wolfSSL **Tuning Guide**

Version 1.2
June 5, 2015

Purpose

This guide provides a reference for developers and engineers to tune and optimize the performance and memory usage of the wolfSSL embedded SSL library. It should be considered a “guide” and as such, it is an evolving work. If there is something you find missing, please let us know and we’ll be happy to add instructions or clarification to the document. One of our main goals for wolfSSL is ease of use.

Audience

This guide caters to developers or engineers interested in optimizing the performance and memory usage of the wolfSSL embedded SSL library.

Considerations

The first step in tuning wolfSSL to your environment is to document your anticipated requirements. At the highest level, design goals for SSL/TLS usually boil down to:

a. Memory Footprint (ROM)
b. Per-session Memory Usage (RAM)
c. SSL Handshake Performance
d. Data Flow Performance (bulk data transfer)
e. Desired Security Level (see Figure 1, below)

Security Level	**Level Name**	**Description**
1	**Relaxed**	You just need to keep out the casual onlookers
2	**Moderate**	You have potential attackers out there, but they are not highly motivated
3	**Corporate**	Secure against professional attacks
4	**Military Grade**	Title speaks for itself
5	**Quantum Survivable**	Can survive against quantum computing based attacks

Figure 1: Desired Security Levels

Each of these top level goals will have trade offs against the others as you evolve your design.

To get started, the key variables you need to define include:

1. Available hardware:
    a. Memory available to SSL/TLS (ROM / RAM)
    b. CPU type and clock speed
2. Required SSL/TLS protocol levels (ie: TLS 1.0, TLS 1.1, TLS 1.2, etc.)
3. Required cipher suites. If the cipher suites are not defined in your requirement, you are free to choose cipher suites that match your performance goals:
    a. Public key algorithm with key length (RSA, ECC, NTRU, PSK, etc.)
    b. Block / Stream ciphers (AES, DES, 3DES, RC4, HC-128, etc.)
    c. Hash functions (SHA, SHA2, MD5, Blake2b, etc.)
4. What side of the connection are you on: client, server, or both?
5. Client authentication?
6. Is the other side of the SSL connection defined?
    a. What SSL implementation is it using?
    b. Which SSL/TLS protocol version?
    c. What is the key length?

    d. Is it a client or server?
7. What are the maximum number of active SSL/TLS connections/sessions needed at one time?
8. What are the SSL handshake performance requirements?
9. What are the bulk data transfer performance requirements, after SSL handshake has completed?
10. Is hardware crypto available? If yes, then what ciphers are available in hardware?
11. Editors Note: For the purpose of keeping this document usable in scope, we are excluding operating system and TCP/IP stack tuning opportunities and reserving that for another document.

Having noted all of the above variables, you will see that there is much to consider, so our approach is to present three optimization recipes for reference in this guide:

1. Optimizing for **minimum footprint** size (heap, stack, static data, code)
2. Optimizing for **maximum speed**
3. Optimizing for **maximum security**

Other optimization recipes are available. Just contact us at info@wolfssl.com. Additional reference recipes that we can help with include:

1. Optimizing for large numbers of connections
2. Optimizing for particular operating systems/chipsets
3. Optimizing for particular applications
4. Optimizing for a combination of higher goals, for example maximum security with minimum footprint
5. Optimization low power consumption
6. Optimizing for fun and relaxation

Recipe #1: Minimum Footprint

Many users are on deeply embedded systems, where memory resources are tight. For those users, this section describes methods to reduce the footprint size of wolfSSL.

1. Limit supported protocol versions to only those required, for example only allowing TLS 1.2 connections.
2. Remove unnecessary library features at compile time - section 2.4.1 of the wolfSSL Manual.
3. Choose a limited set of cipher suites:
    a. Memory usage difference between RSA, ECC, PSK
    b. Choose smaller key sizes - section 4.3 of the wolfSSL manual.
4. Take advantage of hardware crypto if available - section 4.4 of the wolfSSL manual.
5. Use compiler and toolchain optimizations.
6. Decrease maximum SSL record size if you control both ends of the connection.

Recipe #2: Maximum Speed

Adding SSL/TLS to a connection will always result in an inevitable reduction of performance. Our goal is to make that PERFORMANCE decrease as small as possible. This section describes ways to speed up wolfSSL, both during and after the handshake.

There are two main areas of concern regarding performance:

1. SSL/TLS handshake speed
2. Data flow rate (bulk data transfer, after the SSL handshake)

When optimizing SSL handshake performance, items to consider include:

1. Use a faster math library (big integer vs. fastmath).
2. Take advantage of hardware crypto if available - section 4.4 of the wolfSSL manual.
3. Key size - Chapter 4 of the wolfSSL manual.
4. Key type (RSA vs ECC for example)
5. Trade off between handshake speed and security level (such as client/server cert verification - section 4.8 of the wolfSSL manual).
6. Consider using PSK (pre-shared keys) - section 4.7 of the wolfSSL manual.

Maximum data flow rate in a streaming media environment for example, such as a video game, VoIP application, or cloud infrastructure, cipher suite choice is critical. In this recipe, there are many options depending on hardware environment and number of connections. To simplify the recipe to make it usable, we will focus on a single connection environment running on a typical cloud based server.

When optimizing for maximum data flow rate, items to consider include:

1. Choose cipher suites to prioritize faster algorithms over slower ones: Stream Ciphers, Rabbit, HC-128.
2. Take advantage of better compiler optimization. (But I am not sure if this is a user's practical option)
3. Take advantage of hardware crypto if available.

Recipe #3: Maximum Security

The security of a SSL/TLS connection should be of high concern, since having a secure communication channel is the primary reason for adding SSL/TLS to a project.

As with all cryptography-based protocols, SSL/TLS security recommendations can change as new attacks and vulnerabilities are discovered and released. Optimizing for maximum security can have negative effects on both memory usage and performance, depending on configuration.

1. Cipher suite choices based on the best currently available information.

2. Key size choices based on the best currently available information.
3. Other considerations…

As you can see from the basic recipes above, optimizing SSL is a complex multivariate problem that depends heavily on a wide range of assumptions about your initial environment. We are here to help. The wolfSSL team has successfully guided a vast number of our customers through these choices. We can support you in an entire spectrum of ways, from the simple question and answer process of typical commercial support, to short term professional design consulting, up to managing the entire implementation of your SSL project.

Additional Information

wolfSSL Porting Guide: https://www.wolfssl.com/wolfSSL/Docs-wolfssl-porting-guide.html
Download wolfSSL: https://wolfssl.com/wolfSSL/download/downloadForm.php
wolfSSL Support Forums: http://www.wolfssl.com/forums
Product Support Email: support@wolfssl.com

Contact us at info@wolfSSL.com or call us at +1 425 245-8247 for additional information. We can make our kickstart and optimization consulting available to walk you through the details.