MatrixSSL 3.1.X Open Source Release Notes

PeerSec Networks, Inc. 410 Broadway Ave E. #205 Seattle, WA 98102 **T** 425.646.7850 **F** 206.501.4366
support@peersec.com www.peersec.com

Overview

This document is an aggregation of the release notes from MatrixSSL versions 3.1.0 to 3.1.4. MatrixSSL 3.2 is the version that follows 3.1.4.

Who Is This Document For?

- Software developers that are securing applications with MatrixSSL
- Software developers upgrading to MatrixSSL 3.2 from a previous version

MatrixSSL 3.1.0

MatrixSSL 3.1 is the result of starting with the widely deployed PeerSec SSL/TLS engine and optimizing for reduced memory usage, cryptographic speed, and ease-of-integration. The following list is an overview of the most significant changes to this major revision.

Enhancements to Features and Functionality

TLS 1.0 Protocol Support

Beginning in MatrixSSL 3.1 the TLS 1.0 protocol and AES cipher are now available in open source releases.

Improved API

It is now easier than ever to integrate SSL into your application. MatrixSSL has always provided SSL integration to applications at a data buffer level to guarantee support for any given transport mechanism. Previous versions, however, left the management of these data buffers in the hands of the integrator. The new MatrixSSL 3.1 API incorporates size-optimized buffer management so the user is left only with the task of determining when data needs to be read or written, while still maintaing a transport-neutral, zero buffer copy API.

Faster and smaller RSA cryptography

The public key cryptography operations required for RSA mathematics are the primary contributors to high water memory and CPU resources during the SSL handshake. MatrixSSL 3.1 includes specific optimizations that have resulted in major improvements to both speed and memory usage during public cryptography. These substantial memory savings and performance improvements allow MatrixSSL to be used on an even larger number of embedded platforms. The entire SSL handshake, including network buffers can now be completed in as little as 10KB of RAM, with a post-handshake dynamic memory footprint of less than 3KB.

File and functional re-organization

The MatrixSSL 3.1 source code package has been organized to better reflect the individual functional areas. The *core* and *crypto* modules are now clear building blocks on which MatrixSSL relies and each module has an API and Configuration header to manage optional features and functionality.

Supported Client and Server Applications

New client and server examples are now provided as a starting off point for customer integration or new application development. The client application is an example of a simple, blocking sockets API HTTPS client that prints the response to a HTTP GET request. The server example demonstrates a non-blocking HTTPS server that handles multiple connections and session timeouts. The MatrixSSL API usage for both applications is very similar, and should help clarify how to integrate MatrixSSL with other applications.

Self-Test Application

A SSL/TLS protocol test application is now included in the package so that new ports of MatrixSSL can quickly be verified and functionally tested, even before integration with a sockets layer. The application creates virtual SSL connections within a single process using memory buffers as the transport layer. Each supported cipher suite and handshake mode are validated.

Additional Project Formats

Project files for the MatrixSSL library, example and test applications are now provided for Microsoft Visual Studio Express Edition, Apple Xcode and standard GNU make. Projects for the Eclipse IDE can be directly imported from GNU Makefile.

MatrixSSL 3.1.1

This section highlights the differences between version 3.1 and 3.1.1

Enhancements to Features and Functionality

Secure Renegotiations

In late 2009 an exploit in the SSL renegotiation protocol was discovered. A fix for the exploit has been published in RFC 5746 and version 3.1.1 includes the implementation. Re-handshaking is now controlled by three new compile-time defines:

ENABLE_SECURE_REHANDSHAKES

This define can be found in *matrixsslConfig.h* and is enabled by default. Enabling this define will activate the RFC 5746 implementation and allow MatrixSSL applications to securely re-handshake with peers that have also implemented it.

REQUIRE_SECURE_ REHANDSHAKES

This define can be found in *matrixsslConfig.h* and is disabled by default. Enabling this define will only allow the MatrixSSL application to connect with SSL peers that have implemented RFC 5746.

ENABLE_INSECURE_ REHANDSHAKES

This define can be found in matrixsslConfig.h and is disabled by default. Enabling this define will enable legacy re-handshaking support and is NOT RECOMMENDED.

CLIENT_HELLO extension support

Support for adding extensions to CLIENT_HELLO messages is now included in the open source version of MatrixSSL. More information on hello extensions can be found in RFC 3546. To support this feature an existing API has changed prototype and three new APIs have been introduced:

matrixSslNewClientSession

This function prototype has changed. A new function callback parameter has been added to this routine that will be invoked while clients are parsing SERVER_HELLO extensions.

matrixSslNewHelloExtension, matrixSslLoadHelloExtension, and matrixSslDeleteHelloExtension

This family of APIs is now available to client application integrators to append CLIENT_HELLO extensions to the handshake protocol. See the API documentation for details on these new function.

Client cipher suites on re-handshakes

Clients will now resend the full list of supported cipher suites on server-initiated re-handshakes. In previous versions, upon receiving a HELLO_REQUEST from a connected server, the client would only supply the cipher suite that was currently negotiated in the CLIENT_HELLO.

