

**NAME**

hx509 certificate functions -

**Functions**

int **hx509\_cert\_init** (hx509\_context context, const Certificate \*c, hx509\_cert \*cert)  
 int **hx509\_cert\_init\_data** (hx509\_context context, const void \*ptr, size\_t len, hx509\_cert \*cert)  
 void **hx509\_cert\_free** (hx509\_cert cert)  
 hx509\_cert **hx509\_cert\_ref** (hx509\_cert cert)  
 void **hx509\_verify\_ctx\_f\_allow\_default\_trustanchors** (hx509\_verify\_ctx ctx, int boolean)  
 int **hx509\_cert\_find\_subjectAltName\_otherName** (hx509\_context context, hx509\_cert cert, const heim\_oid \*oid, hx509\_octet\_string\_list \*list)  
 int **hx509\_cert\_cmp** (hx509\_cert p, hx509\_cert q)  
 int **hx509\_cert\_get\_issuer** (hx509\_cert p, hx509\_name \*name)  
 int **hx509\_cert\_get\_subject** (hx509\_cert p, hx509\_name \*name)  
 int **hx509\_cert\_get\_base\_subject** (hx509\_context context, hx509\_cert c, hx509\_name \*name)  
 int **hx509\_cert\_get\_serialnumber** (hx509\_cert p, heim\_integer \*i)  
 time\_t **hx509\_cert\_get\_notBefore** (hx509\_cert p)  
 time\_t **hx509\_cert\_get\_notAfter** (hx509\_cert p)  
 int **hx509\_cert\_get\_SPKI** (hx509\_context context, hx509\_cert p, SubjectPublicKeyInfo \*spki)  
 int **hx509\_cert\_get\_SPKI\_AlgorithmIdentifier** (hx509\_context context, hx509\_cert p, AlgorithmIdentifier \*alg)  
 int **hx509\_cert\_get\_issuer\_unique\_id** (hx509\_context context, hx509\_cert p, heim\_bit\_string \*issuer)  
 int **hx509\_cert\_get\_subject\_unique\_id** (hx509\_context context, hx509\_cert p, heim\_bit\_string \*subject)  
 int **hx509\_verify\_hostname** (hx509\_context context, const hx509\_cert cert, int flags, hx509\_hostname\_type type, const char \*hostname, const struct sockaddr \*sa, int sa\_size)  
 hx509\_cert\_attribute **hx509\_cert\_get\_attribute** (hx509\_cert cert, const heim\_oid \*oid)  
 int **hx509\_cert\_set\_friendly\_name** (hx509\_cert cert, const char \*name)  
 const char \* **hx509\_cert\_get\_friendly\_name** (hx509\_cert cert)  
 int **hx509\_query\_alloc** (hx509\_context context, hx509\_query \*\*q)  
 void **hx509\_query\_match\_option** (hx509\_query \*q, hx509\_query\_option option)  
 int **hx509\_query\_match\_issuer\_serial** (hx509\_query \*q, const Name \*issuer, const heim\_integer \*serialNumber)  
 int **hx509\_query\_match\_friendly\_name** (hx509\_query \*q, const char \*name)  
 int **hx509\_query\_match\_eku** (hx509\_query \*q, const heim\_oid \*eku)  
 int **hx509\_query\_match\_cmp\_func** (hx509\_query \*q, int(\*func)(hx509\_context, hx509\_cert, void \*), void \*ctx)  
 void **hx509\_query\_free** (hx509\_context context, hx509\_query \*q)  
 void **hx509\_query\_statistic\_file** (hx509\_context context, const char \*fn)  
 void **hx509\_query\_unparse\_stats** (hx509\_context context, int printtype, FILE \*out)  
 int **hx509\_cert\_check\_eku** (hx509\_context context, hx509\_cert cert, const heim\_oid \*eku, int

allow\_any\_eku)

int **hx509\_cert\_binary** (hx509\_context context, hx509\_cert c, heim\_octet\_string \*os)

int **hx509\_print\_cert** (hx509\_context context, hx509\_cert cert, FILE \*out)

### Detailed Description

See the **The basic certificate** for description and examples.

### Function Documentation

int **hx509\_cert\_binary** (hx509\_context context, hx509\_cert c, heim\_octet\_string \* os)

Encodes the hx509 certificate as a DER encode binary.

