**HTTPS Decryption Common Module Design**

# Requirement

1. Supported deployment mode:
   1. Transparent mode: bridge, WCCP, bridge with upstream proxy.
   2. Proxy mode: proxy, proxy with upstream proxy
2. Server Certificate Validation
   1. CommonName does not match the URL
   2. Allow Wildcard-Certificates
   3. Deny expired or wrong purpose certificates
   4. Verify entire certificate chain
   5. Certificate revocation check by CRL
   6. Blacklist and whitelist, can you do improvement as (Comments from Avery):
      1. Check url or certification common name, support wildcard; if match, it can fall into 1 one follow behavior
      2. Tunnel: the https website will not be decrypted.
      3. Allow: the https website will not do certificate verification.
      4. Backlisted: the https website will not be allowed to access.
      5. Block: a warning message will be shown to end user even the certificate is valid, and end user can choose whether to visit the website continuously.
3. Client Certificate Handling
   1. Tunnel
   2. Block
4. Certificate Authority (used to resign certificate)
   1. Import CA
   2. Export CA
   3. Different CA format support, like pk12.
5. Digital Certificates Management & Synchronization
   1. Active Certificates
      1. [Add](http://10.64.75.212:1812/mmc_cert_add.jsp?CSRFGuardToken=JQZQKBO3IDMI3B0WC8QNGIS7PSK1J79U&type=active)
      2. [Delete](javascript:doDelete();)
      3. [De-activate](javascript:doDeactivate();)
   2. Inactive Certificates
      1. [Add](http://10.64.75.212:1812/mmc_cert_add.jsp?CSRFGuardToken=JQZQKBO3IDMI3B0WC8QNGIS7PSK1J79U&type=active)
      2. [Delete](javascript:doDelete();)
      3. [De-activate](javascript:doDeactivate();)
   3. Auto-archive unknown certificate into Inactive Certificate store
6. Notifications
   1. [HTTPS Access Denied](http://10.64.75.212:1812/notifications_accessdeny.jsp?CSRFGuardToken=JQZQKBO3IDMI3B0WC8QNGIS7PSK1J79U)
   2. [HTTPS Certificate Failure](http://10.64.75.212:1812/notifications_certerr.jsp?CSRFGuardToken=JQZQKBO3IDMI3B0WC8QNGIS7PSK1J79U)
   3. Client Certificate Request
7. SNI support
8. Some internal caches:
   1. FQDN cache
   2. Resigned certificate
   3. Client certificate cache
9. Some requirements from ICS:
10. Multi-CA certificates from different company
11. Multi-digital certificates stores for different company (manage well known certificates)

For these two requirements, we will use one same solution to handle it; all related interfaces will support company independency handle. For IWSVA and DeepEdge we can see all traffic is from one same company.

1. Re-signed same SN certificates from different scanners issues

HTTPS library provide one option for daemon to configure whether we need to force all re-signed certificates are same for one same web server.

