Snippets from TLS handshake

```
bool CTLSClient::PerformClientHandshake()
{
SecBuffer       OutBuffers[1];
    OutBuffers[0].pvBuffer   = NULL;
    OutBuffers[0].BufferType = SECBUFFER_TOKEN;
    OutBuffers[0].cbBuffer   = 0;
    SecBufferDesc   OutBuffer;
    OutBuffer.cBuffers = 1;
    OutBuffer.pBuffers = OutBuffers;
    OutBuffer.ulVersion = SECBUFFER_VERSION;
            ASSERT(m_pSSPI);
    TimeStamp       tsExpiry;

            SECURITY_STATUS secStatus = m_pSSPI->InitializeSecurityContextA(
                                                            m_phClientCreds,
                                                            NULL,
                                                            (char*)(const
char*)m_strServerName,
                    dwSSPInFlags,
                    0,
                    SECURITY_NATIVE_DREP,
                    NULL,
                    0,
                    &m_hContext,
                    &OutBuffer,
                    &dwSSPIOutFlags,
                    &tsExpiry);

    if(secStatus != SEC_I_CONTINUE_NEEDED)
    {
        TRACE("**** Error %d returned by InitializeSecurityContext (1)\n",
secStatus);
        m_strLastError.Format("InitializeSecurityContext Failed - Status =
%d",secStatus);
                        return false;
    }

SECURITY_STATUS CTLSClient::ClientHandshakeLoop(SecBuffer * pExtraDataBuf,bool
bDoInitialRead)
{
    bool fDoRead = bDoInitialRead;
     ////////////////////// ////////////
            ////////  IO_BUFFER_SIZE - cbIoBuffer
            ///////   ////////////////////////////
            const int IOSIZE = 0x10000; // 64K
            PBYTE IoBuf  = (PBYTE)_alloca (IOSIZE);
    //
    // Loop until the handshake is finished or an error occurs.
    //
            SECURITY_STATUS secStatus = SEC_I_CONTINUE_NEEDED;
            DWORD totBytesReceived(0);
            while(secStatus == SEC_I_CONTINUE_NEEDED        ||
```

```
          secStatus == SEC_E_INCOMPLETE_MESSAGE     ||
          secStatus == SEC_I_INCOMPLETE_CREDENTIALS)
   {

        // Read data from server.
        if(0 == totBytesReceived || secStatus == SEC_E_INCOMPLETE_MESSAGE)
        {
            if(fDoRead)
            {

                                                    // test int cbData =
ReceiveToBuffer((char *)IoBuf + totBytesReceived,10,10);
                                                    int cbData =
ReceiveToBuffer((char *)IoBuf + totBytesReceived,IOSIZE -
totBytesReceived,HANDSHAKE_WAIT);
                                                    if(cbData == SOCKET_ERROR)
                                                    {
                                                                TRACE("**** Error %d
reading data from server\n", WSAGetLastError());

m_strLastError.Format("Socket Error %d receiving from
server",WSAGetLastError());
                                                                secStatus =
SEC_E_INTERNAL_ERROR;
                                                                break;
                                                    }
                                                    else if(cbData == 0)
                                                    {
                                                                TRACE("**** Server
unexpectedly disconnected\n");
                                                                m_strLastError =
"Server disconnected unexpectedly.";
                                                                secStatus =
SEC_E_INTERNAL_ERROR;
                                                                break;
                                                    }

                                                    TRACE("%d bytes of handshake
data received\n", cbData);
                totBytesReceived += cbData;
            }
            else
            {
                fDoRead = TRUE;
            }
        }


        //
        // Set up the input buffers. Buffer 0 is used to pass in data
        // received from the server. Schannel will consume some or all
        // of this. Leftover data (if any) will be placed in buffer 1 and
        // given a buffer type of SECBUFFER_EXTRA.