#### Parameters:

*context* A hx509 context.

*c* the certificate to encode.

*os* the encode certificate, set to NULL, 0 on case of error. Free the os->data with **hx509\_xfree**().

#### Returns:

An hx509 error code, see **hx509\_get\_error\_string**().

int **hx509\_cert\_check\_eku** (hx509\_context context, hx509\_cert cert, const heim\_oid \* eku, int allow\_any\_eku)

Check the extended key usage on the hx509 certificate.

#### Parameters:

*context* A hx509 context.

*cert* A hx509 context.

*eku* the EKU to check for

*allow\_any\_eku* if the any EKU is set, allow that to be a substitute.

#### Returns:

An hx509 error code, see **hx509\_get\_error\_string**().

int **hx509\_cert\_cmp** (hx509\_cert p, hx509\_cert q)

Compare to hx509 certificate object, useful for sorting.

#### Parameters:

*p* a hx509 certificate object.

*q* a hx509 certificate object.

#### Returns:

0 if the objects are the same, returns  $> 0$  if  $p$  is 'larger' than  $q$ ,  $< 0$  if  $p$  is 'smaller' than  $q$ .

**int hx509\_cert\_find\_subjectAltName\_otherName (hx509\_context context, hx509\_cert cert, const heim\_oid \* oid, hx509\_octet\_string\_list \* list)**

Return a list of subjectAltNames specified by oid in the certificate. On error the

The returned list of octet string should be freed with **hx509\_free\_octet\_string\_list()**.

**Parameters:**

*context* A hx509 context.

*cert* a hx509 certificate object.

*oid* an oid to for SubjectAltName.

*list* list of matching SubjectAltName.

**Returns:**

An hx509 error code, see **hx509\_get\_error\_string()**.

**void hx509\_cert\_free (hx509\_cert cert)**

Free reference to the hx509 certificate object, if the refcounter reaches 0, the object is freed. Its allowed to pass in NULL.

**Parameters:**

*cert* the cert to free.

**hx509\_cert\_attribute hx509\_cert\_get\_attribute (hx509\_cert cert, const heim\_oid \* oid)**

Get an external attribute for the certificate, examples are friendly name and id.

**Parameters:**

*cert* hx509 certificate object to search

*oid* an oid to search for.

**Returns:**

an hx509\_cert\_attribute, only valid as long as the certificate is referenced.

**int hx509\_cert\_get\_base\_subject (hx509\_context context, hx509\_cert c, hx509\_name \* name)**

Return the name of the base subject of the hx509 certificate. If the certificate is a verified proxy certificate, this function returns the base certificate (root of the proxy chain). If the proxy certificate is not verified with the base certificate HX509\_PROXY\_CERTIFICATE\_NOT\_CANONICALIZED is returned.

**Parameters:**

*context* a hx509 context.

*c* a hx509 certificate object.

*name* a pointer to a hx509 name, should be freed by **hx509\_name\_free()**. See also **hx509\_cert\_get\_subject()**.

**Returns:**

An hx509 error code, see **hx509\_get\_error\_string()**.

**const char\* hx509\_cert\_get\_friendly\_name (hx509\_cert cert)**

Get friendly name of the certificate.

**Parameters:**

*cert* cert to get the friendly name from.

**Returns:**

an friendly name or NULL if there is. The friendly name is only valid as long as the certificate is referenced.

**int hx509\_cert\_get\_issuer (hx509\_cert p, hx509\_name \* name)**

Return the name of the issuer of the hx509 certificate.

**Parameters:**

*p* a hx509 certificate object.

*name* a pointer to a hx509 name, should be freed by **hx509\_name\_free()**.

**Returns:**

An hx509 error code, see **hx509\_get\_error\_string()**.

**int hx509\_cert\_get\_issuer\_unique\_id (hx509\_context context, hx509\_cert p, heim\_bit\_string \* issuer)**

Get a copy of the Issuer Unique ID

**Parameters:**

*context* a hx509\_context

*p* a hx509 certificate

*issuer* the issuer id returned, free with **der\_free\_bit\_string()**

**Returns:**

An hx509 error code, see **hx509\_get\_error\_string()**. The error code

**HX509\_EXTENSION\_NOT\_FOUND** is returned if the certificate doesn't have a issuerUniqueID

**time\_t hx509\_cert\_get\_notAfter (hx509\_cert p)**

Get notAfter time of the certificate.