# Work flow



|  |  |
| --- | --- |
| **Client Browser** | **iwssd(443)** |
| Connect | 1. Accept the socket connection |
| Client hello | 1. https\_new\_ctx(ctx, company\_id, user\_id, server cert verify option, blocking/non-blocking, deploymode …) 2. Connect to web server 3. https\_bind\_fd(ctx, cli\_fd, svr\_fd) 4. https\_handshake(ctx, https\_policy\_check\_cb)   Callback function: https\_policy\_check\_cb (ctx, server\_name/CommonName): check HTTPS decryption policy   * 1. Return 0: not need to decrypt, daemon should direct write the client hello to server and tunnel all below traffic   2. Return 1: follow below decryption steps   3. Return 2: need to wait for the asynchronous policy check finished   Return value:   1. Not https 2. Need more data: client, server 3. Wait for get resign certificate, the daemon will need to re-call this function when the resigned certificate ready. 4. Wait for the asynchronous policy check finished. 5. Successful, the decryption channel had been established 6. Client certificate required    * 1. Found in cache, so daemon not need to break the server connection, daemon can tunnel traffic with this established socket.      2. Found when handshake with server, the daemon should break the old server connection and then re-connect with web server for below tunnel task. 7. Verify certification failed, daemon need to generate warning msg to client browser, detail steps: 8. https\_client\_recv(ctx, buf, len, …)   Get client request, if found https certificate warning message token found, then call https\_ignore\_cert\_warning (ctx) to ignore this certificate error.   1. Prepare warning message and send to client browser. Generate the warning message with the certificate detail error message (https\_get\_error\_msg (ctx)) and warning token (it was used to let user ignore this certificate error), and then call https\_client\_send (ctx, buf, len) to response the client browser. 2. Read the client’s new request (https\_client\_recv) and then check if the request is the ignore warning request, if yes then add this certificate to whitelist 3. Not need to decrypt, but need to re-connect to web server, the old SSL channel had been established. 4. Not need to decrypt, and the SSL channel still not be established, the connection still can be used for below tunnel traffic. 5. Other error, like shakehand error, etc   Prompt client and break connection.   1. https\_err\_no\_hostname\_in\_clienthello   This return only can be returned when set option OP\_HOSTNAME\_ONLY\_FROM\_CLIENT\_HELLO (https\_set\_option(OP\_HOSTNAME\_ONLY\_FROM\_CLIENT\_HELLO)), and no hostname extension in client hello message.  When get this return code, daemon need to send the client hello data (call https\_get\_client\_untunneled\_data to get the data) to server, and then tunnel all below data.   1. Then receive and send request and response between client browser and web server: 2. https\_client\_recv(ctx, buf, len) 3. https\_server\_send(ctx, buf, len) 4. https\_server\_recv(ctx, buf, len)    1. Success    2. IIS Client certificate requirement, this only can happen on the first response.   Send 302 to client to notify user the server require client certificate and let browser re-connect the web server (tunnel)   * 1. Other failure  1. https\_client\_send(ctx, buf, len) 2. https\_delete\_ctx(ctx): |
|  |  |

1. [Comments from Wayne] Regarding the disconnect/reconnect for the case of server IPs without any entry in the re-signed certificate cache, I still think this can be handled without disconnect, but it is trickier than just using SSL\_accept() and SSL\_connect() directly.  In the attached powerpoint I illustrate the method I was talking about in my initial email.  The only tricky part is setting up the initial state for MITM\_CLT (the one which talks to the server) – since we already passed through the real ClientHello, we need to modify the internal state of MITM\_CLT to match what it would have been if we had chosen the same version, random data, timestamp, compression and cipher options and already sent that data upstream.   But I think this is not too big of a programming challenge.

*[Haijun Chen] We will evaluation this solution detai later, but firstly we suggest we can delivery one quick solution based on SSL\_connect directly, and keep one interface for switch to this solution after we can make sure this solution can work and no any side impact.*

1. [Comments from Wayne] Regarding renegotiation, I wasn’t really suggesting that we pass through renegotiation requests from real server down to client and keep session going.  Right now, we detect that IIS wants a client certificate but didn’t get it by examining decrypted application level traffic to find response code 403.7.  But under the covers what IIS does is set up initial SSL session without client certificate request, then later it sends renegotiation and asks for client certificate at that time.  I was thinking we could detect this condition at the SSL level instead of at the application level – if we see renegotiation with client certificate request in a MITM session, then consider that the same error and handle the same way as current 403.7 (send the client 302 and close the session).

*[Haijun Chen] We will detail evaluation this solution later, and its feasibility and side impact and the compatibility between web server and different browsers.*

# Certificate folder structures

1. **https\_ca**

All companies’ CA will be put here.

The default key: default.cer/default\_key.cer/

.default\_passphrase

The company key: [company\_id].cer/[company\_id]\_key.cer/

.[company\_id]\_passphrase

1. **certificates**

All certificates will be put here, include well-known certificates and customer private CA. Then other companies only keep one link to this folder for all trusted or untrusted certificates.

1. **default.trust/default.untrust**

When not specifies company, this library will use this default certificate trust/untrust list to verify certificate, the content is like:

MD5\_qlzDOH-F7nUGuqYGlgd0iQ\_\_.pem

MD5\_ASGE8krmoXFi5OOPfx26gQ\_\_.pem

MD5\_K7zmFFk+SdXd3vjtaZohZA\_\_.pem

MD5\_G0APVxUJepi6ZzaMmq8MQA\_\_.pem

1. **company\_id.trsust/company\_id.untrust**

Company independent trusted/untrusted certificates list.