        //
                        DWORD dwSSPInFlags = ISC_REQ_SEQUENCE_DETECT   |
```

```
ISC_REQ_REPLAY_DETECT     |

ISC_REQ_CONFIDENTIALITY   |

ISC_RET_EXTENDED_ERROR    |

ISC_REQ_ALLOCATE_MEMORY   |

ISC_REQ_STREAM;

                        DWORD dwSSPIOutFlags(0);
                        TimeStamp       tsExpiry;

                        SecBuffer       InBuffers[2];
        InBuffers[0].pvBuffer   = IoBuf;
        InBuffers[0].cbBuffer   = totBytesReceived;
        InBuffers[0].BufferType = SECBUFFER_TOKEN;

        InBuffers[1].pvBuffer   = NULL;
        InBuffers[1].cbBuffer   = 0;
        InBuffers[1].BufferType = SECBUFFER_EMPTY;

                        SecBufferDesc   InBuffer;
        InBuffer.cBuffers       = 2;
        InBuffer.pBuffers       = InBuffers;
        InBuffer.ulVersion      = SECBUFFER_VERSION;



Certificate Verification

bool CDatawireClient::VerifyServerCertificate()
{
            ASSERT(m_bSecurityContextCreated);
            PWSTR   pwszServerName = NULL;
            DWORD cchServerName;
            const CString strServerName (m_ConnectionInfo.m_strHost);
            if(!strServerName.GetLength() )
            {

                        m_strLastError  = ("VerifyServerCertificate Failed (A) -
The target principal name is incorrect.");
                        m_nLastError= SSPI_CERT_E_CRITICAL;

                        return false;
            }
            cchServerName = MultiByteToWideChar(CP_ACP, 0, strServerName, -1,
NULL, 0);
    pwszServerName       = (PWSTR)::LocalAlloc(LMEM_FIXED, cchServerName *
sizeof(WCHAR));
            if(pwszServerName == NULL)
            {
                        m_strLastError = ("VerifyServerCertificate Failed (B) -
Insufficient Memory.");
```

```
                        m_nLastError= SSPI_CERT_E_CRITICAL;
                        return false;

            }
            cchServerName = MultiByteToWideChar(CP_ACP, 0, strServerName, -1,
pwszServerName, cchServerName);
            if(cchServerName == 0 )
            {
                        m_strLastError = ("VerifyServerCertificate Failed (C) -
The target principal name is incorrect.");
                        m_nLastError= SSPI_CERT_E_CRITICAL;
                        if(pwszServerName)
                                    ::LocalFree(pwszServerName);
                        return false;
            }



            PCCERT_CONTEXT pServerCertContext = 0;
            SECURITY_STATUS secStatus = m_pSSPI->QueryContextAttributes(
                                                            &m_hContext,

SECPKG_ATTR_REMOTE_CERT_CONTEXT,

&pServerCertContext);
            if(secStatus != SEC_E_OK)
            {
                        TRACE(TEXT("**** Error 0 %d reading
SECPKG_ATTR_REMOTE_CERT_CONTEXT\n"), secStatus);
                        m_strLastError.Format("VerifyServerCertificate Failed
(C) - Status = %d",secStatus);
                        m_nLastError= SSPI_CERT_E_CRITICAL;
                        if(pwszServerName)
                                    ::LocalFree(pwszServerName);
                        return false;
            }
#ifdef _DEBUG
    DisplayCertChain(pServerCertContext, FALSE);
#endif


            CERT_CHAIN_PARA          ChainPara;
    PCCERT_CHAIN_CONTEXT     pChainContext = NULL;
    LPSTR rgszUsages[] = {  szOID_PKIX_KP_SERVER_AUTH,
                            szOID_SERVER_GATED_CRYPTO,
                            szOID_SGC_NETSCAPE };
    DWORD cUsages = sizeof(rgszUsages) / sizeof(LPSTR);


    //
    // Build certificate chain.
