# Dispelling the New SSL Myth

**Lori MacVittie, 2011-31-01**

*Claiming SSL is not computationally expensive is like saying gas is not expensive when you don't have to drive to work every day.*

### My car is eight years old this year. It has less than 30,000 miles on it.

Yes, you heard that right, less than 30,000 miles. I don't drive my car very often because, well, my commute is a short trip down two flights of stairs. I don't need to go very far when I do drive it's only ten miles or so round trip to the grocery store. So from my perspective, gas isn't really very expensive. I may use a tank of gas a month, which works out to … well, it's really not even worth mentioning the cost. But for someone who commutes every day – especially someone who commutes a long-distance every day – gas is expensive. It's a significant expense every month for them and they would certainly dispute my assertion that the cost of gas isn't a big deal. My youngest daughter, for example, would say gas is very expensive – but she's got a smaller pool of cash from which to buy gas so relatively speaking, we're both right.

The same is true for anyone claiming that SSL is not computationally expensive. The way in which SSL is used – the ciphers, the certificate key lengths, the scale – has a profound impact on whether or not "computationally expensive" is an accurate statement or not. And as usual, it's not just about speed – it's also about the costs associated with achieving that performance. It's about efficiency, and leveraging resources in a way that enables scalability. It's not the cost of gas alone that's problematic, it's the cost of *driving*, which also has to take into consideration factors such as insurance, maintenance, tires, parking fees and other driving-related expenses.

## MYTH: SSL is NOT COMPUTATIONALLY EXPENSIVE TODAY

### SSL is still computationally expensive. Improvements in processor speeds in some circumstances have made that expense less impactful. Circumstances are changing.

Commoditized x86 hardware can in fact handle SSL a lot better today than it ever could before –when you're using 1024-bit keys and "easy" ciphers like RC4. Under such parameters it is true that commodity hardware may perform efficiently and scale up better than ever when supporting SSL. Unfortunately for proponents of SSL-on-the-server, 1024-bit keys are no longer the preferred option and security professionals are likely well-aware that "easy" ciphers are also "easy" pickings for miscreants.

In January 2011, NIST recommendations regarding the deployment of SSL went into effect. While NIST is not a standards body can require compliance or else, they can and do force government and military compliance and have shown their influence with commercial certificate authorities. All commercial certificate authorities now issue only 2048-bit keys. This increase has a huge impact on the capacity of a server to process SSL and renders completely inaccurate the statement that SSL is not computationally expensive anymore. A typical server that could support 1500 TPS using 1024-bit keys will only support 1/5 of that (around 300 TPS) when supporting modern best practices, i.e. 2048-bit keys.

| SSL Performance | | |
|---|---|---|
| | Commodity Hardware | |
| Key Length | 32-bit | 64-bit |
| 1024 | 525 TPS | 1570 TPS |
| 2048 | 96 TPS | 273 TPS |
| 4096 | 15 TPS | 38 TPS |

Also of note is that NIST recommends ephemeral Diffie-Hellman - not RSA - for key exchange, and per TLS 1.0 specification, AES or 3DES-EDE-CBC, not RC4. These are much less "easy" ciphers than RC4 but unfortunately they are also more computationally intense, which also has an impact on overall performance.

Key length and ciphers becomes important to the performance and capacity of SSL not just during the handshaking process, but in bulk-encryption rates. It is one thing to say a standard server deployed to support SSL can handle X handshakes (connections) and quite another to simultaneously perform bulk-encryption on subsequent data responses. The size and number of those responses have a huge impact on the consumption rate of resources when performing SSL-related functions on the overall server's capacity. Larger data sets require more cryptographic attention that can drag down the rate of encryption – that means slower response times for users and higher resource consumption on servers, which decreases resources available for handshaking and server processing and cascades throughout the entire system to result in a reduction of capacity and poor performance.

Tweaked configurations, poorly crafted performance tests, and a failure to consider basic mathematical relationships may seem to indicate SSL is "not" computationally expensive yet this contradicts most experience with deploying SSL on the server. Consider this question and answer in the SSL FAQ for the Apache web server:

> **Why does my webserver have a higher load, now that it serves SSL encrypted traffic?**
>
> SSL uses strong cryptographic encryption, which necessitates a lot of number crunching. When you request a webpage via HTTPS, everything (even the images) is encrypted before it is transferred. So increased HTTPS traffic leads to load increases.