**Parameters:**

*p* a hx509 certificate object.

**Returns:**

return not after time.

**time\_t hx509\_cert\_get\_notBefore (hx509\_cert p)**

Get notBefore time of the certificate.

**Parameters:**

*p* a hx509 certificate object.

**Returns:**

return not before time

**int hx509\_cert\_get\_serialnumber (hx509\_cert p, heim\_integer \* i)**

Get serial number of the certificate.

**Parameters:**

*p* a hx509 certificate object.

*i* serial number, should be freed ith `der_free_heim_integer()`.

**Returns:**

An hx509 error code, see `hx509_get_error_string()`.

**int hx509\_cert\_get\_SPKI (hx509\_context context, hx509\_cert p, SubjectPublicKeyInfo \* spki)**

Get the SubjectPublicKeyInfo structure from the hx509 certificate.

**Parameters:**

*context* a hx509 context.

*p* a hx509 certificate object.

*spki* SubjectPublicKeyInfo, should be freed with `free_SubjectPublicKeyInfo()`.

**Returns:**

An hx509 error code, see `hx509_get_error_string()`.

**int hx509\_cert\_get\_SPKI\_AlgorithmIdentifier (hx509\_context context, hx509\_cert p, AlgorithmIdentifier**

**\* alg)**

Get the AlgorithmIdentifier from the hx509 certificate.

**Parameters:**

*context* a hx509 context.

*p* a hx509 certificate object.

*alg* AlgorithmIdentifier, should be freed with `free_AlgorithmIdentifier()`. The algorithmidentifier is typically `rsaEncryption`, or `id-ecPublicKey`, or some other public key mechanism.

**Returns:**

An hx509 error code, see `hx509_get_error_string()`.

**int hx509\_cert\_get\_subject (hx509\_cert p, hx509\_name \* name)**

Return the name of the subject of the hx509 certificate.

**Parameters:**

*p* a hx509 certificate object.

*name* a pointer to a hx509 name, should be freed by `hx509_name_free()`. See also

`hx509_cert_get_base_subject()`.

**Returns:**

An hx509 error code, see `hx509_get_error_string()`.

**int hx509\_cert\_get\_subject\_unique\_id (hx509\_context context, hx509\_cert p, heim\_bit\_string \* subject)**

Get a copy of the Subject Unique ID

**Parameters:**

*context* a hx509\_context

*p* a hx509 certificate

*subject* the subject id returned, free with `der_free_bit_string()`

**Returns:**

An hx509 error code, see `hx509_get_error_string()`. The error code

`HX509_EXTENSION_NOT_FOUND` is returned if the certificate doesn't have a `subjectUniqueID`

**int hx509\_cert\_init (hx509\_context context, const Certificate \* c, hx509\_cert \* cert)**

Allocate and init an hx509 certificate object from the decoded certificate 'c'.

**Parameters:**

*context* A hx509 context.

*c*

*cert*

**Returns:**

Returns an hx509 error code.

**int hx509\_cert\_init\_data (hx509\_context context, const void \* ptr, size\_t len, hx509\_cert \* cert)**

Just like **hx509\_cert\_init()**, but instead of a decode certificate takes an pointer and length to a memory region that contains a DER/BER encoded certificate.

If the memory region doesn't contain just the certificate and nothing more the function will fail with HX509\_EXTRA\_DATA\_AFTER\_STRUCTURE.

**Parameters:**

*context* A hx509 context.

*ptr* pointer to memory region containing encoded certificate.

*len* length of memory region.

*cert* a return pointer to a hx509 certificate object, will contain NULL on error.

**Returns:**

An hx509 error code, see **hx509\_get\_error\_string()**.

**hx509\_cert hx509\_cert\_ref (hx509\_cert cert)**

Add a reference to a hx509 certificate object.

**Parameters:**

*cert* a pointer to an hx509 certificate object.

**Returns:**

the same object as is passed in.

**int hx509\_cert\_set\_friendly\_name (hx509\_cert cert, const char \* name)**

Set the friendly name on the certificate.

**Parameters:**

*cert* The certificate to set the friendly name on

*name* Friendly name.

**Returns:**

An hx509 error code, see **hx509\_get\_error\_string()**.

