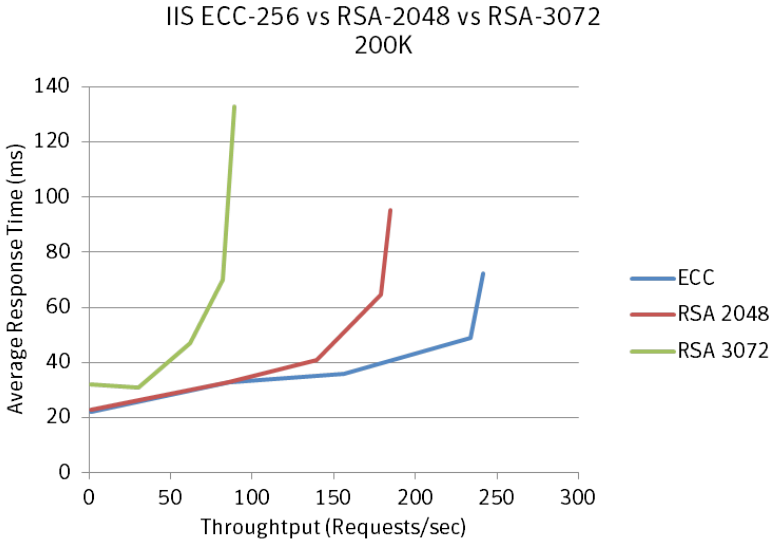

Figure 6: 200K GET with 0% reuse - IIS

200K Get with 68% Reuse

A session reuse of 68% results in 2 out of 3 handshakes being abbreviated. The average response time drops vis-à-vis the test in the preceding section. Also the throughput increases and the saturation gap between the three narrows as well. If session reuse percentages are increased and the graphs plotted, we will witness a narrowing gap.

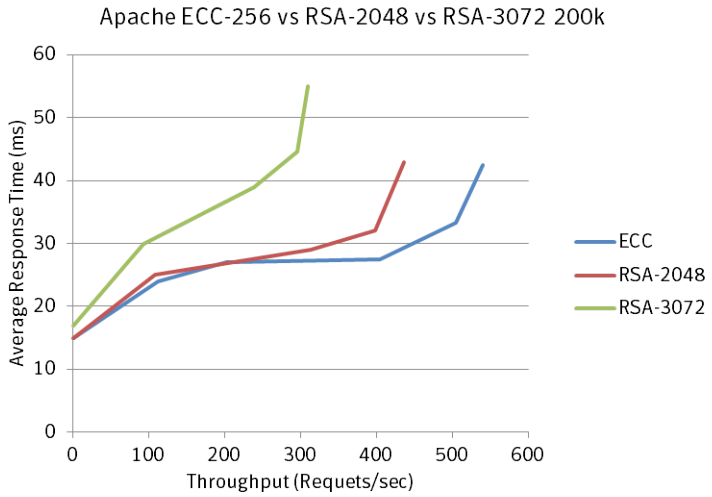

Figure 7: 200K GET with 68% reuse - Apache

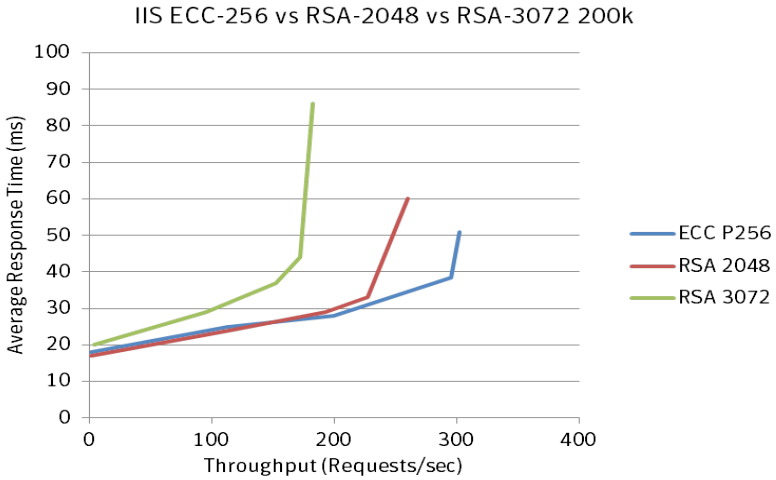

Figure 8: 200K GET with 68% reuse - IIS

1200K Get with 0% Reuse

An interesting phenomenon is observed in case of Apache where the network pipe gets saturated for ECC-256 and RSA-2048 but the CPU utilization limits are reached for RSA-3072. For IIS, the CPU utilization limits are reached for all three.

