

Speeding up Secure Web Transactions Using Elliptic Curve Cryptography

Vipul Gupta, Douglas Stebila*, Stephen Fung*,
Sheueling Chang Shantz, Nils Gura, Hans Eberle
Sun Microsystems, Inc.
2600 Casey Avenue
Mountain View, CA 94043
<http://research.sun.com/projects/crypto>

Abstract

Elliptic Curve Cryptography (ECC) is emerging as an attractive alternative to traditional public-key cryptosystems (RSA, DSA, DH). ECC offers equivalent security with smaller key sizes resulting in faster computations, lower power consumption, as well as memory and bandwidth savings. While these characteristics make ECC especially appealing for mobile devices, they can also alleviate the computational burden on secure web servers.

This article studies the performance impact of using ECC with SSL, the dominant Internet security protocol. We created an ECC-enhanced version of OpenSSL and used it to benchmark the Apache web server. Our results show that, under realistic workloads, an Apache web server can handle 13%–31% more HTTPS requests per second when using ECC-160 rather than RSA-1024 reflecting short-term security levels. At security levels necessary to protect data beyond 2010, the use of ECC-224 over RSA-2048 improves server performance by 120%–279%.

1. Introduction

Secure communication is an intrinsic requirement of today's world of on-line transactions. Whether exchanging financial, business or personal information, people want to know with whom they are communicating (authentication) and they wish to ensure that the information is neither modified (data integrity) nor disclosed (confidentiality) in transit. The Secure Sockets Layer (SSL) protocol [12] is the most popular choice for achieving these goals.¹

*On a student internship from the University of Waterloo.

¹Throughout this paper, we use SSL to refer to all versions of the protocol including version 3.1 also known as Transport Layer Security (TLSv1.0) [10].

The SSL protocol is application independent – conceptually, any application that runs over TCP can also run over SSL. This is an important reason why its deployment has outpaced that of other security protocols such as SSH [32], S/MIME [25] and SET [19]. There are many examples of application protocols like TELNET, FTP, IMAP and LDAP running transparently over SSL. However, the most common usage of SSL is for securing HTTP [11], the main protocol of the World Wide Web.²

Between its conception at Netscape in the mid-1990s, through its standardization within the IETF (Internet Engineering Task Force) in the late-1990s, the protocol and its implementations have been scrutinized by some of the world's foremost security experts [31]. Today, SSL is trusted to secure transactions for sensitive applications ranging from web banking, to stock trading, to e-commerce.

Unfortunately, the use of SSL imposes a significant performance penalty on web servers. Coarfa *et al.* [9] have reported secure web servers running 3.4 to 9 times slower compared to regular web servers on the same hardware platform. Slow response time is a major cause of frustration for on-line shoppers and often leads them to abandon their electronic shopping carts during check out. According to one estimate, the potential revenue loss from e-commerce transactions aborted due to Web performance issues exceeds several billion dollars annually [33].

In its most common usage, SSL utilizes RSA encryption to transmit a randomly chosen secret that is used to derive keys for data encryption and authentication. The RSA decryption operation is the most compute intensive part of an SSL transaction for a secure web server. Several vendors such as Broadcom, nCipher, Rainbow and Sun now offer specialized hardware to offload RSA computations and improve server performance.

This paper explores the use of Elliptic Curve Cryptography (ECC), an efficient alternative to RSA, as a means of

²The use of HTTP over SSL is also referred to as HTTPS.

Table 1. A comparison of public-key cryptosystems [30].

Public-key system	Examples	Mathematical Problem	Best known method for solving math problem (running time)
Integer factorization	RSA, Rabin-Williams	Given a number n , find its prime factors	Number field sieve: $\exp[1.923(\log n)^{1/3}(\log \log n)^{2/3}]$ (Sub-exponential)
Discrete logarithm	Diffie-Hellman (DH), DSA, ElGamal	Given a prime n , and numbers g and h , find x such that $h = g^x \pmod n$	Number field sieve: $\exp[1.923(\log n)^{1/3}(\log \log n)^{2/3}]$ (Sub-exponential)
Elliptic curve discrete logarithm	ECDH, ECDSA	Given an elliptic curve E and points P and Q on E , find x such that $Q = xP$	Pollard-rho algorithm: \sqrt{n} (Fully exponential)

improving SSL performance without resorting to expensive special purpose hardware. ECC was first proposed by Victor Miller [20] and independently by Neal Koblitz [17] in the mid-1980s and has evolved into a mature public-key cryptosystem. Compared to its traditional counterparts, ECC offers the same level of security using much smaller keys. This results in faster computations and savings in memory, power and bandwidth that are especially important in constrained environments, *e.g.* mobile phones, PDAs and smart cards. More importantly, the advantage of ECC over its competitors increases as security needs increase over time.