Makefile auto detects 32 and 64 bit platforms

The top level Makefile now detects whether 32 or 64 bit Linux or Mac OS X is running, and sets some defines appropriately to optimize performance for 64 bit platforms. Previously these defines (specifically `PSTM_64BIT`) had to be defined manually on 64 bit platforms. Also, the stderr output of the archive command `'ar'` is now suppressed to hide spurious warnings about empty object files when features are disabled, producing empty objects.

New documents: Migration to 3.1 and OS Porting Guide

Two new documents are included with the package:

- A guide to migrating from MatrixSSL 1.x and 2.x APIS to the current 3.x api.
- A guide to porting to a new, unsupported OS. Includes information on minimal system call requirements, random number generation, etc.

Public API Changes

New matrixSslNewClientSession prototype

An additional parameter has been added to this routine to improve hello extension support. Clients can now register a callback that will be invoked during the SSL handshakes to parse any SERVER_HELLO extensions that might be sent by the server.

USE_INT64 renamed to HAVE_NATIVE_INT64

This define in *coreConfig.h* has been renamed for clarity.

Bug Fixes

Changing Cipher Suites on Re-handshake

A handshaking failure was discovered during re-handshake testing in some cases where the underlying cipher suite was changing, resulting in an invalid SSL Alert and connection close. This has been fixed as part of the overall handshake protocol change.

Default size for pstm_digit

The default 32-bit platform now explicitly sets the `psmt_digit` type as a 32-bit unsigned integer rather than an `unsigned long`. This fixes a compile issue witbh running with 32-bit math on a 64-bit platform.

MatrixSSL 3.1.2

This section highlights the differences between version 3.1.1 and 3.1.2

Enhancements to Features and Functionality

Explicit API support for processing multi-record data buffers

The 3.1.1 API set did not include a documented mechanism for processing buffers in which multiple application data records are concatenated in a single 'recv' buffer. This is not an uncommon scenario and users are strongly encouraged to update to this latest MatrixSSL version and implement the new `matrixSslProcessedData` function in their applications. Details can be found in the updated API documentation included in this package.

MatrixSSL version defines added

A *version.h* file has been added that includes defines for the MatrixSSL major, minor, and patch build version. The new header is included by matrixsslApi.h and defines the full version and the individual components. For example:

```
#define MATRIXSSL_VERSION          "3.1.2-OPEN"
#define MATRIXSSL_VERSION_MAJOR    3
#define MATRIXSSL_VERSION_MINOR    1
#define MATRIXSSL_VERSION_PATCH    2
#define MATRIXSSL_VERSION_CODE     "OPEN"
```

The sslTest application includes a timing mode

The sslTest application can now be built to measure the connection speeds for clients and servers for the various cipher suites.

Improvements to HTTP parsing in example application code

The server and client example applications now identify partial and multi-record HTTP records.

Public API Changes

New matrixSslProcessedData prototype and return codes

To support the processing of multi-record data buffers, the `matrixSslProcessedData` function prototype and return codes have changed. The new function has two additional parameters that are used to return the next decoded record in the buffer. The return codes for this function have been expanded to inform the user how that second record should be handled.

Please see the API documentation and code examples for detailed information.

Bug Fixes

Fixed return codes where unsigned data types were assigned negative values

The functions `psRsaDecryptPriv`, `psRsaDecryptPub`, and `matrixSslDecode` are now consistent in their use of unsigned vs. signed data types.

MatrixSSL 3.1.3

This section highlights the differences between version 3.1.2 and 3.1.3

Enhancements to Features and Functionality

New server-side configuration option to decrease binary executable size

Servers may now disable a new `USE_CERT_PARSE` define in *crytpoConfig.h* to exclude a relatively large portion of the *x509.c* source code.

Previous versions of MatrixSSL would always pass the server certificate through an X.509 parse phase during initialization. This allowed the library to confirm the format of the certificate and perform algorithm tests based on the chosen cipher suite. However, these tests were in place primarily to prevent user error so if `USE_CERT_PARSE` is disabled, the user must be confident the certificate material is valid for the cipher suites that have been enabled in *matrixsslConfig.h*

New Pseudo-Random Number Generation algorithms

An implementation of Yarrow is now included in the MatrixSSL source code package. Random numbers are now retrieved through Yarrow by default. An entropy source and implementation of `psGetEntropy` is still required for each platform.

Windows project files updated to Microsoft Visual C++ 2010 Express

Previous versions used the 2008 Express Edition of Visual C++

Public API Changes

New members in x509DNattributes_t structure

The Distinguished Name attributes in X.509 certificates such as Common Name, Organization, and Country are now accompanied by the explicit ASN.1 data type and length. Previous versions of MatrixSSL attempted to treat these fields as NULL terminated strings using single byte characters. In order to support a larger variety of certificate formats the Type and Len fields have been added so the user will have all the needed information to interpret certificate information that is passed into the certificate callback routine.