1. **crl and rehash**

Use to store CRL data which downloaded from all CA sites.

1. **script**

All related script will be put here, include certificate management script, crl download tool, etc

# Certificate cache

1. **Resigned certificate cache**

Cache index include: Company ID, Server certificate hash

Cache content: resigned certificate

Support two certificate resign methods:

1, IWSVA solution: increased serial-no, server certificate’s subject and other extension information of the certificate.

2, ICS solution: use original CA’s subject + certificate’s SN to generate the random data for new certificate.

1. **Visit anyway cache**

Cache index include: Company ID, User ID, Server Certificate’s hash value,

Cache content: Begin time (for TTL control)

1. **CA store**

Cache index: Company ID

Cache content: CA

1. **Client certificate cache**

Cache index: Server IP, Server Port, client certificate type (IIS or other)

# Data structures

1. **Error code**

#define https\_err\_base 0

#define https\_err\_success (https\_err\_base)

#define https\_err\_not\_https (https\_err\_base - 1)

#define https\_err\_loadcert (https\_err\_base - 2)

#define https\_err\_company\_cert\_store (https\_err\_base - 3)

#define https\_err\_fd\_bind (https\_err\_base - 4)

#define https\_err\_not\_handshake (https\_err\_base - 5)

#define https\_err\_client\_cert (https\_err\_base - 6)

#define https\_err\_client\_cert\_reconn\_server (https\_err\_base - 7)

#define https\_err\_certificate (https\_err\_base - 8)

#define https\_err\_token (https\_err\_base - 9)

#define https\_err\_policy\_not\_decrypt (https\_err\_base - 10)

#define https\_err\_policy\_not\_decrypt\_reconn\_server (https\_err\_base - 11)

#define https\_err\_shutdown (https\_err\_base - 12)

#define https\_err\_want\_read\_client (https\_err\_base - 13)

#define https\_err\_want\_write\_client (https\_err\_base - 14)

#define https\_err\_want\_read\_server (https\_err\_base - 15)

#define https\_err\_want\_write\_server (https\_err\_base - 16)

#define https\_err\_unknown (https\_err\_base - 1000)

1. **https\_certverify\_option**

/\*

\* define the certificate verify setting

\*/

typedef struct \_https\_certverify\_option{

BOOL verify\_certificate;

BOOL commonname;

BOOL allow\_wildcard;

BOOL expire;

BOOL cert\_chain;

BOOL crl;

BOOL ocsp; // reserved for next version, current not support this verify method

} https\_certverify\_option;

1. **https\_ctx**

/\*

\* context for one whole https session

\*/

typedef struct \_https\_ctx {

SSL \* client\_ssl; // this SSL variable is used to communication with client browser

SSL \* server\_ssl; // this SSL variable is used to communication with web server

char \* company\_id;

char \* user\_id;

https\_certverify\_option verify\_cert\_option;

void \* param; // daemon can put any data with this pointer

/\*

\* below 3 parameters are used internal

\*/

int status; // for non-blocking IO, it will be used to track the status of handshake

int status\_2nd; // for non-blocking handshake and wait the resigned certificate

int err\_code;

char \*err\_msg;

BOOL bcert\_inited; // the resigned certificate had been setup

}https\_ctx;

1. **https\_deploymode**

/\*

\* define daemon work under which deployment mode

\*/

typedef enum \_https\_deploymode

{

Transparent, // transparent mode, like bridge, wccp, etc

Proxy, // proxy mode

ProxyWithUpstream // proxy mode, but it still has upstream proxy

}https\_deploymode;

# API Functions

## Initialize/finalize

1. **https\_init**

/\*

\* Initialize https decryption library.

\* Param:

\* mode: setup the deploymode

\* independent\_ca: whether use different CA to re-sign certificate for different company

\* independent\_trusted\_cert: whether use different certificate store to verify certificate for different company

\* force\_same\_resigned\_cert: force the re-signed cert are same for one same web server certificate

\*

\* Return:

\* https\_err\_success: Successful

\* https\_err\_loadcert: load certificates failed

\* https\_err\_unknown: unknown error

\*/

int https\_init(https\_deploymode mode, BOOL independent\_ca, BOOL independent\_trusted\_cert, char \*root\_cert\_dir, BOOL force\_same\_resigned\_cert);

1. **https\_finalize**

/\*

\* Finalizes https decryption library.

