MatrixSSL Elliptic Curve Cipher Suites

TABLE OF CONTENTS

1 ELLIPTIC CURVE CRYPTOGRAPHY

This document describes the use of Elliptic Curve Cryptography within the TLS protocol and how to utilize ECC cipher suites within MatrixSSL server and client applications.

ECC is a public key cryptographic algorithm that competes with RSA and Diffie-Hellman to perform key exchange and authentication. Key exchange is performed by ECC using a Diffie-Hellman variant and is abbreviated as ECDH. Authentication is performed by ECC using a Digital Signature Algorithm variant and is abbreviated as ECDSA. The appeal to some security implementers is that the mathematical properties of elliptic curves enable an equivalent strength security to RSA and DH while having smaller key sizes. This table can often be found in comparisons between the algorithms.

Equivalent Strength of Key Sizes (bits)

ECC	RSA/DH
163	1024
233	2048
283	3072
409	7680
571	15360

In authentication performance metrics head-to-head with RSA using the key sizes in the above table, the smaller key sizes of ECDSA translate into must faster key generation and slightly faster signature creation. However, RSA is much faster performing a signature validation operation.

In key exchange performance metrics head-to-head with DH, ECDH is always much higher performance. Guidelines on when to use ECC in TLS are given in the sections to follow.

The ECC algorithm must take into account the specific curve from which the keys were derived. MatrixSSL supports these NIST-recommended named prime field curves and Brainpool curves.

secp192r1
secp224r1
secp256r1
secp384r1
secp521r1
brainpool256r1
brainpool384r1
brainpool512r1

These curve restrictions apply to both the EC keys within the certificate material and for the ECDHE key generation.

2 ECC CIPHER SUITES

There are four different ECC cipher suite types available in the TLS protocol. They vary according to the cryptographic algorithms that will be used for key exchange and authentication (digital signature). Key exchange will always be ECDH but can either be ephemeral (ECDHE) or fixed DH. Ephemeral mode requires that new public key pairs be generated each connection so they are typically much slower than fixed mode. Authentication can either be RSA or ECDSA. The choice of cipher suite will determine what types of certificate and keys must be loaded at application initialization. Each cipher suite type is outlined here.

2.1 Variations

2.1.1 ECDHE_ECDSA

Ephemeral ECDH key exchange with ECDSA signatures. These cipher suite types require ECC keys for the server as well as the signing Certificate Authority.

2.1.2 ECDHE_RSA

Ephemeral ECDH key exchange with RSA signatures. Because the key exchange is ephemeral these cipher suite types allow the user to use existing RSA keys and certificates on both the clients and servers.

2.1.3 ECDH_ECDSA

Fixed ECDH with ECDSA-signed certificates. These cipher suite types require ECC keys for the server as well as the signing Certificate Authority.

2.1.4 ECDH_RSA

Fixed ECDH with RSA-signed certificates require both types of public keys to be used. These cipher suite types require ECC keys for the server certificate but an existing RSA Certificate Authority will have used its RSA key to sign that certificate. The intent of this cipher suite is to enable existing trusted RSA CAs to remain in place while allowing servers to use the faster signature creation of ECC.

2.2 Support in MatrixSSL

The define `USE_ECC` must be enabled in the *cryptoConfig.h* header file to compile in Elliptic Curve cryptography support.

The user must also enable each of the ECC cipher suites that are desired. These defines are also listed in the *matrixsslConfig.h* file and are disabled by default. Below is a representative list of the available cipher suites. The full list can be found in the header file.

```
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA

TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA

TLS_ECDH_RSA_WITH_AES_256_CBC_SHA
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
```

3 WHEN ECC CIPHER SUITES ARE A GOOD ALTERNATIVE

Assuming an equivalent key strength is used, here are some basic guidelines of what happens to performance when ECC ciphers suites are used instead of the traditional RSA suites.

1. If the application typically negotiates to an ephemeral DHE_RSA based suite without client authentication, both servers and clients will greatly benefit in switching to a ECDHE suite (either ECDHE_RSA or ECDHE_ECDSA) to take advantage of the much faster key generation.

2. If the application typically negotiates to a standard RSA suite without client authentication (RSA key exchange and authentication), servers will likely benefit by switching to an ECDH_ECDSA suite but the client will suffer decreased performance due to the slower signature validation.

3. If the application typically engages in client authentication handshakes a server will suffer greatly decreased performance due to the two signature validations during the CERTIFIATE and CERTIFICATE_VERIFY message parsing.

