

User's Guide

Table of Contents

1. Introduction.....	2
1.1 Scope.....	2
1.2 References	3
1.3 SDK composition	3
1.4 Program session.....	4
2. The Basic Interface Structures	4
2.1 Decoder options.....	4
2.2 Image info	4
2.3 Symbol info	5
2.4 The Constants	6
2.5 Type definitions.....	7
3. The Interface Procedures and Functions	8
3.1 Connect_DM_Decoder.....	8
3.2 Disconnect_DM_Decoder	8
3.3 Create_DM_Options	9
3.4 Delete_DM_Options	9
3.5 DecodeDM_Bits	9
3.5.1 GetDM_ImageInfo	10
3.5.2 GetDM_Info	10
3.6 Example of the Library usage in Windows OS (Sample_VC_D)	10
4. C# DMPS-Decoding Demo application - GUI.....	14
5. Authentication / Encryption Vetting.....	17
6. Appendix 1. Protection Concept Description - Digital signature	19
7. Appendix 2. Authentication Signatures and Encryption.....	22

Data Matrix Protection Suite - Decoding SDK

1. Introduction.

Data Matrix Protection Suite is an easy-to-use Software Development Kit (SDK) that lets you not only to encode/decode text information into/from a Data Matrix symbol, but to digitally protect this symbol from counterfeiting or tampering with, as well. With **DM Protection Suite**, protecting your products and documents has never been quicker or easier!

DMPS consists of the 2 software packages – encoding (**DMPS_E**) and decoding (**DMPS_D**). The Encoding software has a proprietary built-in mechanism allowing either to create digital signature (enable Authentication) to the encoded Data Matrix symbol or to encrypt it. The Decoding software decrypts the symbol and/or checks it for authenticity while extracting the information encoded into it.

Depending on the application, the packages can be used either together or separately. In a Supply Chain application, for example, the encoding software can be used on the “manufacturing end” and the decoding software – on “receiving end”.

1.1 Scope

This document is applicable to the **Data Matrix Protection Suite - Decoding SDK (DMPS_D)**.

SDK is notated as **DMPS_D_v.x.x**.

Library interface is built for Windows (XP ...10)/64. Windows 32-bit and Linux 32/64 versions are also available per customer request.

The library is designed to:

- decode data encoded into Data Matrices ECC200 in accordance with ISO/IEC 16022 Symbology specification; and
- authenticate Data Matrix symbol if it was enabled for authentication utilizing **DMPS_E** encoding software (User ID/**Authentication key**); and/or
- decrypt Data Matrix symbol if it was encrypted utilizing **DMPS_E** software (Product ID/**Encryption key**).

DMPS_D is based on the latest 2DTG’s Data Matrix Decoding library and can be used both for the DM protection purposes, as it’s described in this User’s Guide, and regular decoding.

DMPD_D is capable to process DM symbols enabled with up to 16 Authentication keys at the same time.

Data Matrix Protection Suite - Decoding SDK

1.2 References

- ISO/IEC 16022 - Symbology specification - Data Matrix
- U.S. Patent No.: 8,297,510 B1 “Mathematical method of 2D barcode authentication and protection for embedded processing”
- 2DTG User’s Guide “**Data Matrix Decoding SDK (Professional, DPM, Enterprise editions)**”

1.3 SDK composition

Trial/Evaluation DMPS_D SDK package – fully functional 30-days trial version downloadable from 2DTG site:

- Windows DLL (**DM_EP_64.dll**) designed to perform Data Matrix decoding, authentication and decryption.
- C# source code of the Demo program **Sharp_DMPS.exe** and C++ (MSVC 2017) source code of the Demo program **Demo_MSVC_DMPS.exe**, illustrating the DLL usage.
- “**ReadMe_How-to-use-DEMO-Samples**”– description of the program evaluation (includes: DEMO UserID (**Authentication key**), DEMO Product ID (**Encryption key**)).
- **DMPS_Samples**:
 - NotProt.bmp (not protected)
 - USig.bmp (protected with User ID - Data Matrix Authentication is enabled)
 - PSig.bmp (protected with Product ID - Data Matrix Encryption)
 - USigPSig.bmp (protected both with User ID and Product ID - combined Data Matrix authentication and encryption)

Operational package includes:

- Licensed DMPS_D SDK package
- Licensed DMPS_E SDK – includes 1 (one) free Development license
- Unique Authentication key/UserID – 2DTG can provide multiple IDs per customer request

OEM Operational package includes:

- Licensed DMPS_D SDK package
- Licensed DMPS_E SDK – includes 5 (five) free Development licenses
- Unique Authentication key/UserID – 2DTG can provide multiple IDs per customer request
- Authentication key/UserID Generator (per customer request)

Data Matrix Protection Suite - Decoding SDK

1.4 Program session

Typical program session looks as follows:

Step 1. Connect decoder

Step 2. Create and set decoder options

Loop

Step 3. Capture/read bitmap image

Step 4. Process image

Step 5. Request image and symbols info

... // further application-specific data processing and interaction with user

End Loop

Step 6. Delete decoder options

Step 7. Disconnect decoder.