Recently, the National Institute of Standards and Technology (NIST) approved ECC for use by the U.S. government [29]. Several standards organizations, such as IEEE, ANSI, OMA (Open Mobile Alliance) and the IETF, have ongoing efforts to include ECC as a required or recommended security mechanism. The use of ECC with SSL is described in an IETF draft [14]. We have implemented that specification in OpenSSL [24] and created a version of the Apache [4] web server capable of handling HTTPS transactions using both RSA and ECC.

The rest of this paper is structured as follows. Section 2 provides an overview of ECC technology. Section 3 describes the SSL protocol and its usage of RSA and ECC public-key cryptosystems. Section 4 outlines the experiments we conducted to compare the performance of RSA and ECC-based SSL. Section 5 presents an analysis of our experimental results. Finally, we summarize our conclusions and discuss future work in Section 6.

2. ECC basics

At the foundation of every public-key cryptosystem is a hard mathematical problem that is computationally intractable. The relative difficulty of solving that problem determines the security strength of the corresponding system. Table 1 summarizes three types of well known public-key cryptosystems. As shown in the last column, RSA, Diffie-

Hellman and DSA can all be attacked using sub-exponential algorithms, but the best known attack on ECC requires exponential time. For this reason, ECC can offer equivalent security with substantially smaller key sizes [18].

Public-key schemes are typically used to transport or exchange keys for symmetric-key ciphers. Since the security of a system is only as good as that of its weakest component, the work factor needed to break a symmetric key must match that needed to break the public-key system used for key exchange. Table 2 shows NIST guidelines [23] on choosing computationally equivalent symmetric and public-key sizes. Notably, the use of 1024-bit RSA does not match the 128-bit or even 112-bit security level now used for symmetric ciphers in SSL, let alone the higher (192- and 256-bit) key sizes offered by AES [22], NIST's new replacement for DES. This underscores the need to migrate to larger RSA key sizes in order to deliver the full security of symmetric algorithms with more than 80-bit keys. Recent work by Shamir and Tromer [26] on integer factorization suggests that the migration needs to happen sooner than previously thought necessary. They estimate that a specialized machine capable of breaking 1024-bit RSA in under one year can be built for \$10-\$50 million dollars. Consequently, RSA Laboratories now considers 1024-bit RSA to be unsafe for data that must be protected beyond 2010 and recommends larger keys for longer term protection [15]. At higher key sizes, RSA performance issues become even more acute. Since the performance advantage of ECC over RSA grows approximately as the cube of the key size ratio, wider adoption of ECC seems inevitable.

Unlike conventional public-key cryptosystems based directly on finite-field arithmetic, ECC operates over a group of points on an elliptic curve defined over a finite field. Its main cryptographic operation is *scalar point multiplication*, which computes $Q = kP$ (a point P multiplied by an integer k resulting in another point Q on the curve). Scalar multiplication is performed through a combination of *point-*

Table 2. Equivalent key sizes (in bits).

Sym-metric	ECC	RSA/DH/DSA	MIPS Yrs to attack	Protection lifetime
80	160	1024	10^{12}	until 2010
112	224	2048	10^{24}	until 2030
128	256	3072	10^{28}	beyond 2031
192	384	7680	10^{47}	
256	512	15360	10^{66}	

additions and *point-doublings*. For example, $11P$ can be expressed as $11P = (2((2(2P)) + P)) + P$. The security of ECC relies on the difficulty of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP), which states that given P and $Q = kP$, it is hard to find k . Besides the curve equation, an important elliptic curve parameter is the *base point*, G , which is fixed for each curve. In ECC, a large random integer k acts as a private key, while the result of multiplying the private key k with the curve’s base point G serves as the corresponding public key.

Not every elliptic curve offers strong security properties and for some curves the ECDLP may be solved efficiently [28]. Since a poor choice of the curve can compromise security, standards organizations like NIST and SECG (Standards for Efficient Cryptography Group) have published a set of curves [8, 29] that possess the necessary security properties. The use of these curves is also recommended as a means of facilitating interoperability between different implementations of a security protocol.

The Elliptic Curve Diffie Hellman (ECDH) key exchange [2] and the Elliptic Curve Digital Signature Algorithm (ECDSA) [3] are elliptic curve counterparts of the well-known Diffie-Hellman and DSA algorithms, respectively. In ECDH key agreement, two communicating parties A and B agree to use the same curve parameters. They generate their private keys k_A and k_B , and corresponding public keys $Q_A = k_A G$ and $Q_B = k_B G$. The parties exchange their public keys and each multiplies its private key and the other’s public key to arrive at a common shared secret $k_A Q_B = k_B Q_A = k_A k_B G$. While a description of ECDSA is not provided here, it similarly parallels DSA.