\*/

int https\_finalize ();

## Traffic handling

1. **https\_new\_ctx**

/\*

\* Create one https decryption context to hold the data for a https session

\* Param:

\* company\_id: if only handle one company's scan task, please set it to NULL

\* user\_id: this parameter will be used to handle visit anyway cache

\* vfy: verify certificate options

\* non\_blocking\_io: set Non-blocking-IO option

\* upstream\_proxy: daemon found this https connection request will be handled by upstream proxy

\* conn\_host, conn\_port: if upstream proxy exist, these 2 params will be the real web server's hostname and port

\* webserver\_ip: the real web server's IP address, it will be used for cache

\*

\* Return:

\* NULL:

\* Failed, you can call https\_get\_error to get the detail error message. The error maybe https\_err\_company\_cert\_store or https\_err\_unknown.

\* one new https\_ctx object

\*/

https\_ctx \*https\_new\_ctx(const char \*company\_id, const char \*user\_id, https\_certverify\_option \*vfy, BOOL non\_blocking\_io, BOOL upstream\_proxy, const char \*conn\_host, int conn\_port, const char webserver\_ip[16]);

1. **https\_delete\_ctx**

/\*

\* Destory one https decryption context

\* Param:

\* ctx: https\_ctx

\*/

void https\_delete\_ctx(https\_ctx \*ctx);

1. **https\_bind\_fd**

/\*

\* Bind client socket and server socket to https context

\* Param:

\* ctx: https context

\* cli\_fd, svr\_fd: client and server socket

\*

\* Return:

\* https\_err\_success: Successful

\* https\_err\_fd\_bind: socket bind failed

\*/

int https\_bind\_fd(https\_ctx \*ctx, int cli\_fd, int svr\_fd);

1. **https\_handshake**

/\*

\* Do https handshake with server and client, it will handle:

\* 1. get server\_name under transparent mode and call the callback function to check https decriyption policy

\* 2. check if it is ssl channel

\* 3. certificate verify

\* 4. re-sign certificate

\* 5. handshake with server and client

\* 6. client certificate check

\* 7. cert cache manage

\* Param:

\* ctx: https context

\* https\_policy\_check\_cb: callback function for daemon check if this https session need to be decrypted, if daemon under proxy mode, you can set it to NULL.

\*

\* Return:

\* 1. https\_err\_success: decryption channel had been established successful

\* 2. https\_err\_not\_https: not https traffic

\* 3. need more data:

\* https\_err\_want\_read\_client, https\_err\_want\_write\_client, https\_err\_want\_read\_server, https\_err\_want\_write\_server, https\_wait\_async\_policy\_check, https\_wait\_resigned\_cert

\* 4. Client certificate requirement:

\* https\_err\_client\_cert:

\* Found in cache, so daemon not need to break the server connection, daemon can tunnel traffic with this established socket.

\*

\* https\_err\_client\_cert\_reconn\_server:

\* Found the server require client certificate when handshake with server, the daemon should break the old server connection and then re-connect with web server for next tunnel task.

\*

\* 5. https\_err\_certificate: verify certification failed, daemon need to generate warning msg to client browser, detail steps:

\* 6. Policy check result: not need to decrypt

\* https\_err\_policy\_not\_decrypt:

\* Not need to decrypt this https, and daemon still can use the established connection with server for next tunnel task

\* https\_err\_policy\_not\_decrypt\_reconn\_server:

\* Not need to decrypt this https, but daemon need to establish one new connection with server for next tunnel task

\*

\* 7. https\_err\_unknown: other unknown error

\* 8. https\_wait\_resigned\_cert: wait for asynchronous get resigned certificate, after daemon get this return code, it should re-call handshake after the resigned cert ready. Daemon can call https\_setup\_resigned\_cert to setup the certificate else common library will still call https\_get\_resigned\_cert to get the resigned cert.

\*

\* 9. https\_err\_no\_hostname\_in\_clienthello

\* This return only can be returned when set option OP\_HOSTNAME\_ONLY\_FROM\_CLIENT\_HELLO (https\_set\_option(OP\_HOSTNAME\_ONLY\_FROM\_CLIENT\_HELLO)), and no hostname extension in client hello message.