2. The Basic Interface Structures

The library includes the following structures:

structTDM_OptMode - the set of decoder options,
structTDM_ImageInfo - features of decoded image,
structTDM_Info - features of decoded symbols,
structTDM_Quality - Quality Parameters of decoded symbols.

2.1 Decoder options

```
/// decoder option modes
structTDM_OptMode
{
intmaxDMCount;    //!< from 1 to 100. 1 by default
  int cellColor;    //!< CL_ANY by default
  int mirrorMode;   //!< MM_NORMAL by default
  int speedMode;    //!< SP_ROBUST by default
  int qualityMask;  //!< DM_QM_NO by default
intlabelMode;     //!< LM_NORMAL by default
inttimeOut;       //!
```

2.2 Image info

```
/// results of decoding the whole Image
structTDM_ImageInfo
```

Data Matrix Protection Suite - Decoding SDK

```
{
intDMCount;          //!< number of well decoded symbols within image
intRejectionReason; //!< not DM_RR_OK if no one matrix has been well decoded
intBreakReason;     //!< 0 - normal termination, 1 - termination by time-out
int AU_Validation;  //!< = 1 AUTHENTICATION File is Valid
};
```

ImageInfo.DMCount = 1 if any Rectangle-shaped object was detected in image.

It happens if

RejectionReason = DM_RR_OK,

RejectionReason = DM_RR_BYCRIT,

RejectionReason = DM_RR_REEDSOLOMON.

If DMCount = 1 the rectangle Corners and some of Quality Parameters are defined.

BreakReason let us know whether the time out or user break happened (for embedded platforms only).

2.3 Symbol info

Each decoded symbol is described by the following structures:

```
/// Data Matrix Quality Parameters
structTDM_Quality
{
    float symbol_contrast;
    float axial_nonuniformity;
    float grid_nonuniformity;
    float fixed_pattern_damage;      //!< the aggregate grade
    float unused_error_correction;
    float vertical_print_growth;
    float horizontal_print_growth;

    float symbol_contrast_grade;
    float axial_nonuniformity_grade;
    float grid_nonuniformity_grade;
    float fixed_pattern_damage_grade;
    float unused_error_correction_grade;
    float modulation_grade;
    float decode_grade;              //!< 4 if DM was successfully decoded
    float overall_grade;             //!< minimum of grades
};
```

```
/// result of decoding of each Data Matrix symbol in image
structTDM_Info
{
    float          rowcols[8];      //!< symbol corner coordinates
    short intpchlen;                //!< length of decoded byte array
    unsigned char* pch;             //!< pointer to that array
    short intRSErr;                //!< number of Reed Solomon errors
    short intVDim, HDim;           //!< vertical and horizontal dimensions of Data
Matrix
```

Data Matrix Protection Suite - Decoding SDK

```
    unsigned char  saTotalSymbolsNumber //!< structured append: total number of
matrices
                                ,saSymbolPosition    //!< current matrix index
                                ,saFileID1           //!< file identifier 1
                                ,saFileID2;         //!< file identifier 2
bool    mirrored;                //!< true if mirrored Data Matrix
booldotpeenstage;              //!< true if dot peen Data Matrix
    unsigned char matrixcolor;    //!< detected color of Data Matrix
    int    UserSign;              //!< OK, N/A, UID-Incorrect, Failure,
    int    ProductSign;          //!< OK, N/A, PID-Incorrect, Failure
TDM_Quality    quality;          //!< symbol Quality Parameters
};
```

2.4 The Constants

```
enum CELL_COLOR{
    CL_BLACKONWHITE = 1,
    CL_WHITEONBLACK = 2,
    CL_ANY          = 3
};

enum MIRROR_MODE{
    MM_NORMAL    = 1,
    MM_MIRROR    = 2,
    MM_ANY       = 3
};

enum DECODER_SPEED{
    SP_ROBUST    = 0,
    SP_FAST      = 1
};

enum LABEL_MODE{
    LM_STANDARD = 0,          //!<-ISO 16022
    LM_DOTPEEN  = 1,
    LM_FAX      = 2
};

///< \enum QUALITY_MASK bits of mask:
enum QUALITY_MASK{
    DM_QM_NO          = 0x0000,
    DM_QM_AXNU       = 0x0001,
    DM_QM_PRGR       = 0x0002,
    DM_QM_SYMCTR     = 0x0004,
    DM_QM_CELLINFO   = 0x0008,
    DM_QM_ALL        = 0x7FFF
};

enum FILTER_MODE{
    FM_NON          = 0, //!< no filter
    FM_SHARP1       = 1, //!< first sharpening filter
    FM_SHARP2       = 2, //!< second sharpening filter
    FM_SHARPMASK    = 3  //!< Mask filter
};
```