3. Overview of the SSL protocol

SSL is the most widely used security protocol on the Internet today. It offers encryption, source authentication and integrity protection for data and is flexible enough to accommodate different cryptographic algorithms for key agreement, encryption and hashing. However, the specification describes particular combinations of these algorithms, called *cipher suites*, which have well-understood security properties. For example, the cipher suite *TLS_RSA_WITH_RC4_128_SHA* uses RSA for key ex-

change, 128-bit RC4 for bulk encryption, and SHA for hashing.

The two main components of SSL are the Handshake protocol and the Record Layer protocol. The Handshake protocol allows an SSL client and server to negotiate a common cipher suite, authenticate each other, and establish a shared *master secret* using public-key algorithms. The Record Layer derives symmetric keys from the master secret and uses them with faster symmetric-key algorithms for bulk encryption and authentication of application data.

Since public-key operations are computationally expensive, the protocol’s designers added the ability for a client and server to reuse a previously established master secret. This feature is also known as “session resumption”, “session reuse” or “session caching”. The resulting abbreviated handshake does not involve any public-key cryptography, and requires fewer and shorter messages. Experiments indicate that session caching does indeed improve web server performance [13]. The following subsections describe full and abbreviated SSL handshakes using RSA and ECC.

SSL allows both client- and server-side authentication. However, due to the difficulty of managing user certificates across multiple client devices, the former is rarely used. User authentication, in such cases, happens at the application layer, *e.g.* through passwords sent over an SSL-protected channel. Authentication of the SSL client is not discussed further in this paper.

3.1. RSA-based full handshake

Today, the most commonly used public-key cryptosystem for establishing the master secret is RSA. Figure 1 shows the operation of an RSA-based handshake.³ The client and server first exchange random nonces (used for replay protection) and negotiate a cipher suite with *ClientHello* and *ServerHello* messages. The server then sends its signed RSA public key in the *ServerCertificate* message. The client verifies the server’s RSA key and uses it to encrypt a randomly generated 48-byte number (the *premaster secret*). The encrypted result is sent in the *ClientKeyExchange* message. The server uses its RSA private-key to decrypt the premaster secret. Both end points then use the premaster secret to create a master secret which, along with previously exchanged nonces, is used to derive the cipher keys, initialization vectors and MAC (Message Authentication Code) keys for bulk encryption and authentication by the Record Layer.

3.2. ECC-based full handshake

The operation of an ECC-based SSL handshake, as specified in [14], is shown in Figure 2. Through the

³In both Figure 1 and Figure 2, messages marked with an asterisk are optional and only sent in certain situations.

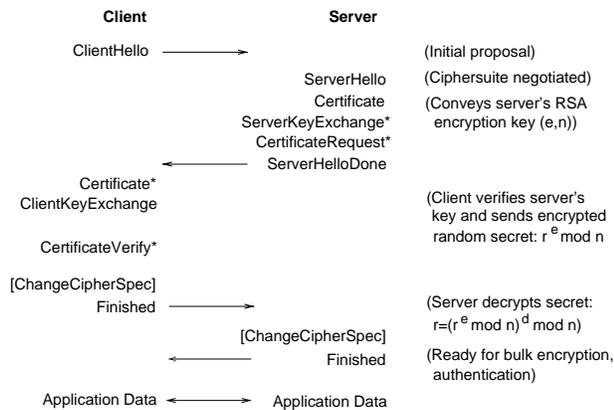


Figure 1. RSA-based SSL handshake.

first two messages (processed in the same way as for RSA), the client and server negotiate an ECC-based cipher suite, e.g. *TLS_ECDH_ECDSA_WITH_RC4_128_SHA*. The *ServerCertificate* message contains the server's ECDH public key signed by a certificate authority using ECDSA. After validating the ECDSA signature, the client conveys its ECDH public key to the server in the *ClientKeyExchange* message. Next, each entity uses its own ECDH private key and the other's public key to perform an ECDH operation and arrive at a shared premaster secret. The derivation of the master secret and symmetric keys is unchanged compared to RSA.

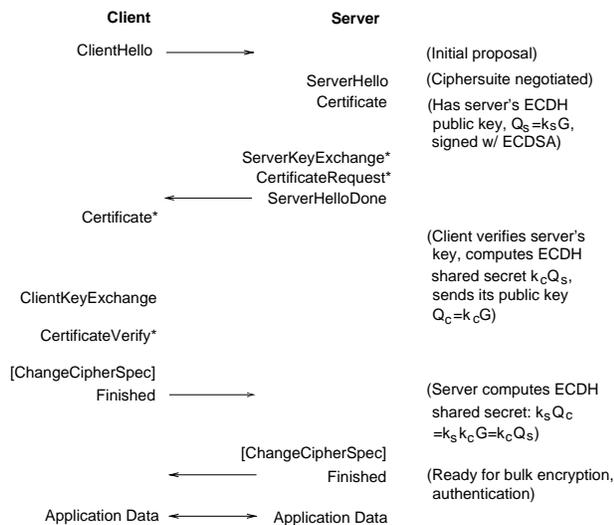


Figure 2. ECC-based SSL handshake (ECDH-ECDSA key exchange).

