



# MatrixSSL with external cryptographic modules

---

Electronic versions are uncontrolled unless directly accessed from the QA Document Control system.

Printed version are uncontrolled except when stamped with 'VALID COPY' in red.

External release of this document may require a NDA.

© INSIDE Secure - 2019 - All rights reserved

# TABLE OF CONTENTS

<b>1</b>	<b>INTRODUCTION .....</b>	<b>3</b>
<b>2</b>	<b>EXTERNAL CLIENT AUTHENTICATION .....</b>	<b>4</b>
2.1	Description .....	4
2.2	Configuration.....	5
2.3	Using the feature in client applications .....	6
2.4	API REFERENCE .....	7
2.4.1	matrixSslNeedCvSignature.....	7
2.4.2	matrixSslGetHSMessagesHash.....	8
2.4.3	matrixSslSetCvSignature .....	9
2.4.4	psExt API.....	9
<b>3</b>	<b>INTEGRATIONS WITH OTHER MODULES .....</b>	<b>10</b>

# 1 INTRODUCTION

INSIDE Secure MatrixSSL is a highly optimized SSL/TLS stack that is designed with compact code paths, efficient memory usage and with support for asynchronous cryptography and network integration.

This document describes the available MatrixSSL integrations with external cryptographic modules. The external client authentication feature (included in *MatrixSSL Commercial Edition*) is documented in detail. Other external module integrations also exist; information about those is available from INSIDE Secure on request. This manual assumes the reader knows how to setup standard MatrixSSL client authentication.

## 2 EXTERNAL CLIENT AUTHENTICATION

### 2.1 Description

MatrixSSL's external client authentication feature allows the client-side private key operation in TLS client authentication—the signing of the handshake\_messages hash in the CertificateVerify handshake message—to be offloaded from MatrixSSL to an external module. The offloaded operation will be handled asynchronously; the handshake is paused until the signature is ready.

The purpose of the CertificateVerify message is to prove that the client is in possession of the private key pair of the public key contained in the client's certificate. With the external client authentication feature, the private key can remain confined to the external module and the signature can be computed securely within the module boundary.

MatrixSSL provides an API for fetching the handshake\_messages hash together with its length from MatrixSSL and for sending back the computed signature. The client program will communicate with MatrixSSL using the above mentioned API and will be responsible for operating the external module.

An example module (`ext/exampleExtCvSigModule.c`) and an integration with the MatrixSSL demo client application (`apps/ssl/client.c`) is provided. The interface between the client and the external module is not fixed; however, MatrixSSL provides an example interface (`matrixssl/psExt.h`), which is used by the example client integration.

## 2.2 Configuration

To use the external client authentication feature, the following preprocessor defines must be enabled:

- `USE_CLIENT_SIDE_SSL` (`matrixsslConfig.h`)
- `USE_CLIENT_AUTH` (`matrixsslConfig.h`)
- `USE_EXT_CERTIFICATE_VERIFY_SIGNING` (`matrixsslConfig.h`)
- `USE_X509` (`cryptoConfig.h`)
- `USE_CERT_PARSE` (`cryptoConfig.h`)

To enable the feature at run-time for the next client session, set the `useExtCvSigOp` field in the session options struct (`sslSessOpts_t`) to 1 before the struct is passed to `matrixSslNewClientSession`.

To test the feature with the example external module and the MatrixSSL demo client program integration, the following additional define is needed:

- `USE_EXT_EXAMPLE_MODULE` (`matrixsslConfig.h`).

Note that the external client authentication feature only supports the TLS 1.0, 1.1 and 1.2 protocols. The DTLS protocol is not supported when using external client authentication; `matrixSslNewClientSession` will return an error code if both DTLS and external client authentication flags are set in the session options.

## 2.3 Using the feature in client applications

This section describes how the flow of processing must be changed in the client program using MatrixSSL when the external client authentication feature is used. This section assumes MatrixSSL has been configured as described in the "Configuration" section.

When external client authentication is desired, the `extCvSigOp` field must be set to 1 in the session options provided to `matrixsslNewClientSession`.

Whenever `matrixSslReceivedData` returns `PS_PENDING`, the client must call `matrixSslNeedCvSignature`; if the latter returns `PS_TRUE`, the handshake has been paused to give the client a chance to ask the external module to compute the signature of the `handshake_messages` hash.

The `matrixSslGetHSMessagesHash` function should then be used to fetch the hash to sign as well as the hash length.