Data Matrix Protection Suite - Decoding SDK

```
enum SIGNATURE_MODE{
    DM_SM_NO           = 0,
    DM_SM_USER        = 1,    //!< 0x01 User Signature
    DM_SM_PRODUCT     = 2,    //!< 0x02 Product Signature
    DM_SM_ALL         = 3     //!< 0x03
};

//Image Errors
enum UID_FILE_VALIDATION{
//-----
    AU_OK             = 0,
    AU_NA             = 1,    // Not Applicable
    AU_ERR_UID_NF     = 3,    // File "User_*.txt" not found
    AU_ERR_HEX        = 4,    // Unresolved hex digit in UserID
    AU_ERR_UID        = 5,    // Wrong User ID structure/length
};

// Symbol Errors
enum DM_SYMBOL_VALIDATION{
    DM_AU_OK          = 0,
    DM_AU_NA          = 1,    // Not Applicable
    DM_AU_FAILURE     = 2,    // Signature failure
    DM_AU_ERR_PID     = 7    // Wrong Product ID structure/length
};

enum DM_REJECTION_REASON{
    DM_RR_OK          = 0,
    DM_RR_NON         = 1,
    DM_RR_NODATAMATRIX = 2,
    DM_RR_BYCRIT      = 3,
    DM_RR_REEDSOLOMON = 5,
    DM_RR_NOMEMORY    = 99,
    DM_RR_UNKNOWN     = 100,
    DM_RR_DISCONNECTED = 200
};

enum DM_BREAK_REASON{
//-----
    DM_ALL_INSPECTED = 0    //!< no breaks occurred
, DM_TIMEOUT        = 1    //!< termination by time out
};
```

2.5 Type definitions

```
typedef void*          PDM_Decoder;    //!< handler of Data Matrix Decoder
typedef void*          PDM_Options;    //!< handler of Decoder Options
typedefTDM_ImageInfo* PDM_ImageInfo;  //!< pointer to Image Info
typedefTDM_Quality*   PDM_Quality;    //!< pointer to symbol Quality
typedefTDM_Info*      PDM_Info;       //!< pointer to symbol Info
typedef unsigned char* TRow;          //!< pointer to bitmap line

// The function creates Data Matrix Decoder and returns Decoder handler
typedefPDM_Decoder (stdcall *TConnect_DM_Decoder)(intmaxrow, intmaxcol);
```

Data Matrix Protection Suite - Decoding SDK

```
/// The function destroys Data Matrix Decoder
typedef void          (stdcall *TDisconnect_DM_Decoder) (PDM_Decoder&pDecoder);

/// The function creates Decoder Options and returns Options handler
typedefPDM_Options (stdcall *TCreate_DM_Options) (PDM_DecoderpDecoder,
TDM_OptModeoptmode);

/// The function destroys Decoder Options
typedef void          (stdcall *TDelete_DM_Options) (PDM_Options&pOptions);

/// The function decodes array ppbits with given Options
typedefint (stdcall *TdecodedM_Bits) (PDM_OptionspOptions, introwcount,
intcolcount, TRow* ppbits);

/// The function returns the ImageInfo of last decoded Image
typedefPDM_ImageInfo (stdcall *TGetDM_ImageInfo) (PDM_OptionspOptions);

/// The function returns the DM_Info(dmNum)
typedefPDM_Info      (stdcall *TGetDM_Info) (PDM_OptionspOptions, intdmNum);
```

3. The Interface Procedures and Functions

Description of the interface procedures is below.

3.1 Connect_DM_Decoder

PDM_DecoderConnect_DM_Decoder (intmaxrowcount, intmaxcolcount);

Description.

Function generates new instance of class encapsulating the decoder functionality.

Parameters.

Maximum of horizontal and vertical image sizes.

Return value.

Pointer to decoder in success, or NULL otherwise.

3.2 Disconnect_DM_Decoder

void Disconnect_DM_Decoder (PDM_Decoder&pDecoder);

Description.

Procedure destroys decoder class and frees memory.

Parameter.

Data Matrix Protection Suite - Decoding SDK

Pointer to decoder. Decoder should be connected.

3.3 Create_DM_Options

Class TDM_Options encapsulates the decoder options and methods of image processing and inspection.

PDM_OptionsCreate_DM_Options (PDM_DecoderpDecoder,TDM_OptModeoptmode);

Description.

Function generates new class to decode image with certain options.

Parameters.

- Pointer to decoder.
- Pointer to option modes that specify the way of image processing

Return value.

The handler that provides decoding of the image with desirable options.

3.4 Delete_DM_Options

void Delete_DM_Options (PDM_Options&pOptions);

Description.

The function destroys a handler.

Parameters.

- Handler of decoder with options.

3.5 DecodeDM_Bits

**intDecodeDM_Bits (PDM_Options pOptions,
int actualrowcount,
int actualcolcount,
TRow* prows);**

Description.

The function processes an image and fills Image Info and array of Symbol Infos.

Parameters.

- Handler produced by 3.3
- Number of image rows

Data Matrix Protection Suite - Decoding SDK

- Number of image columns
- Array of pointers to image rows. Every row is a byte array with 8-bit pixel intensities. (We have **typedef unsigned char* TRow;**)

Return value.

0 if no one symbol was decoded, >0 otherwise.

