MatrixSSL Porting Guide

This document discusses porting MatrixSSL to additional operating systems and platforms. MatrixSSL has been ported to over 20 CPU/OS combinations and runs on platforms from 8 to 64 bits, big and little endian. Compatibility testing is done against multiple independent SSL implementations.

External API Requirements

Memory Allocation	
malloc() free()	MatrixSSL includes support for deterministic memory allocation. This ensures that all MatrixSSL allocations take place within a single block of memory per session, preventing leaks and fragmentation, and greatly reducing the possibility of buffer overruns. MatrixSSL can be compiled to support one memory block per session, or operate entirely out of static memory, with no system memory allocations.
Memory Operations	
memcmp() memcpy() memset() strstr() strlen()	These functions can easily be replaced with custom implementations, should they not be present in the standard platform library.
File Access	
stat() fopen() fclose() fgets()	File access functions are used only to read certificate and private key files. If a filesystem is not supported, the matrixSslReadKeysMem() API, defined in matrixInternal.h can be used to parse certificates and keys from memory buffers, allowing operation without a filesystem. Disable the USE_FILE_SYSTEM define in matrixConfig.h to disable the file system calls on systems that do not support them.
Time	
time()	The time() routine is used to check expiration of the session cache, and to provide the first four bytes of the ServerRandom value. Any known-scale time value such as clock ticks since startup can be used for the first value. The ServerRandom value should have a monotonically increasing value that is preserved across machine restarts to help prevent replay based attacks. Intel platforms use a processor dependant high resolution timer rather than the time() system call.
Debugging	
printf()	These functions are used only for debugging and can easily be

abort()	replaced by other mechanisms of error reporting.
Multithreading	
Mutex APIs	Mutex locks are used only to protect the session cache if multiple threads have simultaneous sessions open. Systems without mutex support typically also lack threading support so these functions should not need to be ported. Disabling the USE_MULTITHREADING define in matrixConfig.h will disable all mutex code. The abstraction layer for thread synchronization is in the OS specific directories under matrixssl/src/os.
Forked Processing	
	Applications using fork() to handle new connections are common on Unix based platforms. Because the MatrixSSL session cache is located in the process data space, a forked process will not be able to update the master session cache, thereby preventing future sessions from being able to take advantage of this speed improvement. In order to support session resumption in forked servers, a file or shared memory based session cache must be implemented.
Networking	
Sockets APIs	MatrixSSL operates independently from the network layer. Existing socket code tuned to your platform can continue to send and receive data that is encoded and decoded by MatrixSSL inline. MatrixSSL includes an example POSIX sockets implementation.
Entropy Gathering	
Random Data	In order to create a secure SSL connection, **it is critical to have a source of good random data on each platform**. Ports of MatrixSSL to any platform **must** support the gathering of cryptographically random entropy bytes. Operating systems typically provide this data through kernel level timers, random keyboard events, etc. Embedded systems are much more predictable in terms of user and kernel timings, so drivers for hardware based entropy are usually used in this case. The built in entropy gathering API, sslGetEntropy() is implemented in the OS specific directories under matrixssl/src/os.