4 API

After enabling ECC cipher suites, the only API difference from the standard library is to use the EC specific certificate and key loading functions documented here. They are analogous to `matrixSslLoadRsaKeys` and `matrixSslLoadRsaKeysMem`.

4.1 matrixSslLoadEcKeys

```
int32 matrixSslLoadEcKeys(sslKeys_t *keys, const char *certFile,
         const char *privFile, const char *privPass,
         const char *trustedCAFiles);
```

Parameter	Input/Output	Description
keys	input/output	Allocated key structure returned from a previous call to `matrixSslNewKeys`. Will become input to `matrixSslNewClientSession` or `matrixSslNewServerSession` to associate key material with a SSL session.
certFile	input	The fully qualified filename(s) of the PEM formatted X.509 identity certificate for this SSL peer. For in-memory support, see `matrixSslLoadEcKeysMem` This parameter is always relevant to servers. Clients will want to supply an identity certificate and private key if supporting client authentication. `NULL` otherwise.
privFile	input	The fully qualified filename of the PEM formatted private EC key that was used to sign `certFile`. Supported formats are PKCS# 8 or "SEC1: Elliptical Curve Cryptography" at www.secg.org. This parameter is always relevant to servers. Clients will want to supply an identity certificate and private key if supporting client authentication. `NULL` otherwise.
privPass	input	The plaintext password used to encrypt the private key file. `NULL` if private key file is not password protected or unused. MatrixSSL supports the MD5 PKCS#5 2.0 PBKDF1 password standard.
trustedCAFiles	input	The fully qualified filename(s) of the trusted root certificates (Certificate Authorities) for this SSL peer. This parameter is always relevant to clients. Servers will want to supply a CA if requesting client authentication. `NULL` otherwise.

Return Value	Test	Description
PS_SUCCESS	0	Success. All input files parsed and the keys parameter is available for use in session creation
PS_CERT_AUTH_FAIL	< 0	Failure. Certificate or chain did not self-authenticate or private key could not authenticate certificate
PS_PLATFORM_FAIL	< 0	Failure. Error locating or opening an input file
PS_ARG_FAIL	< 0	Failure. Bad input function parameter
PS_MEM_FAIL	< 0	Failure. Internal memory allocation failure
PS_PARSE_FAIL	< 0	Failure. Error parsing certificate or private key buffer
PS_FAILURE	< 0	Failure. Password protected decoding failed. Likey incorrect password provided
PS_UNSUPPORTED_FAIL	< 0	Failure. Unsupported key algorithm in certificate material

Servers and Clients

This function is called to load the ECC certificates and private key files from disk that are needed for SSL client-server authentication. The key material is loaded into the `keys` parameter for input into the subsequent session creation APIs `matrixSslNewClientSession` or `matrixSslNewServerSession`. This API can be called at most once for a given `sslKeys_t` parameter.

A standard SSL connection performs one-way authentication (client authenticates server) so the parameters to this function are specific to the client/server role of the application. The `certFile`, `privFile`, and `privPass` parameters are server specific and should identify the certificate and private key file for that server. The `certFile` and `privFile` parameters represent the two halves of the public key so they must both be non-NULL values if either is used.

The `trustedCAFiles` parameter is client specific and should identify the trusted root certificates that will be used to validate the certificates received from a server.

Calling this function is a resource intensive operation because of the file access, parsing, and internal public key authentications required. For this reason, it is advised that this function be called once per set of key files for a given application. All new sessions associated with the certificate material can reuse the existing key pointer. At application shutdown the user must free the key structure using `matrixSslDeleteKeys`.

Client Authentication

If client authentication functionality is desired, all parameters to this function become relevant to both clients and servers. The `certFile` and `privFile` parameters are used to specify the identity certificate of the local peer. Likewise, each entity will need to supply a `trustedCAcertFile` parameter that lists the trusted CAs so that the certificates may be authenticated. It is easiest to think of client authentication as a mirror image of the normal server authentication when considering how certificate and CA files are deployed.

It is possible to configure a server to engage in a client authentication handshake without loading CA files. Enable the `SERVER_CAN_SEND_EMPTY_CERT_REQUEST` define in *matrixsslConfig.h* to allow the server to send an empty CertificateRequest message. The server can then use the certificate callback function to perform a custom authentication on the certificate returned from the client.