If the only symbol was decoded then Rejection Reason may be not DM_RR_OK.

3.5.1 GetDM_ImageInfo

PDM_ImageInfoGetDM_ImageInfo (PDM_OptionspOptions);

Description.

The function returns image info.

Return value.

Pointer to Image Info.

3.5.2 GetDM_Info

PDM_InfoGetDM_Info (PDM_OptionspOptions, intdmNum);

Description.

The function returns Data Matrix symbol info.

Parameters.

- Handler of decoder with options
 - Number (index) of decoded symbol in image.
- If no symbols were decoded we return Info about the most probable symbol location.

Return value.

Pointer to Symbol Info.

3.6 Example of the Library usage in Windows OS (Sample_VC_D)

```
//-----  
#include <stdlib.h>  
#include <stdio.h>  
#include <math.h>  
#include <string.h>  
  
#include <time.h>  
#include <Windows.h>  
  
#include "DMPPro_Types.h"
```

Data Matrix Protection Suite - Decoding SDK

```
#include "LoadBmp.cpp"

const int // the images size
    maxrowQ = 4000,
    maxcolQ = 4000;

int main(int argc, char **argv)
{
    TRow*    pbits;
    TRow    pmembers; // Image in Memory
    int      ResLoadBMP;
    int      rowcount, colcount, row, result;
    int      i, k;
    int      rr;

    PDM_Decoder pDecoder;
    PDM_Options decl;
    TDM_OptMode opt;
    TDM_ImageInfo* pdm_imageinfo;
    TDM_Info* pdm_info;

    clock_t beg, end;
    int    dmq;
    int    welldec;

    char  endrun;
    char  UID_Val[9];
    char  Res_USig[9];
    char  Res_PSig[9];

    HINSTANCE dllinstance;

    TConnect_DM_Decoder    Connect_DM_Decoder;
    TDisconnect_DM_Decoder Disconnect_DM_Decoder;
    TCreate_DM_Options     Create_DM_Options;
    TDelete_DM_Options     Delete_DM_Options;
    TDecodeDM_Bits        DecodeDM_Bits;
    TGetDM_ImageInfo      GetDM_ImageInfo;
    TGetDM_Info           GetDM_Info;

    dllinstance = LoadLibrary(L"..\\DM_DecWinPro.dll");
    if (dllinstance==NULL)
        goto doExit;

    Connect_DM_Decoder    = (TConnect_DM_Decoder
)GetProcAddress(dllinstance, "Connect_DM_Decoder");
    Disconnect_DM_Decoder = (TDisconnect_DM_Decoder
)GetProcAddress(dllinstance, "Disconnect_DM_Decoder");
    Create_DM_Options     = (TCreate_DM_Options
)GetProcAddress(dllinstance, "Create_DM_Options");
    Delete_DM_Options     = (TDelete_DM_Options
)GetProcAddress(dllinstance, "Delete_DM_Options");
    DecodeDM_Bits        = (TDecodeBitsF
)GetProcAddress(dllinstance, "DecodeDM_Bits");
    GetDM_ImageInfo      = (TGetDM_ImageInfo
)GetProcAddress(dllinstance, "GetDM_ImageInfo");
```

Data Matrix Protection Suite - Decoding SDK

```
GetDM_Info          = (TGetDM_Info
)GetProcAddress(dllinstance, "GetDM_Info");

pmembits = (TRow ) malloc(maxrowQ*maxcolQ);          // Image in Memory
pbits    = (TRow*) malloc(maxrowQ*sizeof(TRow));    // pointers to ScanLines
for (row = 0; row < maxrowQ; row++){
    pbits[row] = &pmembits[maxcolQ*row];
}
welldec = 0;
if (Connect_DM_Decoder==NULL)
    goto doExit;
pDecoder = Connect_DM_Decoder(maxcolQ,maxrowQ);
if (pDecoder != NULL){

    opt.maxDMCount = 100;
    opt.cellColor  = CL_ANY;
    opt.mirrorMode = MM_ANY;
    opt.speedMode  = SP_ROBUST;
    opt.qualityMask = DM_QM_ALL;
    opt.labelMode  = LM_STANDARD;
    opt.timeOut    = 0;
    opt.filterMode = 0;
    opt.signMode   = DM_SM_ALL;
    strcpy(opt.productID, "123");

    decl = Create_DM_Options(pDecoder, opt);

    rowcount = maxrowQ;
    colcount = maxcolQ;
    ResLoadBMP = LoadBMP ("U_123.bmp" ,pbits ,rowcount ,colcount);
    if (ResLoadBMP == 0){

        beg = clock();
        result = DecodeDM_Bits(decl,rowcount, colcount, pbits);
        end = clock();

        pdm_imageinfo = GetDM_ImageInfo(decl);
        dmq = pdm_imageinfo->DMCount;
        printf("\n DM count = %4d", dmq);
        rr = pdm_imageinfo->RejectionReason;
        if (pdm_imageinfo->AU_Validation==AU_OK)    strcpy(UID_Val, " OK ");
        else if (pdm_imageinfo->AU_Validation==AU_NA) strcpy(UID_Val, " N/A
");
        else          strcpy(UID_Val, " INVALID");
        printf("\n\n      User.id: %s", UID_Val);
        if (dmq>0){
            for (i=0; i<=dmq-1; i++){
                pdm_info = GetDM_Info(decl, i);
// signatures:
                memset(Res_USig, 0, 9);
                memset(Res_PSig, 0, 9);
                if (pdm_info->UserSign==DM_AU_OK)    strcpy(Res_USig, " OK ");
                else if (pdm_info->UserSign==DM_AU_NA)    strcpy(Res_USig, " N/A
");
                else    strcpy(Res_USig, " FAILURE");
                if (pdm_info->ProductSign==DM_AU_OK)    strcpy(Res_PSig, " OK ");