    //

    ZeroMemory(&ChainPara, sizeof(ChainPara));
    ChainPara.cbSize = sizeof(ChainPara);
```

```
    ChainPara.RequestedUsage.dwType = USAGE_MATCH_TYPE_OR;
    ChainPara.RequestedUsage.Usage.cUsageIdentifier     = cUsages;
    ChainPara.RequestedUsage.Usage.rgpszUsageIdentifier = rgszUsages;

    if(!CertGetCertificateChain(
                            NULL, // default chain engine
                            pServerCertContext,
                            NULL,// current system time
                            pServerCertContext->hCertStore,
                            &ChainPara,
                            0, // no flags
                            NULL,
                            &pChainContext))
    {
                        TRACE(TEXT("**** Error 0x%x CertGetCertificateChain\n"),
GetLastError());
                        m_strLastError.Format("CertGetCertificateChain Failed
(D) Errod %d",GetLastError());
                        m_nLastError= SSPI_CERT_E_CRITICAL;
                        if(pServerCertContext)

::CertFreeCertificateContext(pServerCertContext);
                        if (pChainContext)
                                    ::CertFreeCertificateChain(pChainContext);
                        if(pwszServerName)
                                    ::LocalFree(pwszServerName);
                        return false;
    }

    //
    // Validate certificate chain.
    //
            HTTPSPolicyCallbackData  polHttps;
    CERT_CHAIN_POLICY_PARA   PolicyPara;
    CERT_CHAIN_POLICY_STATUS PolicyStatus;

    ZeroMemory(&polHttps, sizeof(HTTPSPolicyCallbackData));
    polHttps.cbStruct           = sizeof(HTTPSPolicyCallbackData);
    polHttps.dwAuthType         = AUTHTYPE_SERVER;
    polHttps.fdwChecks          = 0;
    polHttps.pwszServerName     = pwszServerName;

    memset(&PolicyPara, 0, sizeof(PolicyPara));
    PolicyPara.cbSize            = sizeof(PolicyPara);
    PolicyPara.pvExtraPolicyPara = &polHttps;

    memset(&PolicyStatus, 0, sizeof(PolicyStatus));
    PolicyStatus.cbSize = sizeof(PolicyStatus);

    if(!CertVerifyCertificateChainPolicy(
                            CERT_CHAIN_POLICY_TLS,
                            pChainContext,
                            &PolicyPara,
                            &PolicyStatus))
    {
```

```
                        TRACE(TEXT("**** Error 0x%x VerifyCertificateChain\n"),
GetLastError());
                        m_strLastError.Format("VerifyCertificateChain Failed (E)
- Status = %d",GetLastError());
                        m_nLastError= SSPI_CERT_E_CRITICAL;
                        if(pServerCertContext)

::CertFreeCertificateContext(pServerCertContext);
                        if (pChainContext)
                                    ::CertFreeCertificateChain(pChainContext);
                        if(pwszServerName)
                                    ::LocalFree(pwszServerName);
                        return false;
    }


Public private key generation

bool CCryptoSynchro::StartCrypto()
{
CSingleLock sl(m_pLockable,TRUE);
            DWORD rv;
            CString logString ("Error %x  %s Attempting to Start  Crypto");
            LOGIT ( "Trying ::CtAcCt('A')");

if(!::CryptAcquireContext(&m_hCryptProv,KEY_CONTAINER_NAME,MS_ENHANCED_PROV,PROV
_RSA_FULL,0))
            {
                        rv = ::GetLastError();
                        LOGIT ( "Failed ('A') with [0x%x]
%s",rv,CDSI::LookupSystemError(rv));
                        if( rv == NTE_BAD_KEYSET || rv == NTE_BAD_KEY_STATE)
                        {
                                    LOGIT("NTE_BAD_KEYSET and NTE_BAD_KEY_STATE
handler");
                                    LOGIT ( "Trying
::CryptAcquireContext('B')");

if(!::CryptAcquireContext(&m_hCryptProv,KEY_CONTAINER_NAME,MS_ENHANCED_PROV,PROV
_RSA_FULL,CRYPT_NEWKEYSET ) )
                                    {
                                                rv = ::GetLastError();
                                                LOGIT ( "Failed ('B') with
[0x%x] %s",rv,CDSI::LookupSystemError(rv));
                                                //LOGIT("NTE_BAD_KEYSET and
NEWKEYSET Failed attempting to delete Keyset...");
                                                // delete the key set if
possible and try again
                                                LOGIT ( "Trying
::CryptAcquireContext('C')");

if(::CryptAcquireContext(&m_hCryptProv,KEY_CONTAINER_NAME,MS_ENHANCED_PROV,PROV_
RSA_FULL,CRYPT_DELETEKEYSET ) )
                                                {
```

```
                                                                LOGIT ( "Success on
Trying ::CryptAcquireContext('C')");
                                                                LOGIT("Deleted
Keyset ...");
                                                                LOGIT ( "Trying
::CryptAcquireContext('D')");