New x509DNattributes_t members.

```
short   countryType;
short   countryLen;
short   stateType;
short   stateLen;
short   localityType;
short   localityLen;
short   organizationType;
short   organizationLen;
short   orgUnitType;
short   orgUnitLen;
short   commonNameType;
short   commonNameLen;
```

Type members will be one of the following:

```
ASN_PRINTABLESTRING
ASN_UTF8STRING
ASN_IA5STRING
ASN_T61STRING
ASN_BMPSTRING
```

Bug Fixes

Error return code fixed for matrixSslReceivedData

One code path through `matrixSslReceivedData` was performing an 'unsigned char' typecast on a potentially negative return code which converted it to a positive value. This resulted in an undocumented and ambiguous return code. The typecast has been removed and all error cases now return negative values as documented.

MatrixSSL 3.1.4

This section highlights the differences between version 3.1.3 and 3.1.4

Enhancements to Features and Functionality

Primary crypto algorithms now have configuration options for size vs. speed tradeoffs

Previous versions of MatrixSSL had an undocumented compile time define (`SMALL_CODE`) that influenced the binary code size of some symmetric cipher algorithms. Each algorithm that used this define has now been given its own define to control whether the user wants to build the library for faster algorithm support at the cost of an increased binary code size. The size vs. speed tradeoff is platform dependent but, in general, the speed improvements will be about 5%-10% at the cost of 10-20KB for each algorithm. The default, in each case, is that these defines are disabled in *cryptoConfig.h* to compile in favor of smallest binary footprint.

Define	Notes
`PS_AES_IMPROVE_PERF_INCREASE_CODESIZE`	Enable to improve AES performance at the cost of a larger binary footprint. Speed vs size tradeoffs are platform dependent.
`PS_3DES_IMPROVE_PERF_INCREASE_CODESIZE`	Enable to improve 3DES performance at the cost of a larger binary footprint. Speed vs size tradeoffs are platform dependent.
`PS_MD5_IMPROVE_PERF_INCREASE_CODESIZE`	Enable to improve MD5 performance at the cost of a larger binary footprint. Speed vs size tradeoffs are platform dependent.

Define	Notes
PS_SHA1_IMPROVE_PERF_INCREASE_CODESIZE	Enable to improve SHA1 performance at the cost of a larger binary footprint. Speed vs size tradeoffs are platform dependent.

RSA algorithm now has configuration option for memory usage vs. speed tradeoff

A pair of defines have been added to determine whether the RSA algorithm should be compiled for smaller RAM usage or faster performance. The default is to compile for smaller RAM usage.

Define	Notes
PS_PUBKEY_OPTIMIZE_FOR_SMALLER_RAM	The memory savings for optimizing for ram is around 50%
PS_PUBKEY_OPTIMIZE_FOR_FASTER_SPEED	The speed gain for optimizing for speed is around 5%

Servers can now disable specific cipher suites at runtime

Cipher suites that have been compiled into the library can now be programatically disabled (and re-enabled) on a per-session basis. This is useful for servers that wish to limit the supported ciphers suites for a specific connecting client. A new API, `matrixSslSetCipherSuiteEnabledStatus`, has been added to support this functionality. Please see the MatrixSSL API documentation for detailed information on this new feature.

An Xcode project for iPhone development is now included

In the *apps/iphone* directory the user can now find a Mac Xcode project for developing SSL/TLS client applications for the iPhone.

Server compatibility with Chrome browsers that use “false start”

The Google Chrome browser has introduced a new protocol mechanism called “false start” that is incompatible with strict TLS implementations that do not allow application data exchange before the handshake protocol is complete. Enabling `ENABLE_FALSE_START` in *matrixsslConfig.h* will allow newer versions of the Chrome browser to connect with MatrixSSL servers.

A new explicit int16 data type has been added

The *osdep.h* file now includes a typedef for a 16-bit integer type called `int16`. The initial internal use of this new data type can be found in the *pstm.c* math function to help improve performance on some platforms.

Public API Changes

Compile-time define for file system support has been renamed

The `USE_FILE_SYSTEM` define has been renamed to include a `PS_` prefix so that it is now `PS_USE_FILE_SYSTEM`. In addition, this define is no longer present in the *coreConfig.h* header file. It should be included in the platform build environment as a compile-time define if file system support is needed.

Return types changed for osdep.c Open and Close routines

The platform interface functions implemented in osdep.c have undergone prototype changes.

New 3.1.4 Prototype	Old 3.1.3 Prototype
`int osdepTimeOpen(void)`	`int32 osdepTimeOpen(void)`
`void osdepTimeClose(void)`	`int32 osdepTimeClose(void)`
`int osdepEntropyOpen(void)`	`int32 osdepEntropyOpen(void)`
`void osdepEntropyClose(void)`	`int32 osdepEntropyClose(void)`
`int osdepTraceOpen(void)`	`int32 osdepTraceOpen(void)`
`void osdepTraceClose(void)`	`int32 osdepTraceClose(void)`
`int osdepMutexOpen(void)`	`int32 osdepMutexOpen(void)`
`void osdepMutexClose(void)`	`int32 osdepMutexClose(void)`