The use of ECDH and ECDSA in TLS mimics the use of DH and DSA, respectively. The TLS specification [10] already defines cipher suites based on DH and DSA so the

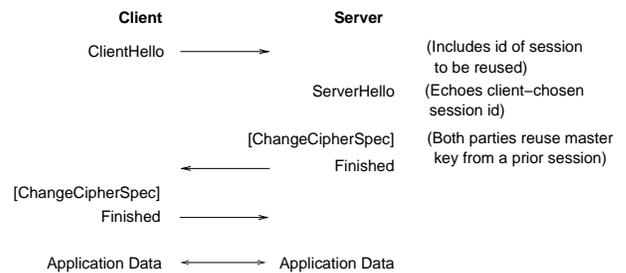


Figure 3. Abbreviated SSL handshake.

incorporation of ECC is not a large change. However, due to the greater popularity of RSA-based key exchange, the rest of this paper compares ECC against RSA rather than DH/DSA.

3.3. Abbreviated handshake

The abbreviated handshake protocol is shown in Figure 3. Here, the *ClientHello* message includes the non-zero ID of a previously negotiated session. If the server still has that session information cached and is willing to reuse the corresponding master secret, it echoes the session ID in the *ServerHello* message.⁴ Otherwise, it returns a new session ID thereby signaling the client to engage in a full handshake. The derivation of symmetric keys from the master secret and the exchange of *ChangeCipherSpec* and *Finished* messages is identical to the full handshake scenario.

The abbreviated handshake does not involve certificates or public-key cryptographic operations, so fewer (and shorter) messages are exchanged. Consequently, an abbreviated handshake is significantly faster than a full handshake.

3.4. Public-key cryptography in SSL

The public-key cryptographic operations performed by a client and server in different modes of the SSL handshake are summarized below and in Table 3.

1. RSA-based handshake

The client performs two RSA public-key operations – one to verify the server's certificate and another to encrypt the premaster secret with the server's public key. The server performs one RSA private-key operation to decrypt the *ClientKeyExchange* message and recover the premaster secret.

2. ECDH-ECDSA-based handshake

The client performs an ECDSA verification to verify the server's certificate and then an ECDH operation using its private ECDH key and the server's public

⁴The likelihood of a cache hit depends on the server's configuration and its current workload.

Table 3. Public-key cryptographic operations in an SSL handshake.

	RSA	ECDH-ECDSA	Abbreviated
Client	$RSA_{verify} + RSA_{encrypt}$	$ECDSA_{verify} + ECDH_{op}$	none
Server	$RSA_{decrypt}$	$ECDH_{op}$	none

ECDH key to compute the shared premaster. All the server needs to do is perform an ECDH operation to arrive at the same secret.

3. *Abbreviated handshake* No public-key operations are performed in the abbreviated handshake since the client and server reuse a previously calculated master secret.

4. Evaluation methodology

The main goal of our experiments was to study the performance impact of replacing RSA with ECC in the SSL protocol. Besides public-key cryptography, an HTTPS transaction involves several other operations including symmetric encryption, hashing, message parsing and file system access. The cost of encryption and hashing depends on the amount of data transferred. The effective cost of public-key operations is determined by the frequency of session reuse which eliminates the need for public-key operations for some transactions. In order to get a realistic estimate of SSL performance, it is important to use an appropriate workload for the tests.

Other studies on SSL performance [5, 6, 9] have either reused the workload for a standard (not secure) web server or synthesized one based on measurements from a sampling of secure web sites [7, 9]. We chose the latter approach since “real-life” workloads for standard and secure web servers are likely to be different. In particular, our workload is based on Badia’s survey [7] of six popular banking, investment and retail sites (Amazon, Datek, ETrade, Fidelity, Merrill Lynch and Wells Fargo). The survey found that the aggregate page size ranges between 10KB to 70KB with a 30KB median.⁵ It also identified two primary usage models impacting SSL session reuse.

1. In the *shopping cart* model, web sites reserve SSL strictly for transporting sensitive information like credit card numbers and personal information. Amazon is a representative example of this usage model – SSL is used when a customer is finalizing a purchase but not when he is browsing through available products. The survey found an average of one new SSL session for every three pages in this usage model.

⁵A “page” consists of an HTML file and one or more embedded images. An average page in [7] consists of an 18KB HTML file and seven image files averaging 1245 bytes each.

2. In the *financial institution* model, the first web page provides a link to a login screen protected by SSL. After a successful login, all subsequent pages are also protected. This usage is exemplified by ETrade and Wells Fargo and, on average, requires one new session for every eight pages.