```

Data Matrix Protection Suite - Decoding SDK

```
    else if (pdm_info->ProductSign==DM_AU_NA) strcpy(Res_PSig," N/A
");
    else strcpy(Res_PSig," FAILURE");
    printf("\n --- USER SIGN = %s ---      PRODUCT SIGN = %s\n",
Res_USig, Res_PSig);
// other infos:
    if (pdm_info->pchlen>0)
        printf("\n Decoded array: %s\n",pdm_info->pch);
    printf("\n Corners: ");
    for (k=0;k<4;k++){
        printf("(%4d,%4d),    ", int(pdm_info->rowcols[2*k])
        , int(pdm_info->rowcols[2*k+1]));
    }
    if (rr == 0)
        welldec++;
    printf("\n Dimensions: %4d * %4d",
        pdm_info->VDim, pdm_info->HDim);
    printf("\n vert. & horiz. print_growth = %7.2f,%7.2f"
        ,pdm_info->quality.vertical_print_growth
        ,pdm_info->quality.horizontal_print_growth);
// Quality Parameters:
    printf("\n axial_nonuniformity (grade) = %7.2f    (%d)"
        ,pdm_info->quality.axial_nonuniformity
        ,int(pdm_info->quality.axial_nonuniformity_grade));
    printf("\n grid_nonuniformity (grade) = %7.2f    (%d)"
        ,pdm_info->quality.grid_nonuniformity
        ,int(pdm_info->quality.grid_nonuniformity_grade));
    printf("\n symbol_contrast (grade) = %7.2f    (%d)"
        ,pdm_info->quality.symbol_contrast
        ,int(pdm_info->quality.symbol_contrast_grade));
    printf("\n unused_error_corr. (grade) = %7.2f    (%d)"
        , pdm_info->quality.unused_error_correction
        , int(pdm_info->quality.unused_error_correction_grade));
    printf("\n fixed_patt_damage. (grade) = %7.2f    (%d)"
        , pdm_info->quality.fixed_pattern_damage
        , int(pdm_info->quality.fixed_pattern_damage_grade));
    printf("\n (modulation grade) =                (%d)"
        ,int(pdm_info->quality.modulation_grade));
    printf("\n (decode grade) =                (%d)"
        ,int(pdm_info->quality.decode_grade));
    printf("\n (overall grade) =                (%d)"
        ,int(pdm_info->quality.overall_grade));

    }
    }
    printf("\n time = %f\n", (end - beg) / CLK_TCK);
}else{
    printf("\n\nLoadBMP !=0\n");
}
Delete_DM_Options(dec1);
Disconnect_DM_Decoder(pDecoder);
}

printf("\n\n Well decoded images:  %d \n",welldec);
free(pbits);
free(pmembers);
```

Data Matrix Protection Suite - Decoding SDK

```
doExit:
    printf(">");
    scanf("%c", &endrun);

    return 0;
}
```

4. C# DMPS-Decoding Demo application - GUI

Application is designed to demonstrate full encoding and authentication/encryption functionality of the **DMPS_D** SDK and how to incorporate decoding DLL into the customer application.

Double-click **Sharp_DMPS.exe** (in Windows 10 you may need to run it as administrator first time you run the program) – “**Decode DMPS**” window will appear:

The screenshot displays the 'Demo DMPS' application window. At the top, there are fields for 'Decoded DM' (value: 1), 'Rejection Reason' (value: OK), and 'Time ms' (value: 1). Below these is the file path: C:\Users\vas00\Documents\2DTG\2DTG WEB 2011\Brand Protection\DMPS_Decoding_v18.01.8.2\DMPS_D_WIN64_v18.01.8.2\USigPSig.bmp. The main interface is divided into several sections:

- Options:** A list of settings including max DM Count (100), Mirror (Normal & Mirror), Decode Mode (Ultimate), Label Mode (St+Dot), Quality (Yes), Color (Any), Filter (NON), Quiet Zone (Normal), Signature (Both), Product ID (s#123456789), and UserID (*User_Company_Name*13351).
- Symbol Info:** Fields for Symbol Number (0), Symbology ID (jd1), V Dim (24), H Dim (24), Actual Color (Black), Mirrored (No), Dot Peen (No), and R-S Errors (1).
- Authentication/Decryption:** Two buttons labeled 'Authentication' and 'Decryption', both showing 'OK'.
- Decoded Data:** A central area showing a Data Matrix symbol with a green border. Below it, the text representation is '65001 - Unicode (UTF-8)' and the decoded text is '2DTG - Your optimum choice in robust performance'.
- Symbol Quality:** A table showing various quality metrics and their grades.
- Print Growth:** Fields for 'v' (0.09) and 'h' (-0.19).