This is not myth, this is a well-understood fact – SSL requires higher computational load which translates into higher consumption of resources. That consumption of resources increases with load. Having more resources does not change the consumption of SSL, it simply means that from a mathematical point of view the consumption rates relative to the total *appear* to be different. The "amount" of resources consumed by SSL (which is really the amount of resources consumed by cryptographic operations) is proportional to the total system resources available. The additional consumption of resources from SSL is highly dependent on the type and size of data being encrypted, the load on the server from both processing SSL and application requests, and on the volume of requests.

Interestingly enough, the same improvements in capacity and performance of SSL associated with "modern" processors and architecture is also applicable to intermediate SSL-managing devices. Both their specialized hardware (if applicable) and general purpose CPUs significantly increase the capacity and performance of SSL/TLS encrypted traffic on such solutions, making their economy of scale much greater than that of server-side deployed SSL solutions.

## THE SSL-SERVER DEPLOYED DISECONOMY of SCALE

**Certainly if you have only one or even two servers supporting an application for which you want to enable SSL the costs are going to be significantly different than for an organization that may have ten or more servers comprising such a farm. It is not just the computational costs that make SSL deployed on servers problematic, it is also the associated impact on infrastructure and the cost of management.**

Reports that fail to factor in the associated performance and financial costs of maintaining valid certificates on each and every server – and the management / creation of SSL certificates for ephemeral virtual machines – are misleading. Such solutions assume a static environment and a deep pocket or perhaps less than ethical business practices. Such tactics attempt to reduce the capital expense associated with external SSL intermediaries by increasing the operational expense of purchasing and managing large numbers of SSL certificates – including having a ready store that can be used for virtual machine instances.

As the number of services for which you want to provide SSL secured communication increase and the scale of those services increases, the more costly it becomes to manage the required environment. Like IP address management in an increasingly dynamic environment, there is a diseconomy of scale that becomes evident as you attempt to scale the systems and processes involved.

### DISECONOMY of SCALE #1: CERTIFICATE MANAGEMENT

Obviously the more servers you have, the more certificates you need to deploy. The costs associated with management of those certificates – especially in dynamic environments – continues to rise and the possibility of missing an expiring certificate increase with the number of servers on which certificates are deployed. The promise of virtualization and cloud computing is to address the diseconomy of scale; the ability to provision and ready-to-function server complete with the appropriate web or application stack serving up an application for purposes of scale assumes that everything is ready. Unless you're failing to properly provision SSL certificates you cannot achieve this with a server-deployed SSL strategy. Each virtual image upon which a certificate is deployed must be pre-configured with the appropriate certificate and keys and you can't launch the same one twice. This has the result of negating the benefits of a dynamically provisioned, scalable application environment and unnecessarily increases storage requirements because images aren't small. Failure to recognize and address the management and resulting impact on other areas of infrastructure (such as storage and scalability processes) means ignoring completely the actual real-world costs of a server-deployed SSL strategy.

It is always interesting to note the inability of web servers to support SSL for multiple hosts on the same server, i.e. virtual hosts.

> **Why can't I use SSL with name-based/non-IP-based virtual hosts?**
>
> The reason is very technical, and a somewhat "chicken and egg" problem. The SSL protocol layer stays below the HTTP protocol layer and encapsulates HTTP. When an SSL connection (HTTPS) is established Apache/mod_ssl has to negotiate the SSL protocol parameters with the client. For this, mod_ssl has to consult the configuration of the virtual server (for instance it has to look for the cipher suite, the server certificate, etc.). But in order to go to the correct virtual server Apache has to know the `Host` HTTP header field. To do this, the HTTP request header has to be read. This cannot be done before the SSL handshake is finished, but the information is needed in order to complete the SSL handshake phase. Bingo!