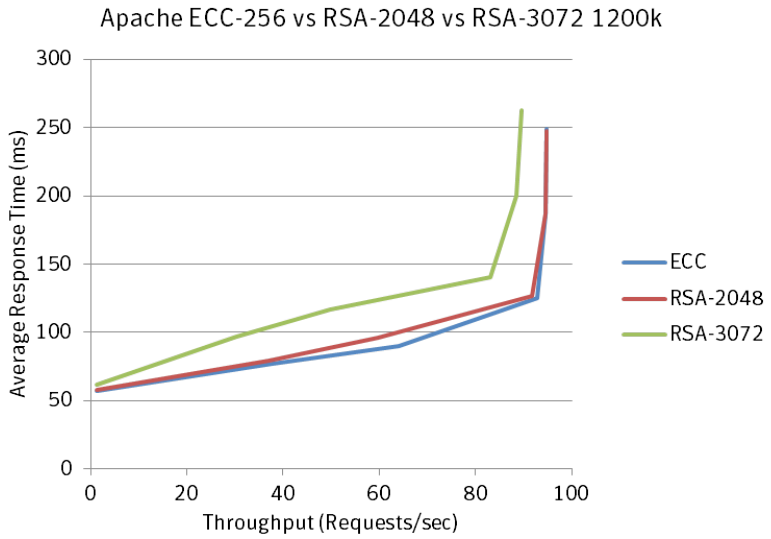

Figure 9: 1200K GET with 0% reuse - Apache

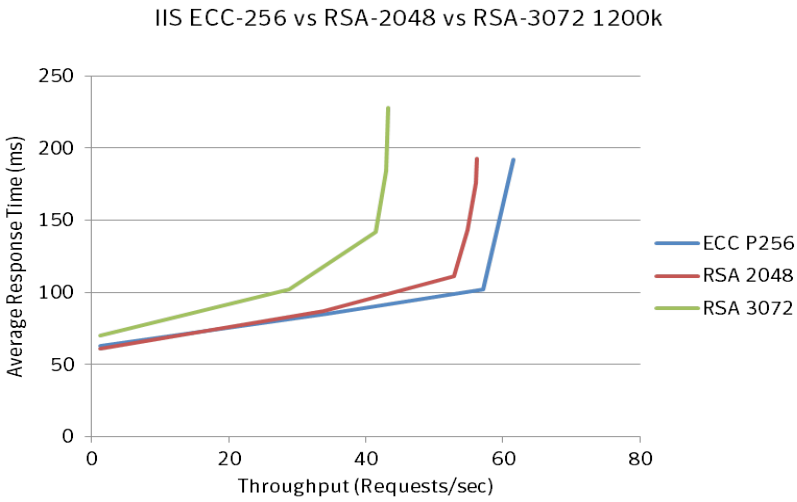

Figure 10: 1200K GET with 0% reuse – IIS

Client: Mobile Response Time and Throughput Metrics (MRTT)

Here, the aim was to collect the response times of a transaction that was initiated through a mobile device. More details on this page.

Test	Specifications		Web Server	Session Reuse %	File Size
	Desktop Client	Server			
90K GET with 68% reuse on Mobile	XLarge	Medium	Apache	68%	90K
			IIS		

90K Get with 68% Reuse on Mobile

The mobile tests have a dependency on the mobile/wireless provider (AT&T LTE) network and that is a variant and therefore numbers captured here are likely to change on subsequent tests. We had run these tests a multitude of times and a general trend was observed in that RSA beats ECC and this was confirmed by the response time numbers clocked on the mobile device. That is emphasized through the inclusion of the following graphs.

From these, we do see the effect of the vagaries of LTE traffic but a pattern is observed in which RSA (both RSA-2048 and RSA-3072) outperforms ECC-256. This can be attributed to the effect of the public-key cryptographic operations that are more intensive on the client for ECC-256.