4.1. Performance metrics

An SSL client fetches web pages sequentially but a server handles multiple requests concurrently. Due to this difference in operation, we use two distinct metrics for evaluating performance from the client’s and server’s perspective.

First-Response Time: This is the delay between initiating an SSL handshake (either full or abbreviated) and receiving the first packet in the HTTPS response. It models the latency experienced by a user between clicking on a URL and seeing the first update to the browser window.

Fetches per second: This measures the rate at which a server fulfills web page requests.

4.2. Experiments performed

We used a public-domain tool called `http_load` [1] to run multiple HTTPS fetches in parallel and measured the rate at which an Apache server satisfies these requests as well as the response time experienced by clients. We performed this experiment using:

- Two different cipher suites: *TLS_RSA_WITH_RC4_128_SHA* and *TLS_ECDH_ECDSA_WITH_RC4_128_SHA* to compare the use of RSA and ECC in an SSL handshake. For each cipher suite, we studied three different security levels — 1024, 1536 and 2048 bits for RSA and 160, 192 and 224 bits for ECC. For ECC, we chose three elliptic curves (`secp160r1`, `secp192r1`, and `secp224r1` [8, 29]) defined over prime integer fields recommended by NIST and/or SECG. In the following we will use ECC-160, ECC-192 and ECC-224 to indicate the aforementioned curves and RSA-1024, RSA-1536, RSA-2048 to indicate RSA with key sizes of 1024, 1536 and 2048 bit, respectively. As shown in Table 2, ECC-160 provides the same security as RSA-1024 and ECC-224 matches RSA-2048. The smaller keys are considered adequate for short-term protection

Table 4. Measured performance of public-key algorithms.

	ECC-160	RSA-1024	ECC-192	RSA-1536	ECC-224	RSA-2048
Time (ms)	3.69	8.75	3.87	27.47	5.12	56.18
Ops/sec	271.3	114.3	258.1	36.4	195.5	17.8
Performance ratio	2.4 : 1		7.1 : 1		11 : 1	
Key-size ratio	1 : 6.4		1 : 8		1 : 9.1	

but the larger keys are recommended for longer-term protection (beyond 2010).

- Four different file sizes: 0KB, 10KB, 30KB and 70KB. These choices allow us to study the relative cost of handshake and record layer processing in SSL under a variety of conditions.
- Four different session reuse models: 0% reuse (all fetches create a new session), 66% reuse (1 new session for every three fetches), 87.5% reuse (1 new session for every eight fetches) and ~100% reuse (only the first fetch from a client creates a new session). We had to modify `http_load` to support session reuse. The measurements obtained for 0% and 100% reuse do not have much practical significance, but they do allow us to analytically predict server throughput for any intermediate reuse figure.

We also used the OpenSSL speed command to measure the performance of raw RSA and ECC operations for different key sizes. Since any security protocol will likely involve other (non public-key) operations, these measurements provide an upper bound on the expected performance improvement from replacing RSA with ECC.

4.3. Platform

Our experiments used the Apache 2.0.45 web server compiled with OpenSSL-SNAP-20030309 (to be released as OpenSSL 0.9.8) using the Sun Forte Developer 7 C compiler. This snapshot of the development version of OpenSSL includes ECC code contributed by Sun Microsystems Laboratories [27]. Enhancements were made to the `mod_ssl` component of Apache in order to make it ECC aware.

We ran the server on a single 900 MHz UltraSPARC III processor with 2GB of memory inside a Sun Fire V480 server running the Solaris 9 operating system.⁶ For the HTTPS clients, we used a prototype Sun Fire server equipped with seven 900 MHz UltraSPARC III processors, 14GB of memory and also running the Solaris 9 operating system. The server and client machines were connected via a 100Mb ethernet network.

⁶The server used in our tests is equipped with four such processors but the other three were turned off for these experiments.

5. Analysis of experimental results

5.1. ECC and RSA microbenchmarks

We used the OpenSSL speed program to measure RSA decryption and ECDH operation for different key sizes (a minor enhancement was made for collecting RSA-1536 numbers). These are the public-key cryptographic operations performed by a server when using RSA and ECC cipher suites, respectively. Our results, shown in Table 4, highlight the performance advantage of ECC over RSA for different security levels. Note that the performance advantage of ECC gets even better than its key-size advantage as security needs increase.

5.2. Server-side costs of an HTTPS fetch

A characteristic measure for the performance of a secure web server is the rate at which it can service HTTPS connections. Figure 4 shows the average time taken by the server to fulfill an HTTPS request for different page sizes and public keys with no session reuse. Microbenchmarks for ECC, RSA, RC4, and SHA allowed us to estimate their relative costs within the overall processing time. RSA decryption continues to be the dominant cost in all of these cases. According to SPECWeb99 which models real-world web traffic, 85% of the files are under 10KB. For such files, RSA takes up anywhere between 63% to 88% of the overall time depending on the security level.