Because an intermediary terminates the SSL session and then determines where to route the requests, a variety of architectures can be more easily supported without the hassle of configuring each and every web server – which must be bound to IP address to support SSL in a virtual host environment. This isn't just a problem for hosting/cloud computing providers, this is a common issue faced by organizations supporting different "hosts" across the domain for tracking, for routing, for architectural control. For example, api.example.com and www.example.com often end up on the same web server, but use different "hosts" for a variety of reasons. Each requires its own certificate and SSL configuration – and they must be bound to IP address – making scalability, particularly auto-scalability, more challenging and more prone to the introduction of human error. The OpEx savings in a single year from SSL certificate costs alone could easily provide an ROI justification for the CapEx of deploying an SSL device before even considering the costs associated with managing such an environment. CapEx is a onetime expense while OpEx is recurring and expensive.

### DISECONOMY of SCALE #2: CERTIFICATE/KEY SECURITY

The simplistic nature of the argument also fails to take into account the sensitive nature of keys and certificates and regulatory compliance issues that may require hardware-based storage and management of those keys regardless of where they are deployed (FIPS 140-2 level 2 and above). While there are secure and compliant HSM (Hardware Security Modules) that can be deployed on each server, this requires serious attention and an increase of management and skills to deploy. The alternative is to fail to meet compliance (not acceptable for some) or simply deploy the keys and certificates on commoditized hardware (increases the risk of theft which could lead to far more impactful breaches).

For some IT organizations to meet business requirements they will have to rely on some form of hardware-based solution for certificate and key management such as an HSM or FIPS 140-2 compliant hardware. The choices are deploy on every server (note this may become very problematic when trying to support virtual machines) or deploy on a single intermediary that can support all servers at the same time, and scale without requiring additional hardware/software support.

> The big bad wolf was still a wolf under that sheepskin. Encrypted malicious code and data is still malicious code and data. Encryption just makes it nearly impossible to see it for what it is until it's on your doorstep.

*DISECONOMY of SCALE #3: LOSS of VISIBILITY / SECURITY / AGILITY*

SSL "all the way to the server" has a profound impact on the rest of the infrastructure, too, and the scalability of services. Encrypted traffic cannot be evaluated or scanned or routed based on content by any upstream device. IDS and IPS and even so-called "deep packet inspection" devices upstream of the server cannot perform their tasks upon the traffic because it is encrypted. The solution is to deploy the certificates from every machine on the devices such that they can decrypt and re-encrypt the traffic. Obviously this introduces unacceptable amounts of latency into the exchange of data, but the alternative is to not scan or inspect the traffic, leaving the organization open to potential compromise.

It is also important to note that encrypted "bad" traffic, e.g. malicious code, malware, phishing links, etc… does not change the nature of that traffic. It's still bad, it's also now "hidden" to every piece of security infrastructure that was designed and deployed to detect and stop it.

A server-deployed SSL strategy eliminates visibility and control and the ability to rapidly address both technical and business-related concerns. Security is particularly negatively impacted. Emerging threats such as a new worm or virus for which AV scans have not yet but updated can be immediately addressed by an intelligent intermediary – whether as a long-term solution or stop-gap measure. Vulnerabilities in security protocols themselves, such as the TLS man-in-the-middle attack, can be immediately addressed by an intelligent, flexible intermediary long before the actual solutions providing the service can be patched and upgraded.

A purely technical approach to architectural decisions regarding the deployment of SSL or any other technology is simply unacceptable in an IT organization that is actively trying to support and align itself with the business. Architectural decisions of this nature can have a profound impact on the ability of IT to subsequently design, deploy and manage business-related applications and solutions and should not be made in a technical or business vacuum, without a full understanding of the ramifications.

---


- The Anatomy of an SSL Handshake [Network Computing]
- Get Ready for the Impact of 2048-bit RSA Keys [Network Computing]
- SSL handshake latency and HTTPS optimizations [semicomplete.com]
- Black Hat: PKI Hack Demonstrates Flaws in Digital Certificate Technology [DarkReading]
- SSL/TLS Strong Encryption: FAQ [apache.org]
- The Open Performance Testing Initiative
- The Order of (Network) Operations
- Congratulations! You do no nothing faster than anyone else!
- Data Center Feng Shui: SSL
- WILS: SSL TPS versus HTTP TPS over SSL
- F5 Friday: The 2048-bit Keys to the Kingdom

- TLS Man-in-the-Middle Attack Disclosed Yesterday Solved Today with Network-Side Scripting

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com