Once the external module has finished computing the signature, the signature must be passed onwards to MatrixSSL with `matrixSslSetCvSignature`. The format of the signature is defined in the documentation for `matrixSslSetCvSignature`.

Next, it is necessary to call `matrixSslReceivedData` again to encode the response flight containing the `CertificateVerify` message. The encoded flight must then be sent over-the-wire as usual.

For some more details, please refer to the example integration in the MatrixSSL demo client application.

## 2.4 API REFERENCE

### 2.4.1 matrixSslNeedCvSignature

```
PSPUBLIC int32_t matrixSslNeedCvSignature(ssl_t *ssl);
```

Check whether an external signature for the CertificateVerify message is needed.

When the SSL state machine is in the pending state (`matrixSslReceivedData` has returned `PS_PENDING`), this function can be used to check whether the pending operation is the signing of the handshake\_messages hash for the CertificateVerify handshake message, using the client's private key.

If this function returns `PS_TRUE`, the handshake\_messages hash should be fetched with `matrixSslGetHSMessagesHash`, signed with the client's private key and copied to MatrixSSL using `matrixSslSetCvSignature`.

Parameter	Input/Output	Description
ssl	input	Pointer to the SSL session struct

Return Value	Description
PS_TRUE	The SSL state machine is waiting for the CertificateVerify signature.
PS_FALSE	The SSL state machine is not in the pending state or the pending operation is not the CertificateVerify signature.

## 2.4.2 matrixSslGetHSMessagesHash

```
PSPUBLIC int32_t matrixSslGetHSMessagesHash(ssl_t *ssl, unsigned char *hash,  
size_t *hash_len);
```

Fetch the handshake\_messages hash.

This function will fetch the hash of all handshake messages seen so far until the CertificateVerify message. The signature of this hash is to be included in the CertificateVerify message.

This function will return the raw digest; it will not return a DigestInfo structure.

Parameter	Input/Output	Description
ssl	input	Pointer to the SSL session struct
hash	input/output	Pointer to a buffer where the handshake_messages hash will be copied.
hash_len	input/output	(In:) length of the hash buffer, (Out:) length of the handshake_messages hash.

Return Value	Description
PS_SUCCESS	The operation was successful.
PS_OUTPUT_LENGTH	The output buffer is too small. The function should be called again with a larger output buffer.
PS_FAILURE	The SSL state machine is in incorrect state



### 2.4.3 matrixSslSetCvSignature

```
PSPUBLIC int32_t matrixSslSetCvSignature(ssl_t *ssl, const unsigned char* sig,
const size_t sig_len);
```

Assign the signature of the handshake\_messages hash to the CertificateVerify message.

When RSA is used as the signature algorithm, the signature scheme to use depends on the TLS protocol version. For TLS 1.2 (RFC 5246), the RSA signature scheme must be RSASSA-PKCS1-v1\_5 (RFC 3447). For TLS <1.2 (RFC 4346), PKCS #1 RSA Encryption with block type 1 encoding must be used. Note that the RSASSA-PKCS1-v1\_5 scheme requires the hash value to be wrapped within a DigestInfo structure and the signature is computed over the DigestInfo. To determine which TLS version has been negotiated for the current handshake, hash length returned by `matrixSslGetHSMessagesHash` can be used: hash length 36 indicates TLS <1.2, other hash lengths indicate TLS 1.2.

When ECDSA is used as the signature algorithm, the signature must be computed according to ANS X9.62 / RFC 4492.

Parameter	Input/Output	Description
ssl	input	Pointer to the SSL session struct
sig	input	The signature of the handshake_messages hash.
sig_len	input	The length of the signature..

Return Value	Description
PS_SUCCESS	The operation was successful.
PS_FAILURE	The SSL state machine is in incorrect state.
PS_MEM_FAIL	Out of memory

### 2.4.4 psExt API

The psExt example API for facilitating communication between the client program and the external module is described in the header file `matrixssl/psExt.h`. An example implementation of this API can be found in `ext/psext-example/exampleExtCvSigModule.c`. This API is provided both as an example and as a convenience. If the example module driver implements this API, it is straightforward to test external client authentication using that module with the MatrixSSL demo client application (`apps/ssl/client.c`).

### 3 INTEGRATIONS WITH OTHER MODULES

Information on MatrixSSL integrations with other cryptographic modules and hardware, such as Intel QuickAssist, Tiler MiCa and VaultSSL is available from INSIDE Secure on request.