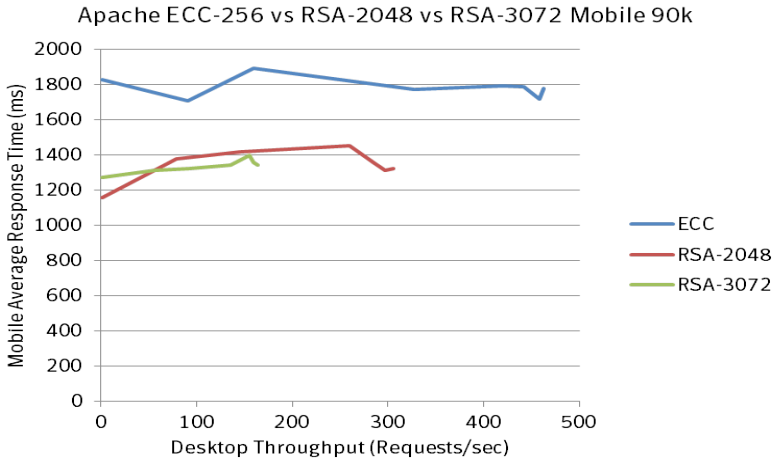

Figure 11: 90K GET with 68% - Apache

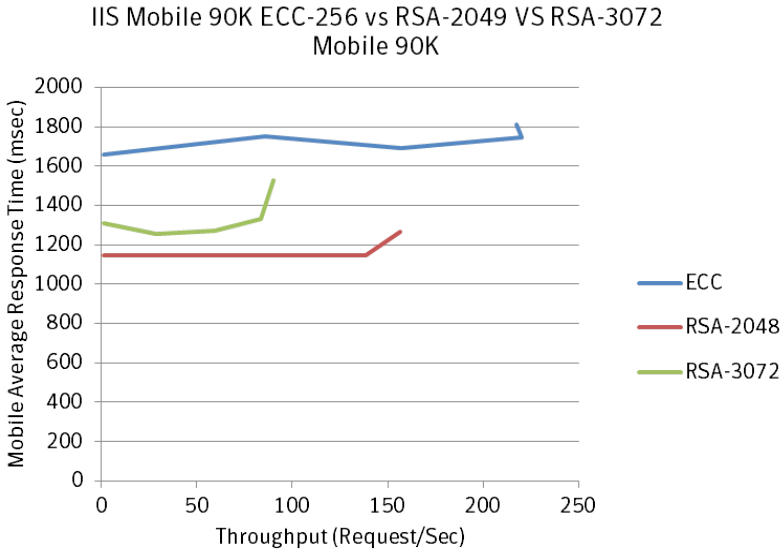

Figure 12: 90K GET with 68% - IIS

ECC Ubiquity

ECC has found ubiquitous support across a wide variety of cryptographic engines and utilities. *A subset of those is provided below*. If you have an addition to this list, please contact us at hari_veladanda@symantec.com

Table 7: Cryptographic Libraries which Support ECC

Library	Version	References
NSS	3.11 onwards	ECC TLS Ciphers supported since version 3.11 *http://www.mozilla.org/projects/security/pki/nss/nss-3.11/nss-3.11-algorithms.html* (TLS ECC cipher suites are listed in the 3.11, ECC support contributed by Sun was added in 3.8) NSS 3.8 Plan (with details on ECC implementation) *http://www.mozilla.org/projects/security/pki/nss/nss-3.8/nss-3.8-plan.html* NSS 3.8 Release Notes *http://www.mozilla.org/projects/security/pki/nss/nss-3.8/nss-3.8-release-notes.html* Mozilla Bug 195135 - Add support for Elliptic Curve Cryptography to NSS & SSL *https://bugzilla.mozilla.org/show_bug.cgi?id=195135*
OpenSSL	1.0 onwards	OpenSSL change log *http://www.openssl.org/news/changelog.html* (64-bit optimized ECC implementations were added in version 1.0.1)
MS Crypto API NG	Windows Server 2008, Windows Vista and above	Runtime requirements *http://msdn.microsoft.com/en-us/library/windows/desktop/aa376210(v=vs.85).aspx* Details on ECC support http://msdn.microsoft.com/en-us/library/windows/desktop/bb204775(v=vs.85).aspx
BouncyCastle	1.32 onwards	Support added for SEC/NIST elliptic curves in version 1.32. *http://www.bouncycastle.org/releasenotes.html* (Elliptic curve support starting with version 1.04)