**int hx509\_print\_cert (hx509\_context context, hx509\_cert cert, FILE \* out)**

Print a simple representation of a certificate

**Parameters:**

*context* A hx509 context, can be NULL

*cert* certificate to print

*out* the stdio output stream, if NULL, stdout is used

**Returns:**

An hx509 error code

**int hx509\_query\_alloc (hx509\_context context, hx509\_query \*\* q)**

Allocate an query controller. Free using **hx509\_query\_free()**.

**Parameters:**

*context* A hx509 context.

*q* return pointer to a hx509\_query.

**Returns:**

An hx509 error code, see **hx509\_get\_error\_string()**.

**void hx509\_query\_free (hx509\_context context, hx509\_query \* q)**

Free the query controller.

**Parameters:**

*context* A hx509 context.

*q* a pointer to the query controller.

**int hx509\_query\_match\_cmp\_func (hx509\_query \* q, int(\*) (hx509\_context, hx509\_cert, void \*) func, void \* ctx)**

Set the query controller to match using a specific match function.

**Parameters:**

*q* a hx509 query controller.

*func* function to use for matching, if the argument is NULL, the match function is removed.

*ctx* context passed to the function.

**Returns:**



An hx509 error code, see **hx509\_get\_error\_string()**.

**int hx509\_query\_match\_eku (hx509\_query \* q, const heim\_oid \* eku)**

Set the query controller to require an one specific EKU (extended key usage). Any previous EKU matching is overwitten. If NULL is passed in as the eku, the EKU requirement is reset.

**Parameters:**

*q* a hx509 query controller.

*eku* an EKU to match on.

**Returns:**

An hx509 error code, see **hx509\_get\_error\_string()**.

**int hx509\_query\_match\_friendly\_name (hx509\_query \* q, const char \* name)**

Set the query controller to match on a friendly name

**Parameters:**

*q* a hx509 query controller.

*name* a friendly name to match on

**Returns:**

An hx509 error code, see **hx509\_get\_error\_string()**.

**int hx509\_query\_match\_issuer\_serial (hx509\_query \* q, const Name \* issuer, const heim\_integer \* serialNumber)**

Set the issuer and serial number of match in the query controller. The function make copies of the issuer and serial number.

**Parameters:**

*q* a hx509 query controller

*issuer* issuer to search for

*serialNumber* the serialNumber of the issuer.

**Returns:**

An hx509 error code, see **hx509\_get\_error\_string()**.

**void hx509\_query\_match\_option (hx509\_query \* q, hx509\_query\_option option)**

Set match options for the hx509 query controller.

**Parameters:**

*q* query controller.

*option* options to control the query controller.

**Returns:**

An hx509 error code, see **hx509\_get\_error\_string()**.

**void hx509\_query\_statistic\_file (hx509\_context context, const char \* fn)**

Set a statistic file for the query statistics.

**Parameters:**

*context* A hx509 context.

*fn* statistics file name

**void hx509\_query\_unparse\_stats (hx509\_context context, int printtype, FILE \* out)**

Unparse the statistics file and print the result on a FILE descriptor.

**Parameters:**

*context* A hx509 context.

*printtype* type to print

*out* the FILE to write the data on.

**void hx509\_verify\_ctx\_f\_allow\_default\_trustanchors (hx509\_verify\_ctx ctx, int boolean)**

Allow using the operating system builtin trust anchors if no other trust anchors are configured.

**Parameters:**

*ctx* a verification context

*boolean* if non zero, using the operating systems builtin trust anchors.

**Returns:**

An hx509 error code, see **hx509\_get\_error\_string()**.

**int hx509\_verify\_hostname (hx509\_context context, const hx509\_cert cert, int flags, hx509\_hostname\_type type, const char \* hostname, const struct sockaddr \* sa, int sa\_size)**

Verify that the certificate is allowed to be used for the hostname and address.

**Parameters:**

*context* A hx509 context.

*cert* the certificate to match with

*flags* Flags to modify the behavior:

⊕ HX509\_VHN\_F\_ALLOW\_NO\_MATCH no match is ok

*type* type of hostname:

⊕ HX509\_HN\_HOSTNAME for plain hostname.

⊕ HX509\_HN\_DNSSRV for DNS SRV names.

*hostname* the hostname to check

*sa* address of the host

*sa\_size* length of address

**Returns:**

An hx509 error code, see **hx509\_get\_error\_string()**.