Symbol Quality	Value	Grade
Axial Non-Uniformity	0.00	4
Grid Non-Uniformity	0.32	4
Symbol Contrast	100.00	4
Unused Error Correction	0.92	4
Fixed Pattern Damage	4.00	4
Modulation		4
Decode Grade		4
Overall Grade		4

Data Matrix Protection Suite - Decoding SDK

GUI looks pretty much the same way as for DM Enterprise Decoder (2DTG User's Guide "[Data Matrix Decoding SDK \(Professional, DPM, Enterprise editions\)](#)") with two additional Sections (shown in Red above) .

Decode Settings Options:

- **Max DM count** – number of Data Matrix symbols within an image (if known in advance) – default number = 100, total – 400;
- **Mirror** – Normal, Mirror, Normal&Mirror (default, if not known in advance)
- **Decode/Speed** – Ultimate+ (Default), Ultimate, Regular, Express
- **Label Mode** – Standard, Dot Peen (DPM), St+Dot (default)
- **Symbol Quality** – YES (default)/NO
- **Color** – Black, White, Any (default, if not known in advance)
- **Filter** – default – “None” (see Section 5.4 for detail)
- **Quiet Zone** – Normal (per ISO 16022), Small (default – “Normal”)

DMPS Addition:

- **Signature** - choose the option corresponding to the open symbol from the drop-down menu:
 - None
 - UserSign – **Authentication** only
 - ProdSign – **Encryption** only
 - Both – **Authentication + Encryption**
- **Product ID** - enter your **Encryption Key** (s#123456789 – evaluation example).
- **UserID** – enter UserID received from 2DTG. Up to 16 UserIDs are allowed (default data string is an evaluation example)

Overall decode info:

- **Decoded DM** – number of Data Matrix decoded in this image
- **Rejection Reason** – returns decode result:
 - “**OK**” successful decoding (DM_RR_OK = 0) **or**Error Code - in some cases decoding library can return certain error codes associated with the decoding process. They are as follows:
 - **Error Code 1** – (DM_RR_NON = 1) – no “structured formations” found within the image

Data Matrix Protection Suite - Decoding SDK

- **Error Code 2** – (DM_RR_NODATAMATRIX= 2) - no “matrix-like formations” found within the image
- **Error Code 3** – (DM_RR_BYCRIT = 3) - alternating pattern is incorrect (dark and light modules in the finder pattern do not meet alternation criteria)
- **Error Code 5** – (DM_RR_REEDSOLOMON = 5) – excessive number of Reed-Solomon error
- **Time** (ms) – total decode time

Symbol Info:

- **Symbol Number** – symbol for which the decode result is displayed (starts with number “0”) assuming multiple number of symbols in the image
- **Symbol ID** – GS1/Regular Data Matrix identifier for displayed symbol
- **V Dm, H Dm** – Data Matrix dimensions (Vertical, Horizontal) – in number of modules
- **Actual Color** – shows if the color of displayed symbol is regular or inversed
- **Mirrored** – shows if displayed symbol is mirrored or not
- **DotPeen** – shows if displayed symbol was decode using Dot Peen algorithm or Standard one
- **R-S Errors** – number of Reed-Solomon errors in displayed decoded symbol

DMPS Addition:

- **Authentication**
- **Decryption**

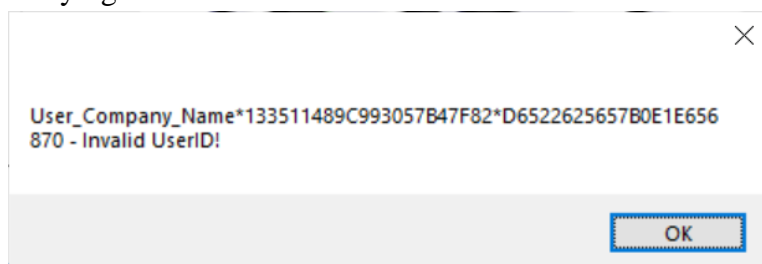
(Both fields can return 3 optional values: OK, Failed, N/A (see Section 5 for more info)).

Symbol Quality – results of the symbol quality assessment in accordance with ISO/IEC 15415

Print Growth - calculated per ISO/IEC 16022

Note:

DMPS library has a built-in mechanism for **UserID Validation**. If you see an error message like the one below, when trying to authenticate Data Matrix:







UserID should be checked for integrity and authentication process be repeated.