Library	Version	References
JSSE	6 onwards	Supports ECC cipher suites as per RFC 4492: *http://docs.oracle.com/javase/6/docs/technotes/guides/security/SunProviders.html#SupportedCipherSuites*
BSAFE	4.0	The following press releases indicate that ECC support was added from BSAFE 4.0 *http://www.rsa.com/press_release.aspx?id=590* *http://www.rsa.com/press_release.aspx?id=544*

We have also tested with various browser versions and all of them except Safari do support ECC based cipher suites. The following table captures the browser support matrix. The aim was to demonstrate ECC cipher suite support by major browsers and the results from sampling of a few versions is provided.

Table 8: Browser Support Matrix

	IE	Firefox	Chrome
Win-7	Works (IE 8 and IE 9)	Works (FF 19)	Works (Chrome 25)
Win Vista	Works (IE9, IE8, IE7)	Works (FF 19)	Works (Chrome 25)
Win XP	Unsupported	Works (FF 19)	Unsupported
Linux	Not Available	* Works (FF 11 on RHEL 5.1) * According to *https://wiki.mozilla.org/NSS:Versions* NSS 3.11 has been used by FF since FF 2.0.0.14	
Mac	Not Available (last version is IE5)	Works (FF 19)	Works (Chrome 25)

Mobile Devices:

- Chrome on Android
- iPhone 4 Safari doesn't include it
- Blackberry doesn't include it

References

SSLDump
http://www.rtfm.com/ssldump/

The SSLDump 0.9b3 version is utilized to delineate SSL traffic on HTTP (HTTPS). It operates on the packet capture either through tcpdump on Linux or WireShark on windows.

SSL and TLS
http://www.amazon.com/SSL-TLS-Designing-Building-Systems/dp/0201615983

Eric Rescorla (For an in-depth study of SSL)

Implementing SSL/TLS using Cryptography and PKI
http://www.amazon.com/Implementing-SSL-TLS-Using-Cryptography/dp/0470920416

Joshua Davies

Appendix

Figures

Figure 1: Ephemeral Handshake 7
Figure 2: Server Time 17
Figure 3: 0K GET with 0% reuse - Apache 18
Figure 4: 0K GET with 0% reuse - IIS 18
Figure 5: 200K GET with 0% reuse - Apache 19
Figure 6: 200K GET with 0% reuse - IIS 19
Figure 7: 200K GET with 68% reuse - Apache 20
Figure 8: 200K GET with 68% reuse - IIS 20
Figure 9: 1200K GET with 0% reuse - Apache 21
Figure 10: 1200K GET with 0% reuse – IIS 21
Figure 11: 90K GET with 68% - Apache 22
Figure 12: 90K GET with 68% - IIS 22

Tables

Table 1: Public Key Cryptographic Operations 10
Table 2: Certificate Hierarchies 11
Table 3: OpenSSL 1.0.1c Speed Numbers with 64 bit ECC Optimizations 11
Table 4: Comparable Key Sizes20 12
Table 5: Amazon AMIs and Instance Types 15
Table 6: Test Cases and Scenarios 17
Table 7: Cryptographic Libraries which Support ECC 23
Table 8: Browser Support Matrix 24

More information:

Visit our website

http://go.symantec.com/ssl-certificates/

To speak with a Product Specialist in the U.S.

Call 1 (866) 893-6565 or 1 (650) 426-5112

To speak with a Product Specialist outside the U.S.

For specific country offices and contact numbers, please visit our website.

About Symantec

Symantec protects the world's information, and is a global leader in security, backup and availability solutions. Our innovative products and services protect people and information in any environment – from the smallest mobile device, to the enterprise data center, to cloud-based systems. Our world-renowned expertise in protecting data, identities and interactions gives our customers confidence in a connected world. More information is available at www.symantec.com or by connecting with Symantec at: go.symantec.com/socialmedia.

Symantec Corporation World Headquarters

350 Ellis Street

Mountain View, CA 94043 USA

1 (800) 721 3934

www.symantec.com

UID:202/06/13