NVC-SDK

Generic SDK for encrypting/decrypting of various data types

Table of content:

1.) Overview:

NVC-SDK consists of:

NVCDLL - a library supplying necessary functions for data decryption and key management.

NVsample – a VC++ sample application

Installation

Install sample Data CD provided by Nautische Veröffentlichung, this installs the DLL and the necessary keys.

Copy NVsample project to a folder of your choice.

How it works:

BSB points to .EAP instead of .KAP files. The EAP file structure is identical with the original KAP file. Each scan line of image data is encrypted and have to be decrypted with the nvc_decrypt_block() function before they are decompressed by the bsb compressor.

Load DLL and get function Pointers.

Authorize your Software with nvc_init_dll()

Obtain a decryption handle for filename with nvc_getkeyhandle().

Decrypt data with nvc_decrypt_block().

3.) Functions exported by nvcdll.dll

3.1. Using the DLL

Check the registry for HKEY_LOCAL_MACHINE \Software\NV-Filedatabase\nvcdll for the location of the DLL. If the key exists load the DLL by calling LoadLibrary() system call. After successfully loading the DLL retrieve the function pointers for nvc_init_dll, nvc_getkeyhandle, nvc_decrypt_block

See nvsample for details

3.2. nvc_init_dll

Initializes the DLL and authorizes your software with the supplied callback function.

```
char* nvc_init_dll(
        char* softwarename,
        void* (callback)(unsigned long*)
        )
```

return: version information, NULL if function fails

nvc_init_dll initializes the DLL and the available key handles. This function has to be called at application startup. Return value is a version string or NULL if function fails. This function allocates approximately 200 Bytes per installed product regardless of the number of files which are installed.

The DLL uses the callback function to authenticate your software. The callback function must use your key for authentication.

3.3. nvc_getkeyhandle

```
int nvc_getkeyhandle(const char* filename)
```

in: filename i.e. H5005_1.EAP
return: handle of the key required for decryption, negative on error

nvc_getkeyhandle() retrieves a handle needed for decryption of the given filename. The filename must be fully qualified. ie c:\bsbchart\chartkit\H5005_1.EAP

3.4 nvc_decrypt_block

```
int nvc_decrypt_block(int keyhandle, int numberofbytes,
                                        const void* inbuffer, void *outbuffer)
```

in: keyhandle returnd by nvc_getkeyhandle()
in: numberofbytes in inbuffer/space in outbuffer. In place decryption is possible (inbuffer=outbuffer).
return: number of decrypted bytes in outbuffer, negative on error.

nvc_decrypt_block() decrypts an given piece of data. This must be the same amount of data used in the encryption process.