\* When get this return code, daemon need to send the client hello data (call https\_get\_client\_untunneled\_data to get the data) to server, and then tunnel all below data.

\*

\* int (\*https\_policy\_check\_cb)(https\_ctx \*ctx, const char \*server\_name):

\* callback function for daemon check if this https session need to be decrypted.

\* Return:

\* 0: not need to decrypt

\* 1: need to decrypt

\* 2: need to re-call this callback function for asynchronous callback found, like support asynchronous TMUFE query

\* Note:

\* if handshake return tunnel data, the daemon need to call https\_get\_client\_untunneled\_data to get the data which kept in common library and then send to web server. (Major for the client-hello package).

\*/

int https\_handshake(https\_ctx \*ctx, int (\*https\_policy\_check\_cb)(https\_ctx \*ctx, const char \*server\_name));

1. **https\_get\_client\_untunneled\_data**

/\*

\* Get the un-tunneled data when handshake return value was tunnel (the data is client hello package)

\* Param:

\* ctx: https\_ctx

\* ppdata: it will take the buffer of the data, it is a output parameter

\* Return:

\* the length of the data which need to send to web server (tunnel)

\* 0: means no data

\*/

int https\_get\_client\_untunneled\_data(https\_ctx \*ctx, char \*\*ppdata);

1. **https\_post\_commonname\_check**

/\*

\* Handshake cannot get the really hostname at sometimes, so handshake cannot do commonName

\* check, it should be handled when daemon get the real host and then call https\_post\_commonname\_check

\* to check the commonName.

\*

\* Param:

\* ctx: https\_ctx

\* hostname: the hostname of the web server

\* Return:

\* https\_err\_success: no any error, it was matched

\* https\_err\_certificate: commonName not match hostname

\* https\_err\_unknown: unknown error happened

\*/

int https\_post\_commonname\_check(https\_ctx \*ctx, const char \*hostname);

1. **https\_setup\_resigned\_cert**

/\*

\* Setup the resigned certificate, only used for asynchronous get the resigned certificate

\*/

BOOL https\_setup\_resigned\_cert(https\_ctx \*ctx, X509 \*cert, EVP\_PKEY \*pkey);

1. **send/receive plain text content from https context**

/\*

\* Send/receive plain text content from https context

\*

\* Param:

\* ctx: https\_ctx

\* buf: send/receive data buffer

\* len: the length of send/receive data buffer

\*

\* Return:

\* >0:

\* The send/recv operation was successful, the return value is the number of bytes actually sent/recv

\* https\_err\_not\_handshake:

\* The handshake still not done.

\* https\_err\_shutdown:

\* The connection had been shutdown by peer.

\* https\_err\_unknown:

\* other unknown error

\* https\_err\_want\_read\_client / https\_err\_want\_write\_client / https\_err\_want\_read\_server / https\_err\_want\_write\_server:

\*

\* One specially return code for function: https\_server\_recv

\* https\_client\_certificate\_iis:

\* this web server is IIS and it require client certificate

\*/

int https\_client\_recv(https\_ctx \*ctx, void \*buf, int len);

int https\_client\_send(https\_ctx \*ctx, const void \*buf, int len);

int https\_server\_recv(https\_ctx \*ctx, void \*buf, int len);

int https\_server\_send(https\_ctx \*ctx, const void \*buf, int len);

1. **check if still has some data buffered in ssl buffer**

/\*

\* check if still has some pending data in SSL buffer

\*/

BOOL https\_client\_pending(https\_ctx \*ctx);

BOOL https\_server\_pending(https\_ctx \*ctx);

## Other functions

1. **https\_get\_error**

/\*

\* Get the last error code

\*/

int https\_get\_error(https\_ctx \*ctx);

1. **https\_get\_error\_msg**

/\*

\* Get the last error message

\*/

char \* https\_get\_error\_msg(https\_ctx \*ctx);

1. **https\_ignore\_cert\_warning**

/\*

\* Ignore the certificate warning

\*

\* Param:

\* ctx: https\_ctx

\*

\* Return:

\* None

\*/

void https\_ignore\_cert\_warning(https\_ctx \*ctx);

1. **https\_get\_org\_cert**

/\*

\* Get the original certificate when popup the cert warning message

\*

\* Param:

\* ctx: https\_ctx

\* buf: output buffer for output the original certificate (DER)

\* len: the buffer size

\*

\* Return:

\* >0: the certificate length in the buf

\* https\_err\_too\_short: buffer length is not enough

\* https\_err\_unknown: get the original certificate failed

\*/

int https\_get\_org\_cert(https\_ctx \*ctx, unsigned char \*buf, int len);

1. **https\_reload\_ca(const char \*company\_id)**

/\*

\* Reload the CA certificate after one customer upload new CA certificate

\* Param:

\* company\_id: company id

\*

\* Return:

\* https\_err\_success: successful

\* https\_err\_unknown: error happened

\*/

int https\_reload\_ca(const char \*company\_id);

1. **https\_clean\_cert\_cache(const char \*company\_id, https\_cert\_cache\_type etype)**

/\*

\* Clean one company’s certificate cache

\* Param:

\* company\_id: company id

\* etype: cache type

\*

\* Return:

\* https\_err\_success: successful

\* https\_err\_unknown: error happened

\*/

typedef enum \_https\_cert\_cache\_type{ALL, CLIENT\_CERT, CLIENT\_CERT\_IIS, CLIENT\_CERT\_OTHER, RESIGNED\_CERT, VISIT\_ANYWAY} https\_cert\_cache\_type;

int https\_clean\_cert\_cache(const char \*company\_id, https\_cert\_cache\_type etype);

1. **set/get daemon’s private data into https\_ctx**

/\*

\* Daemon can set some private data into https\_ctx for some especially objective, like generate log.

\*

\* Param:

\* ctx:

\* data: some data of daemon

\*

\* Return:

\* None

\*/

void https\_ctx\_set\_private\_data(https\_ctx \*ctx, void \*data);

void \*https\_ctx\_get\_private\_data(https\_ctx \*ctx);

1. **https\_set\_option**

/\*

\* Config some setting

\* Parameters:

\* OP\_DEBUG\_LOG: log file name

\* https\_set\_option(OP\_DEBUG\_LOG, "/etc/iscan/log/https.log");

\* OP\_DISABLE\_RESIGNED\_CERT\_CACHE: None

\* https\_set\_option(OP\_DISABLE\_RESIGNED\_CERT\_CACHE );

\* OP\_SET\_TTL\_VISITANYWAY (second)

\* https\_set\_option(OP\_SET\_TTL\_VISITANYWAY, 60\*60);

\* OP\_FORCE\_HANDSHAKE\_1ST\_DETECT\_CLIENT\_CERT:

\* Force finished the handshake when 1st time detect one site need client cert,

\* it is for error code https\_err\_client\_cert\_reconn\_server when handshake.

\* Note: it is not for IIS client cert.

\*

\* OP\_HOSTNAME\_ONLY\_FROM\_CLIENT\_HELLO

\* https\_set\_option(OP\_HOSTNAME\_ONLY\_FROM\_CLIENT\_HELLO);

\* Only get the hostname from the client hello, if no hostname in client helle, not

\* need to get the hostname from the server certificate.

\* Currently, most of newer version's browser support hostname extension in client

\* hello message. so we don't need to get the hostname from server's certificate at

\* most condition.

\*/

typedef enum \_HTTPS\_OPTIONS

{

OP\_DEBUG\_LOG,

OP\_DISABLE\_RESIGNED\_CERT\_CACHE,

OP\_SET\_TTL\_VISITANYWAY,

OP\_FORCE\_HANDSHAKE\_1ST\_DETECT\_CLIENT\_CERT,

OP\_HOSTNAME\_ONLY\_FROM\_CLIENT\_HELLO

}HTTPS\_OPTIONS;

void https\_set\_option(HTTPS\_OPTIONS op, ...);

# Certificate cache method replacement

Some products maybe need to implement themselves cache mechanism or central manage the certificate cache for multi-scanners. HTTPs common library will export all related API pointer, daemon can replace these method with other.

The methods which can be replaced included:

1. Manage the CA for every company
2. Re-signed certificate and cache them
3. Visit anyway cache
4. Client certificate cache
5. Server hostname cache
6. **Replace the cache method**

Daemon need to fill one below structure and then call https\_replace\_cache\_method to replace the related method. If daemon only needs to replace part of the common library’s method, just keep the other to NULL.