Data Matrix Protection Suite - Decoding SDK

5. Authentication / Encryption Vetting

DMPS D may return the following results when checking for Data Matrix Authenticity /Encryption (all symbols are from trial SDK package and all of them get correct decoding, even if Authentication failed):

Protection Level	Data Matrix Type	Data Matrix Sample	Signature Authentication	Failure Code
<u>Level “1” Protection</u> Authentication only Signature setting: UserSign	Regular Data Matrix		User: Failure	1
	Authentication enabled Data Matrix		User: OK	
<u>Level “2” Protection</u> Encryption only Signature setting: ProdSign	Encrypted Data Matrix		Product: OK	
<u>Level “3” Protection</u> Authentication + Encryption Signature setting: Both	Encrypted Data Matrix		User: Failure Product: OK	2
	Authentication enabled + encrypted Data Matrix		User: OK Product: OK	

Failure Code 1 indicates that vetted symbol was counterfeited. Most probably, original Data Matrix was captured, decoded and the obtained data were used for generating counterfeited symbols for placing them on counterfeited goods, documents, etc.

Failure Code 2 indicates that both Encryption Key and Authentication Key are compromised and need to be changed immediately.

Additional comments on vetting results:

Data Matrix Protection Suite - Decoding SDK

1. The **Product Signature (Encryption)** is always being analyzed first (if this option is “ON”) and if it returns “**Failure**” – the process stops, matrix is not decoded and no further analyzes of the **User Signature** is performed. This is a result of the predetermined sequence of creating digital signatures in the Protection mechanism - **User Signature** must always antedate the creation of **Product Signature**.
2. If **User Signature (Authentication vetting)** returned “**Failure**”, the matrix can still be decoded, indicating at the same time that it was counterfeited on the way to the Receiver.

6. Appendix 1. Protection Concept Description - Digital signature

One of the widely used concepts of data authentication is the concept of digital signature. A digital signature scheme allows one to sign an electronic message and later the produced signature can be validated by the owner of the message or by any verifier. This concept employs asymmetric cryptography and is covered by a few international and domestic standards.

Because asymmetric key algorithms are nearly always very computationally intensive direct use of this concept for barcode applications is not practical or even feasible. In most “field” applications, the barcode decoding algorithm is embedded into the digital signal processing (“DSP”) platform (for example, a scanner), having limited computational resources. Running cryptographical software on such devices would make decoding processes slow if not impossible.

DMPS employs proprietary patented symmetric key authentication mechanism that is built into an existing encoding/decoding algorithm as an optional “security feature” and is suitable for DSP platforms.

This algorithm represents modified digital signature concept and term “signature” here is used just to underline the fact that each Data Matrix, generated using this mechanism, will have the unique digital characteristics distinguishing it from any other Data Matrix symbol with the same encoded content.

DMPS provides for the two types of “digital signatures” – **User Signature (Authentication)** and **Product Signature (Encryption)**.

User Signature (referenced also as an **Authentication Key**) is used for symbol authentication. It is calculated using the encoded data sequence, matrix dimensions and shared key (referenced as “**User ID**”), subject to a key-exchange protocol between originator and receiver. **User ID** is provided as a part of 2DTG’s **DMPS_E** package. Since the **Authentication key** depends on the encoded data sequence it is different for every matrix going through the algorithm.

The resulting symbol image CAN be decoded by any commercially available decoding software. Counterfeiter most probably will not even recognize that this symbol is carrying “digital signature” with it, because it looks identical to unprotected image with the same content. However, 2DTG decoding software **DMPS_D** allows to authenticate the image by verifying its digital signature.

Product Signature (referenced also as an **Encryption Key**) is used for Data Matrix encryption. It depends on three parameters: **Product ID** (Product Serial number, Part No. or any arbitrary number - up to 100 alpha-numeric characters), **User ID** and matrix