This suggests that efforts to reduce the RSA cost or replace it with a cheaper alternative will have a significant payoff. Indeed, employing ECC reduces the server's processing time for new SSL connections across the entire range of page sizes in our study. We measured a reduction in processing time from 29% for a page size of 70K comparing ECC-160 with RSA-1024 up to 85% for a file size of 10K comparing ECC-224 with RSA-2048. In our experiments, it wasn't until we increased the file size to 1MB that we noticed ECC and RSA performing comparably. At these sizes, public-key costs are no longer dominant and other factors such as network saturation become more important.

5.3. Client latency v/s request rate

Latency v/s throughput plots provide another means to study the efficiency of a system. A more efficient web server

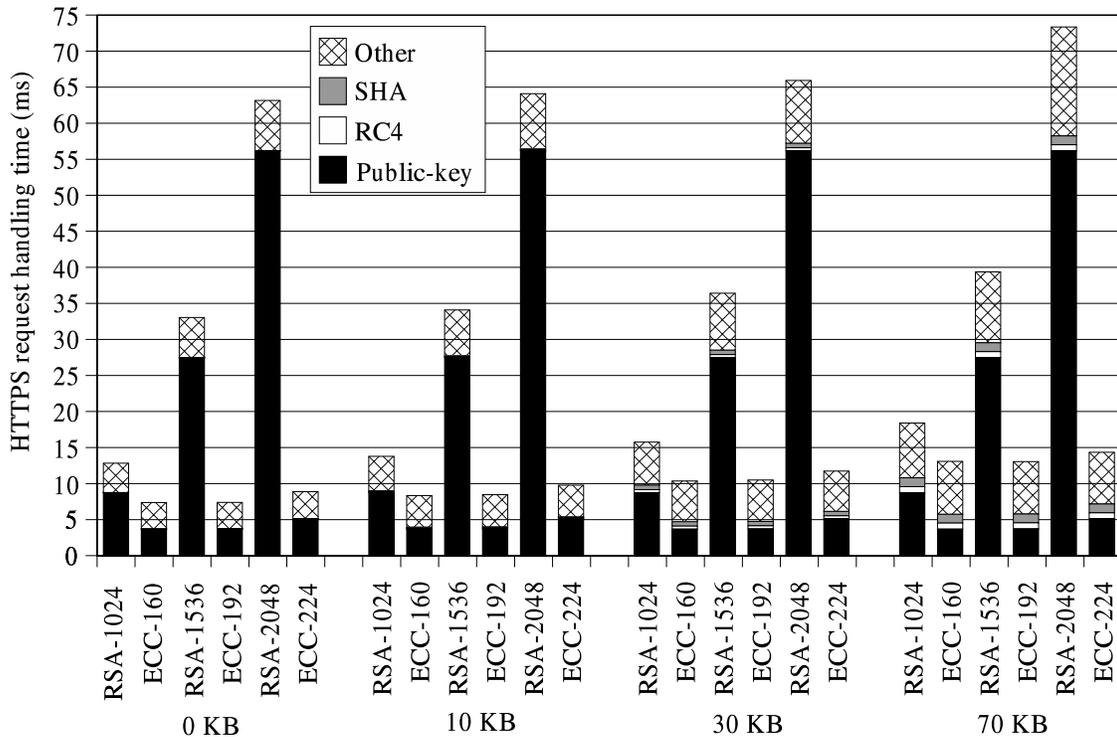


Figure 4. Relative costs in an HTTPS transaction for different file sizes.

can be expected to handle a higher page request rate with lower delays. Figure 5 plots the first-response time reported by http.load as a function of page requests generated per second. The results shown are for a 30KB page size and 66% session reuse reflecting the shopping cart usage model.

Here again, we notice that the use of ECC allows the server to handle a larger number of requests (30%-270% more) compared to RSA. At current security levels and under low load, clients experience comparable latencies for ECC-160 and RSA-1024. That is, the time taken for public-key operations is low compared to SSL processing overheads and network latencies. However, the saturation point of the server is reached earlier with RSA-1024 leading to a sharp increase in latency at around 110 requests per second. For RSA-2048, clients experience 90ms of latency, primarily due to the RSA operation on the server, even under low server load. In comparison, the exhibited latency for ECC-224 is only 35ms or less than 40% of the RSA case.⁷ In addition, the server saturation point for ECC-224 is reached significantly later at around 140 requests per second compared to 40 requests per second for RSA-2048.