3.) Appendix A – Sample Application

```
// nvsample.cpp : Definiert den Einstiegspunkt für die Konsolenanwendung.
//

#include "stdafx.h"

// START NVC
// nvc_return codes
#define CXERR_DATABASE_ERROR -1
#define CXERR_FILE_NOT_IN_DATABASE -2
#define CXERR_FILE_NOT_FOUND -3
#define CXERR_INVALID_PARAMETER -4
#define CXERR_INVALID_SEEK -5
#define CXERR_INVALID_KEY -6
#define CXERR_NO_INSTALLATION -7
#define CXERR_TO_MANY_OPEN_FILES -8
#define CXERR_INVALID_HANDLE -9
#define CXERR_OUT_OF_MEMORY -10
#define CXERR_CANNOT_WRITE_TO_FILE -11

// nvc_io functions
typedef char* (*fnvc_init_dll)(char* appkey,void (func)(unsigned long*));
typedef int (*fnvc_getkeyhandle)(const char* filename);
typedef int (*fnvc_decrypt_block)(int handle,int size,const void
*inbuff,void *outbuff);

// global function pointers
fnvc_init_dll nvc_init_dll;
fnvc_getkeyhandle nvc_getkeyhandle;
fnvc_decrypt_block nvc_decrypt_block;

// swap longword
inline long longswap(long w)
{
  long ret;
  char *a=(char*)&w;
  char *b=(char*)&ret;
  b[0]=a[3];
  b[1]=a[2];
  b[2]=a[1];
  b[3]=a[0];
  return ret;
}

// Decrypt BSB version 3.0
// only the scanlines are encrypted
// header and directory remains unchanged
//
void sample_decrypt_bsb(char *infile, char *outfile)
{
  char *bsb;
  struct _stat st;

  // stat file
  if (_stat(infile,&st)) {
    printf("Cannot stat %s\n",infile);
    exit(1);
  }

  // alloc memory
  bsb=(char*)malloc(st.st_size);
```

```
  if (!bsb) {
    printf("Out of Memory [%s] %d bytes\n",infile,st.st_size);
    exit(1);
  }

  // read file into memory
  int u=_open(infile,O_BINARY | O_RDONLY);
  if (_read(u,bsb,st.st_size) != st.st_size) {
    printf("Cannot Read [%s]\n",infile,st.st_size);
    exit(1);
  }
  _close(u);

  // get decryptionhandle for infile
  int decrypthandle=nvc_getkeyhandle(infile);
  if (decrypthandle < 1) {
    printf("File [%s] not encrypted or in not database\n",infile);
    exit(1);
  }

  // scanline directory
  int directorystart=longswap(*(int*)(bsb+st.st_size-4));
  int *directory=(int*)(bsb+directorystart);
  int chunkcount=(st.st_size-directorystart-4)/4;
  printf("%s %d rows\n",outfile,chunkcount);

  // decrypt chunk by chunk (normaly one chunk should be one row in BSB)
  for (int y=0; y < chunkcount; y++) {
    int d0=longswap(directory[y]);
    int d1=longswap(directory[y+1]);
    int blksize=d1-d0;
    char tmp[0x10000];  // temporary scanline buffer
    memcpy(tmp,bsb+d0,blksize); // copy scanline
    nvc_decrypt_block(decrypthandle,blksize,tmp,bsb+d0); // decryt to
memory - avoid inplace decryption
  }

  // write file from memory to disk
  u=_open(outfile,O_BINARY | O_RDWR | O_TRUNC | O_CREAT,0666);
  if (_write(u,bsb,st.st_size) != st.st_size) {
    printf("Cannot Write [%s]\n",outfile,st.st_size);
    exit(1);
  }
  _close(u);
  free(bsb);
}

// Sample for decrypting an AES file
// AES files are encrypted in 512 Bytes blocks
// using a CBC block cipher, so you CANNOT
// seek to to a arbitary position in the file.
// You MUST decyrpt each 512 byte block in one step
// unlese you use ECB Mode (16 Bytes granularity)

void sample_decrypt_aes(char *infile,char* outfile)
{

  // get decryptionhandle for infile
  int decrypthandle=nvc_getkeyhandle(infile);
  if (decrypthandle < 1) {
    printf("File [%s] not encrypted or in not database\n",infile);
    exit(1);
  }
```

```
  // decrypt file
  HANDLE h,d;
  h=CreateFile(infile,GENERIC_READ,0,0,OPEN_EXISTING,0,0);
  d=CreateFile(outfile,GENERIC_WRITE,0,0,CREATE_ALWAYS |
TRUNCATE_EXISTING,0,0);
  if (h != INVALID_HANDLE_VALUE) {
    char inbuffer[512],outbuffer[512];
    DWORD bytesread;
    do {
      ReadFile(h,inbuffer,512,&bytesread,0);
      nvc_decrypt_block(decrypthandle,bytesread,inbuffer,outbuffer);
      WriteFile(d,outbuffer,bytesread,0,0);
    } while (bytesread==512);
    CloseHandle(d);
    CloseHandle(h);
  }
}

// callback function to authorize your software key.

void nvc_dll_authcheck(unsigned long *Data)
{
  // replace this entry with the provided key
  unsigned long Key[]=  { 0,0,0,0x77034680 };
  // xtea cypher
  register unsigned long delta, sum;
  short    cnt;
  sum   = 0;
  delta = 0x9E3779B9;
  cnt   = 32;
  while(cnt-- > 0) {
    Data[0] += ((Data[1]<<4 ^ Data[1]>>5) + Data[1]) ^ (sum + Key[sum&3]);
    sum += delta;
    Data[1] += ((Data[0]<<4 ^ Data[0]>>5) + Data[0]) ^ (sum + Key[sum>>11 &
3]);
  }
}

int main(int argc, char** argv)
{

  // Get path of NVCDLL.DLL from Registry
  HKEY hkey;
  int r=RegOpenKeyEx(HKEY_LOCAL_MACHINE,"Software\\NV-
Filedatabase",0,KEY_READ,&hkey); // Open key for NV-Filedatabase
  DWORD len=256;
  char nvcdll[256];
  r=RegQueryValueEx(hkey,"nvcdll",0,0,(BYTE*)nvcdll,&len); // query for
nvcdll
  RegCloseKey(hkey); // close registry
  if (r) return 1; // do nothing if NV Software is not installed

  // Load DDL
  HMODULE plg=LoadLibrary(nvcdll);
  if (!plg) return 2; // cannot load dll

  // get function pointers from DLL
  nvc_init_dll=(fnvc_init_dll)GetProcAddress(plg,"nvc_init_dll");
  if (!nvc_init_dll) return 2; // invalid dll


nvc_getkeyhandle=(fnvc_getkeyhandle)GetProcAddress(plg,"nvc_getkeyhandle");
```

```
  if (!nvc_getkeyhandle) return 2; // invalid dll


nvc_decrypt_block=(fnvc_decrypt_block)GetProcAddress(plg,"nvc_decrypt_block
");
  if (!nvc_decrypt_block) return 2; // invalid dll

  // Init DDL with *YOUR* key
  char *init=nvc_init_dll("Application Name",nvc_dll_authcheck);
  if (!init) return 2; // Something went wrong - or no data for your key
  printf("%s\n",init);

  // decrpyt AES to stdout
  if (argc==4 && *argv[1]=='a') sample_decrypt_aes(argv[2],argv[3]);

  // decrpyt BSB  to file
  if (argc==4 && *argv[1]=='b') sample_decrypt_bsb(argv[2],argv[3]);

  // release Dll
  FreeLibrary(plg);

}
```