You can see below for detail description about every method.

typedef struct \_cert\_cache\_method {

int (\*https\_get\_resigned\_cert)(const char \*company\_id, X509 \*org\_cert, X509 \*\*cert, EVP\_PKEY \*\*pkey);

int (\*https\_get\_default\_entity\_cert)(const char \*company\_id, X509 \*\*cert, EVP\_PKEY \*\*pkey);

BOOL (\*https\_check\_client\_cert)(const char \*server\_ip, unsigned short server\_port);

void (\*https\_new\_client\_cert)(const char \*server\_ip, unsigned short server\_port, https\_cert\_cache\_type etype);

int (\*https\_get\_hostname)(const char \*server\_ip, unsigned short server\_port, char \*hostname, int max\_len);

int (\*https\_set\_hostname)(const char \*server\_ip, unsigned short server\_port, const char \*hostname);

BOOL (\*https\_check\_visit\_anyway)(const char \*company\_id, const char \*user\_id, X509 \*cer);

void (\*https\_new\_visit\_anyway)(const char \*company\_id, const char \*user\_id, X509 \*cer);

}cert\_cache\_method;

/\*

\* Replace cert cache method

\* Param:

\* method: the new cache methods

\*

\* Return:

\* https\_err\_success: successful

\* https\_err\_unknown: error happened

\*/

int https\_replace\_cache\_method(cert\_cache\_method \*method);

1. **Detail method**
2. https\_get\_resigned\_cert

/\*

\* Get the resigned certificate

\* Param:

\* company\_id: company id

\* org\_cert: original server certificate

\* cert/pkey: resigned certificate and private key

\*

\* Return:

\* https\_err\_success: successful

\* https\_err\_unknown: error happened

\*/

int https\_get\_resigned\_cert(const char \*company\_id, X509 \*org\_cert, X509 \*\*cert, EVP\_PKEY \*\*pkey);

1. https\_get\_default\_entity\_cert

/\*

\* Get the default entity certificate which signed by the CA of company, this certificate will be used when

\* cannot get server certificate, but we also need to use one certificate to communication

\* with client.

\*

\* Param:

\* company\_id: company id

\* cert/pkey: resigned certificate and private key

\*

\* Return:

\* https\_err\_success: successful

\* https\_err\_unknown: error happened

\*/

int https\_get\_default\_entity\_cert(const char \*company\_id, X509 \*\*cert, EVP\_PKEY \*\*pkey);

1. check and add client cert cache

/\*

\* Check if the web server require client certificate

\*

\* Param:

\* server\_ip/server\_port: web server's IP and port

\*

\* Return:

\* TRUE; this web server require client certificate

\* FALSE: this web server not require client certificate or not in our cache

\*/

BOOL https\_check\_client\_cert (const char \*server\_ip, unsigned short server\_port);

/\*

\* Find new web server require client certificate

\*

\* Param:

\* Server\_ip/server\_port: web server's IP and port

\* etype: CLIENT\_CERT\_IIS or CLIENT\_CERT\_OTHER

\*

\* Return:

\* None

\*/

void https\_new\_client\_cert(const char \*server\_ip, unsigned short server\_port, https\_cert\_cache\_type etype);

1. https\_get\_hostname/ https\_set\_hostname

/\*

\* Get/set the hostname from/to the cache

\*

\* Param:

\* server\_ip/server\_port: web server's IP and port

\* hostname/max\_len: return the hostname

\*

\* Return:

\* https\_err\_success: get the hostname from cache successful

\* https\_err\_unknown: not find the hostname from cache

\*/

int https\_get\_hostname(const char \*server\_ip, unsigned short server\_port, char \*hostname, int max\_len);

int https\_set\_hostname(const char \*server\_ip, unsigned short server\_port, const char \*hostname);

1. Visit anyway cache check and add new record

/\*

\* Check if customer wants to continue visit the untrusted web server

\*

\* Param:

\* company\_id: company id

\* user\_id: user id

\* cer: the web server's certificate

\*

\* Return:

\* TRUE; the customer wants to visit the site anyway

\* FALSE: not found in cache

\*/

BOOL https\_check\_visit\_anyway (const char \*company\_id, const char \*user\_id, X509 \*cer);

/\*

\* Customer want to visit the untrusted web server anyway, he ignore the certificate warning message