The MatrixSSL library must be compiled with `USE_CLIENT_AUTH` defined in *matrixsslConfig.h* to enable client authentication support.

Multiple CA Certificates and Certificate Chaining

It is not uncommon for a server to work from a certificate chain in which a series of certificates form a child-to-parent hierarchy. It is even more common for a client to load multiple trusted CA certificates if numerous servers are being supported.

There are two ways to pass multiple certificates to the `matrixSslLoadRsaKeys` API. The first is to pass a semi-colon delimited list of files to the `certFile` or `trustedCAcertFiles` parameters. The second way is to append several PEM certificates into a single file and pass that file to either of the two parameters. Regardless of which way is chosen, the `certFile` parameter MUST be passed in a child-to-parent order. The first certificate parsed in the chain MUST be the child-most certificate and each subsequent certificate must be the parent (issuer) of the former. There must only ever be one private key file passed to this routine and it must correspond with the child-most certificate.

Encrypted Private Keys

It is strongly recommended that private keys be password protected when stored in files. The `privPass` parameter of this API is the plaintext password that will be used if the private key is encrypted. MatrixSSL supports an MD5 based PKCS#5 2.0 PBKDF1 standard for password encryption. The most common way a password is retrieved is through user input during the initialization of an application.

Memory Profile

The keys parameter must be freed with `matrixSslDeleteKeys` after its useful life.

4.2 matrixSslLoadEcKeysMem

```
int32 matrixSslLoadEcKeysMem(sslKeys_t *keys, unsigned char *certBuf,
        int32 certLen, unsigned char *privBuf, int32 privLen,
        unsigned char *trustedCABuf, int32 trustedCALen);
```

Parameter	Input/Output	Description
keys	input/output	Allocated key structure returned from a previous call to `matrixSslNewKeys`. Will become input to `matrixSslNewClientSession` or `matrixSslNewServerSession` to associate key material with a SSL session.
certBuf	input	The X.509 ASN.1 identity certificate for this SSL peer. For file-based support, see `matrixSslLoadEcKeys` This parameter is always relevant to servers. Clients will want to supply an identity certificate and private key if supporting mutual authentication. `NULL` otherwise.
certLen	input	Byte length of `certBuf`
privBuf	input	The PKCS#8 or "SEC1: Elliptical Curve Cryptography" private EC key that was used to sign the `certBuf`. This parameter is always relevant to servers. Clients will want to supply an identity certificate and private key if supporting mutual authentication. `NULL` otherwise.
privLen	input	Byte length of `privBuf`
trustedCABuf	input	The X.509 ASN.1 stream of the trusted root certificates (Certificate Authorities) for this SSL peer. This parameter is always relevant to clients. Servers will want to supply a CA if requesting mutual authentication. `NULL` otherwise.
trustedCALen	input	Byte length of `trustedCABuf`

Return Value	Test	Description
PS_SUCCESS	0	Success. All input buffers parsed successfully and the keys parameter is available for use in session creation
PS_CERT_AUTH_FAIL	< 0	Failure. Certificate or chain did not self-authenticate or private key could not authenticate certificate
PS_PLATFORM_FAIL	< 0	Failure. Error locating or opening an input file
PS_ARG_FAIL	< 0	Failure. Bad input function parameter
PS_MEM_FAIL	< 0	Failure. Internal memory allocation failure
PS_PARSE_FAIL	< 0	Failure. Error parsing certificate or private key buffer
PS_UNSUPPORTED_FAIL	< 0	Failure. Unsupported key algorithm in certificate material

Servers and Clients

This function is the in-memory equivalent of the `matrixSslLoadEcKeys` API to support environments where the certificate material is not stored as files on disk. Please consult the documentation for `matrixSslLoadEcKeys` for detailed information on how clients and servers should manage the certificate and private key parameters. This API can be called at most once for a given `sslKeys_t` parameter.

There is no password protection support for private key buffers. It is recommended that the user implement secure storage for the private key material.

Multiple CA Certificates and Certificate Chaining

This in-memory version of the key parser also supports multiple CAs and/or certificate chains. Simply append the ASN.1 certificate streams together for either the `certBuf` or `trustedCAbuf` parameters. If using a certificate chain in the `certBuf` parameter the order of the certificates still MUST be in child-to-parent order with the `privBuf` being the key associated with the child-most certificate.

Memory Profile

The keys parameter must be freed with `matrixSslDeleteKeys` after its useful life.