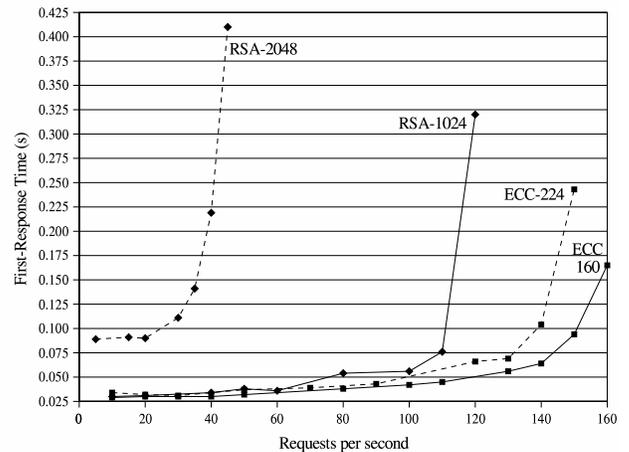


Figure 5. Response time v/s throughput plot for Apache web server.

⁷The measured roundtrip latency between the client and server was 3ms for a 15KB ping payload and 6ms for a 30KB payload.

5.4. Impact of session reuse

Session reuse diminishes the impact of public-key operations on the average HTTPS processing time at the web server. Figure 6 was obtained by measuring the maximum sustained server throughput reported by `http_load` for 30KB page accesses with 0% and 100% session reuse. Throughput numbers for other reuse values were derived analytically using the following formula (here T_r denotes server throughput for $r\%$ session reuse) and the values derived for T_{66} and $T_{87.5}$ were verified empirically.

$$T_r = \frac{1}{\left(1 - \frac{r}{100}\right)/T_0 + \left(\frac{r}{100}\right)/T_{100}}$$

As expected, increasing the percentage of session reuse decreases the performance impact of choosing any particular public-key cryptosystems. However, even under the financial institution usage model with session reuse value as high as 87.5%, an ECC based server handles 13% more requests compared to RSA at smaller key sizes and 120% more at larger key sizes. Considering the shopping cart model, the performance advantage due to ECC at 66% session reuse increases significantly to 31% for ECC-160/RSA-1024 and 279% for ECC-224/RSA-2048. The relative performance of ECC over RSA improves further for smaller pages.

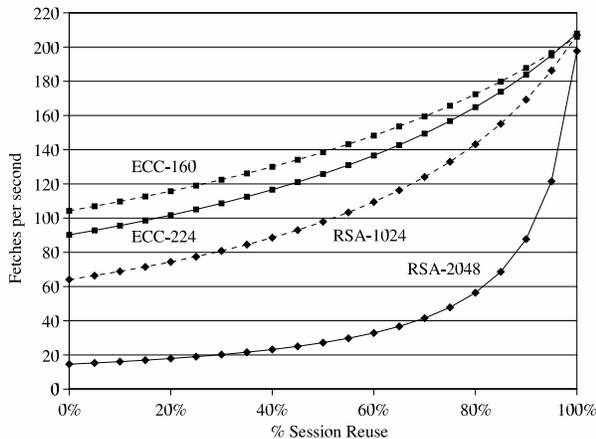


Figure 6. Throughput v/s session reuse plot for Apache web server.

6. Conclusions and future work

The above analysis suggests that the use of ECC cipher suites offers significant performance benefits to SSL clients and servers especially as security needs increase. While this study has focused on SSL, similar improvements can be

expected for other protocols like S/MIME, SSH and IPsec [16].

Already, there is considerable momentum behind widespread adoption of the Advanced Encryption Standard (AES) which specifies the use of 128-bit, 192-bit and 256-bit symmetric keys. As indicated in Table 2, key sizes for public-key cryptosystems used to establish AES keys will correspondingly need to increase from current levels. This would favor the use of ECC over RSA. Furthermore, as users become increasingly sensitive to on-line privacy issues, they are likely to demand protection for more of their transactions. For example, Yahoo! users might demand the option to protect their email accesses, not just the login password, with SSL. Similarly, book lovers might demand privacy protection for their browsing habits on Amazon. We believe these trends bode well for broader deployment of ECC, in not just wireless environments but also desktop/server environments.

Besides OpenSSL, we have also added ECC support to Netscape Security Services (NSS) [21]. This open-source cryptographic library powers the Mozilla/Netscape browsers and the web, directory and messaging servers in Sun's Java Enterprise System. We are now targeting ECC support in these servers and intend to perform a similar study for their representative work loads.

7. Acknowledgments

The authors would like to thank Sumit Gupta for his help in implementing ECC cipher suites in OpenSSL and Shih-Hao Hung for adding session reuse support to `http_load` and his help in setting up our performance test bed.

Sun, Sun Microsystems, Forte Developer, Sun Fire, UltraSPARC, Solaris and Java Enterprise System are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States and other countries.

References

- [1] Acme Labs Software, see http://www.acme.com/software/http_load/
- [2] ANSI X9.63, "Elliptic Curve Key Agreement and Key Transport Protocols", American Bankers Association, 1999.
- [3] ANSI X9.62, "The Elliptic Curve Digital Signature Algorithm (ECDSA)", American Bankers Association, 1999.
- [4] Apache Software Foundation, see <http://www.apache.org/>.
- [5] G. Apostolopoulos, V. Peris, D. Saha, "Transport Layer Security: How much does it really cost?", *In Proc. of IEEE Infocom*, 1999.