if(!::CryptAcquireContext(&m_hCryptProv,KEY_CONTAINER_NAME,MS_ENHANCED_PROV,PROV
_RSA_FULL,CRYPT_NEWKEYSET ) )
                                                                {
                                                                        rv =
::GetLastError();
                                                                        LOGIT (
"Failed ('D') with [0x%x] %s",rv,CDSI::LookupSystemError(rv));

LOGIT(logString + CString ("2"),rv,CDSI::LookupSystemError(rv) );
                                                                        return
false;
                                                                }
                                                                else
                                                                {
                                                                        LOGIT (
"Success on Trying ::CryptAcquireContext('D')");
                                                                }
                                                        }
                                                        else
                                                        {
                                                                rv =
::GetLastError();
                                                                LOGIT ( "Failed
('C') with [0x%x] %s",rv,CDSI::LookupSystemError(rv));
                                                                LOGIT(logString
,rv,CDSI::LookupSystemError(rv) );
                                                                LOGIT("Unable to
Delete Keyset...");
                                                                return false;
                                                        }
                                        }
                                        else
                                        {
                                                LOGIT ( "Success on Trying
::CryptAcquireContext('B')");
                                        }
                                }
                                else
                                {
                                        LOGIT("Not Handled When Failure is not
NTE_BAD_KEYSET and is not NTE_BAD_KEY_STATE");

                                }
                        }
                        else
                        {
                                LOGIT ( "Success on Trying ::CtAcCt('A')");
                        }
```

```
            LOGIT ( "Trying ::CtGtUKy('GA')");

if(!::CryptGetUserKey(m_hCryptProv,AT_KEYEXCHANGE,&m_hXChangePublicKey))
            {
                        rv = ::GetLastError();
                        LOGIT ( "Failed ('GA') with [0x%x]
%s",rv,CDSI::LookupSystemError(rv));
                        if (rv == NTE_NO_KEY)
                        {
                                    LOGIT("Failure is NTE_NO_KEY");
                                    LOGIT ( "Trying ::CryptGenKey('GB')");

if(!CryptGenKey(m_hCryptProv,AT_KEYEXCHANGE,0,&m_hXChangePublicKey))
                                    {
                                                rv = ::GetLastError();
                                                LOGIT ( "Failed ('GB') with
[0x%x] %s",rv,CDSI::LookupSystemError(rv));
                                                LOGIT(logString
,rv,CDSI::LookupSystemError(rv) );
                                                return false;
                                    }
                                    else
                                    {
                                                LOGIT ( "Success on Trying
CryptGenKey('GB')");
                                    }
                        }
                        else
                        {
                                    LOGIT("Not Handled When Failure is not
NTE_NO_KEY");
                        }
            }
            else
            {
                        LOGIT ( "Success on Trying CtGtUKy('GA')");
            }

            // get the length of the exported blob
            DWORD dwLen(0);
            if(!CryptExportKey(m_hXChangePublicKey,0,PUBLICKEYBLOB,0,0,&dwLen))
            {
                        rv = ::GetLastError();
                        LOGIT(logString ,rv,CDSI::LookupSystemError(rv) );
                        return false;
            }
            // get the blob
            BYTE * pBlob =  new BYTE[dwLen];

if(!CryptExportKey(m_hXChangePublicKey,0,PUBLICKEYBLOB,0,pBlob,&dwLen))
            {
                        rv = ::GetLastError();
                        LOGIT(logString ,rv,CDSI::LookupSystemError(rv) );
                        delete [] pBlob;
```

```
                    return false;
            }
            // convert it to ascii
           m_strPublicKeyBlob       =          CString(DC1) +
t.BytesToCString(dwLen,pBlob) + CString(DC2);
            delete [] pBlob;
```