CyaSSL Additional Features

09.30.2010

http://www.yassl.com
info@yassl.com
phone: +1 206 369 4800

I. The Benefit of Stream Ciphers

Ever wondered what the difference between a block cipher and a stream cipher was? A block cipher has to be encrypted in chunks that are the block size for the cipher. For example, AES has block size of 16 bytes. So if you're encrypting a bunch of small, 2 or 3 byte, chucks back and forth, over 80% of the data is useless padding, decreasing the speed of the encryption/decryption process and needlessly wasting network bandwidth to boot. Basically block ciphers are designed for large chucks of data, have block sizes requiring padding, and use a fixed, unvarying transformation.

Stream ciphers work well for large or small chucks of data. They are suitable for smaller data sizes because no block size is required. If speed is a concern, stream ciphers are your answer, because they use a simpler transformation that typically involves an xor'd keystream. So if you need to stream media, encrypt various data sizes including small ones, or have a need for a fast cipher then stream ciphers are your best bet.

SSL uses RC4 as the default stream cipher. It's a pretty good one, though it's getting a little older. There are some interesting advancements being made in the field and nearly two years ago CyaSSL added two ciphers from the eStream project into the code base, RABBIT and HC-128. RABBIT is nearly twice as fast as RC4 and HC-128 is about 5 times as fast! So if you've ever decided not to use SSL because of speed concerns, using CyaSSL's stream ciphers should lessen or eliminate that performance doubt.

Both RABBIT and HC-128 are built by default into CyaSSL. Please see the examples or the documentation for usage. Links to these ciphers can be found below:

Stream Cipher	http://en.wikipedia.org/wiki/Stream_cipher
Block Cipher	http://en.wikipedia.org/wiki/Block_cipher
HC-128	http://www.ecrypt.eu.org/stream/p3ciphers/hc/hc128_p3.pdf
Rabbit	http://www.cryptico.com/Files/filer/rabbit_fse.pdf
RC4	http://en.wikipedia.org/wiki/Rc4

II. AES-NI Support

AES is a key encryption standard used by governments worldwide, which CyaSSL has always supported. Intel has released a new set of instructions that is a faster way to implement AES. CyaSSL is currently the first SSL library to fully support the new instruction set for production environments.

Essentially, Intel has added AES instructions at the chip level that perform the computational-intensive parts of the AES algorithm, boosting performance.

We have added the functionality to CyaSSL to allow it to call the instructions directly from the chip, instead of running the algorithm in software. This means that when you're running CyaSSL on a chipset that supports AES-NI, you can run your AES crypto 5-10 times faster!

References and further reading, ordered from general to specific are listed below. See the CyaSSL README for instructions on building CyaSSL with AES-NI.

AES (Wikipedia)	http://en.wikipedia.org/wiki/Advanced_Encryption_Standard
AES-NI (Wikipedia)	http://en.wikipedia.org/wiki/AES_instruction_set
AES-NI (Intel Software Network page)	http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni/

III. Digitally Signing and Authenticating with CyaSSL

CyaSSL is a popular tool for digitally signing applications, libraries, or files prior to loading them on embedded devices. Most desktop and server operating systems allow creation of this type of functionality through system libraries, but stripped down embedded operating systems do not. The reason that embedded RTOS environments do not include digital signature functionality is because it has historically not been a requirement for most embedded applications. In today's world of connected devices and heightened security concerns, digitally signing what is loaded onto your embedded or mobile device has become a top priority.

Examples of embedded connected devices where this requirement was not found in years past include set top boxes, DVR's, POS systems, both VoIP and mobile phones, and even automobile based computing systems. Because CyaSSL supports the key embedded and real time operating systems, encryption standards and authentication functionality, it is a natural choice for embedded systems developers to use when adding digital signature functionality.

Generally, the process for setting up code and file signing on an embedded device are as follows:

1. The embedded systems developer will generate an RSA key pair.
2. A server-side script-based tool is developed
    a. The server side tool will create a hash of the code to be loaded on the device with SHA-256 for example.
    b. The hash is then digitally signed, also called RSA private encrypt.
    c. A package is created that contains the code along with the digital signature.
3. The package is loaded on the device along with a way to get the RSA public key. The hash is re-created on the device then digitally verified (also called RSA public decrypt) against the existing digital signature.

Benefits to enabling digital signatures on your device:

1. Easily enable a secure method for allowing third parties to load files to your device.
2. Ensure against malicious files finding their way on to your device.
3. Digitally secure firmware updates
4. Ensure against firmware updates from unauthorized parties

More background on code signing:

A great article on the topic at embedded.com: http://embedded.com/design/216500493?printable=true

General information on code signing: http://en.wikipedia.org/wiki/Code_signing

IV. IPv6 Support

If you are an adopter of IPv6 and want to use an embedded SSL implementation then you may have been wondering if CyaSSL supports IPv6. The answer is yes, we do support CyaSSL running on top of IPv6. Note that our current test applications default to IPv4, so as to apply to a broader number of systems. Please see http://www.yassl.com/yaSSL/Docs_Building_CyaSSL.html --enable-ipv6 to change the test applications to IPv6.

Further information on IPv6 can be found here: http://en.wikipedia.org/wiki/IPv6.