- [6] G. Apostolopoulos, V. Peris, P. Pradhan, D. Saha, "Securing electronic commerce: reducing the SSL overhead", *IEEE Network*, Vol 14, No 4, pp 8–16, Jul. 2000.
- [7] L. Badia, "Real World SSL Benchmarking", Rainbow Technologies Whitepaper, Sep. 2001, see <http://www.rainbow.com/insights/whitePDF/RealWorldSSLBenchmarking.pdf>.
- [8] Certicom Research, "SEC 2: Recommended Elliptic Curve Domain Parameters", Standards for Efficient Cryptography, Version 1.0, Sep. 2000.
- [9] C. Coarfa, P. Druschel, D. Wallach, "Performance Analysis of TLS Web Servers", Network and Distributed Systems Security Symposium '02, San Diego, California, Feb. 2002.
- [10] T. Dierks and C. Allen, January 1999. "The TLS Protocol - Version 1.0.", IETF RFC 2246, see <http://www.ietf.org/rfc/rfc2246.txt>
- [11] R. Fielding et al., "Hypertext Transfer Protocol – HTTP/1.1", *RFC 2616*, Jun. 1999.
- [12] A. Frier, P. Karlton and P. Kocher, "The SSL3.0 Protocol Version 3.0", see <http://home.netscape.com/eng/ssl3/>.
- [13] A. Goldberg, R. Buff, A. Schmitt, "Secure Web Server Performance Dramatically Improved by Caching SSL Session Keys", In *Proc. of Workshop on Internet Server Performance, SIGMETRICS'98*, Jun. 1998.
- [14] V. Gupta, S. Blake-Wilson, B. Moeller, C. Hawk, "ECC Cipher Suites for TLS", IETF internet draft <draft-ietf-tls-ecc-03.txt>, work in progress, Jun. 2003.
- [15] B. Kaliski, "TWIRL and RSA Key Size", RSA Laboratories Technical Note, May 2003, see <http://www.rsasecurity.com/rsalabs/technotes/twirl.html>.
- [16] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol", *RFC 2401*, Nov. 1998.
- [17] N. Koblitz, "Elliptic curve cryptosystems", *Mathematics of Computation*, 48:203-209, 1987.
- [18] A. Lenstra and E. Verheul, "Selecting Cryptographic Key Sizes", *Journal of Cryptology* 14 (2001) pp. 255-293, see <http://www.cryptosavvy.com/>.
- [19] MasterCard International and Visa International, "Secure Electronic Transaction Specification, Version 1.0", <http://www.setco.org/>, May 1997.
- [20] V. Miller, "Uses of elliptic curves in cryptography", *Crypto 1985*, LNCS 218: Advances in Cryptology, Springer-Verlag, 1986.
- [21] Mozilla Organization, "Netscape Security Services (NSS)", see <http://www.mozilla.org/projects/security/pki/nss>
- [22] NIST, "Advanced Encryption Standard (AES)", see <http://csrc.nist.gov/CryptoToolkit/aes>, Dec. 2001.
- [23] NIST, "Special Publication 800-57: Recommendation for Key Management. Part 1: General Guideline", Draft Jan. 2003.
- [24] OpenSSL Project, see <http://www.openssl.org/>.
- [25] B. Ramsdell, "S/MIME Version 3 Message Specification", RFC 2633, Jun. 1999.
- [26] A. Shamir and E. Tromer, "Factoring Large Numbers with the TWIRL Device", *Crypto 2003*, LNCS 2729, Springer-Verlag, Aug. 2003.
- [27] S. Shankland, "Open-source group gets Sun security gift", Sep. 2002, see <http://news.com.com/2100-1001-958679.html>.
- [28] N. Smart, "How Secure Are Elliptic Curves over Composite Extension Fields?", *EUROCRYPT 2001*, LNCS 2045, Springer-Verlag, pp. 30–39, 2001.
- [29] U.S. Dept. of Commerce/NIST, "Digital Signature Standard (DSS)", FIPS PUB 186-2, Jan. 2000.
- [30] S. A. Vanstone, "Next generation security for wireless: elliptic curve cryptography", *Computers and Security*, Vol 22, No 5, Aug. 2003.
- [31] D. Wagner, B. Schneier, "Analysis of the SSL 3.0 protocol", 2nd USENIX Workshop on Electronic Commerce, Nov. 1996.
- [32] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, S. Lehtinen "SSH Protocol Architecture", IETF Internet draft, work in progress, Jul. 2003.
- [33] Zona Research, "The Need for Speed II", Zona Market Bulletin, Issue 5, Apr. 2001.