\*

\* Param:

\* company\_id: company id

\* user\_id: user id

\* cer: the web server's certificate

\*

\* Return:

\* None

\*/

void https\_new\_visit\_anyway(const char \*company\_id, const char \*user\_id, X509 \*cer);

# Some helper function for Web

1. **Import/export CA**
2. https\_ca\_import [–c company\_id] –p [password] --ca1 certfile1 [--ca2 certifile2]

Import CA certificate for re-sign web server’s certificate, detail parameters:

*company\_id: optional parameter, currently use for ICS, by default is import the default CA*

*password: the password of the certificate*

*certfile1[ certfile2] :If certificate and private key are separated, certfile1 is certificate file and certfile2 is private key file. Otherwise, only --ca1 should be specified.*

Return value: 0, successful

-1, Invalid CA certificate

-2, Invalid CA key or error password

-3, CA cert and CA key are not match.

-4, unknown error.

1. https\_ca\_export [–c company\_id] –format [pkcs12|pem] filename

*pkcs12|pem: the exported CA file’s format, support pkcs12 or base64 pem format*

*filename: exported file, the CA will be exported into this file*

1. **Trusted/untrusted certificate management**
2. **List all trusted or untrusted or both of them CA certificate:**

https\_store\_list\_cert [–c company\_id] –trust|-untrust|-all

This command’s output will be a json file, like:

{

"return": 0,

"data": [

{

"cert\_id": “abc29203df”,

"cn": "Trust Universal CA 1",

"validFrom": "2009-03-05 13:00:00",

"validTo": "2029-03-05 13:00:00",

"status": "inactive|revoke"

},

{

"cert\_id": “abc29203de”,

"cn": "Trust Universal CA 2",

"validFrom": "2009-03-05 13:00:00",

"validTo": "2029-03-05 13:00:00",

"status": "active"

}

]

}

1. **Get one certificate’s detail information:**

https\_store\_get\_detail\_info cert\_id

{

"return": 0,

"data": [

{

"cert\_id": “abc29203df”,

"cn": "Trust Universal CA 1",

“type”: “Root CA certificate”

“sn”: “1”

"validFrom": "2009-03-05 13:00:00"

"validTo": "2029-03-05 13:00:00",

"issuer": "CN=GeoTrust Universal CA, O=GeoTrust Inc., C=US"

“subject”:” CN=GeoTrust Universal CA, O=GeoTrust Inc., C=US”

"pubKey": "RSA (2048 bits)"

"usage": "Digital Signature, Certificate Signing, CRL Signing"

"MD5ft": "9265 588b a21a 3172 7368 5cb4 a57a 0748"

"SHA1ft": "e621 f335 4379 059a 4b68 309d 8a2f 7422 1587 ec79"

},

]

}

1. **Active/deactivate one certificate**

https\_store\_active\_cert [-c company\_id] cert\_id

https\_store\_deactive\_cert [-c company\_id] cert\_id

1. **Import/export certificates**

https\_store\_export\_cert [-c company\_id] –o export\_filename [cert\_id cert\_id2 ….]

*Export one or more certificates into one file, if not specific cert\_id, it will export all certificates (This API’s requirement is too strange, what can user do with the exported file?)*

https\_store\_import\_cert [-c company\_id] –trust/-untrust import\_cert\_file

*Import certificate into trust or untrusted certificate list from one certificate files.*

*Return value: 0 successful*

*-1 unsupported format*

*-2 not CA cert*

*-3 duplicated cert*

1. **Delete certicates**

**https\_store\_delete\_cert [-c company\_id] cert\_id**

# Other integration tasks

1. **CRL**
   1. Need to add one schedule task (crontab) to schedule download the CRL list

Script: certstore/script/download\_crl.sh

Note:

This script has one dependency; the ***wget*** command must be included in system.

* 1. Startup the “certstore/script/wait\_new\_crl.py start” before daemon startup, it will be used to receive new CRL from daemon.

1. **Rehash certificates**
   1. Call rehash script (certstore/script/rehash\_cert.sh) to rehash all certificates for common library after the certstore folder was defined.

# About C++ interface

Maybe some product need the C++ interface, DeepEdge will require the C interface emergency, so we will implement the C interface first and then wrap the C interface into one C++ class for C++ product use.