Data Matrix Protection Suite - Decoding SDK

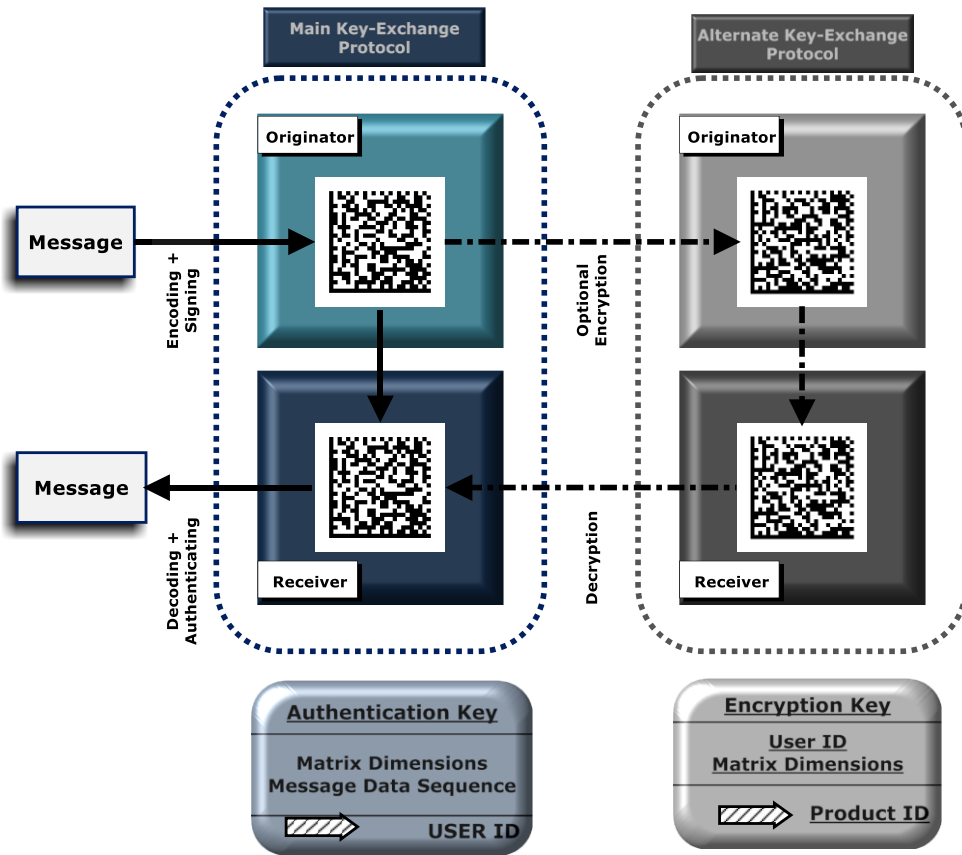
dimensions. **Product ID** represents an additional secret exchange key established within the alternate (additional) key-exchange protocol. Though this protocol can be utilized alone for Data Matrix encryption, it is designed mainly for strengthening the authentication mechanism within the main key-exchange protocol for some particularly sensitive applications. In other words, it provides one more layer of protection on top of authentication.

On the contrary to the **User Signature** option, the resulting symbol image CANNOT be decoded by any commercially available decoding software except of **DMPS D**.

Combined use of these two mechanisms provides the best Data Matrix protection. Diagram below illustrates the mechanism of “combined protection” explaining the sequence of events when both protection mechanisms are employed – digital signature and encryption.

This diagram illustrates also that protected matrix looks very similar to an original one. Without special efforts, for example, it may be difficult for anyone even to recognize that the matrix is “signed” or encrypted. Matrixes, circulating in a supply chain, for instance, are never the ideal matrixes – they always have defects, irregularities, or damages, which are randomized “by nature”. It is impossible to say without extensive investigation if some of these irregularities may have different origin. From this perspective the implemented digital signature algorithm possesses steganography features, providing additional “safety” to the whole method.

Data Matrix Protection Suite - Decoding SDK



If the process fails on either of the two protocols – matrix cannot be decoded (alternate protocol) or matrix did not pass authentication (main protocol) – it means that the system is under counterfeiting attack (note, that matrix authentication may fail even if it was successfully decoded).

Even if the Alternate protocol was compromised, it is still not enough to decrypt the matrix, because the **Encryption key** involves the “**User ID**” as one of the components for its calculation. So, the attacker must compromise the Main protocol, as well, to succeed.

In addition, even if both protocols are compromised, the attacker may not enjoy the benefit of it for a long time. Unlike the **User ID**, the **Product ID** is designed to be changeable on a regular basis, so if the problem with the protocol integrity arises it will be discovered promptly.

7. Appendix 2. Authentication Signatures and Encryption

There are, normally, two ways how symbol can be counterfeited:

Repeated encoding – original Data Matrix is captured, decoded and the obtained data are used for generating counterfeited symbols for placing them on counterfeited goods, documents, etc. At this stage the original data can also be tempered. This is the most “cost efficient” and “reliable” method for counterfeiters and that is why it is mostly widely used.

Direct copying the symbol from original (product, document, etc.) and placing it to counterfeited item – less reliable and technologically sometimes more complicated, particularly for DPM applications.

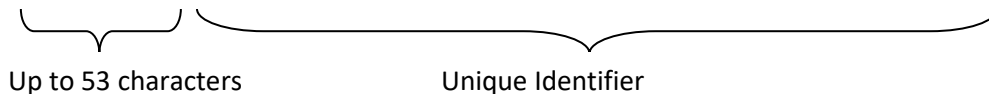
DMPS offers two protection mechanisms effectively addressing both of these counterfeiting techniques.

1. **User Signature.** This method provides a Master Company (Customer) with the ability to sign a Data Matrix symbol with its own unique digital signature so that later the produced signature can be validated by the owner of the message or by any verifier. This makes Repeated encoding useless for an intruder, though it still not protects the symbol from direct copying.

Protecting from direct copying can be effectively implemented (particularly when protecting documents) by printing Data Matrix in UV (Ultra-Violet) inks and “reading” them with specially retrofitted [UV scanners, offered by 2DTG](#).

The **user.ID** file contains the string of up to 100 alphanumeric characters as follows:

***User Name*689B18B4DEFF181276F245*033FA9F560B6830036CADC**



Main features:

- Allows for Data Matrix Symbol Authentication utilizing **DMPS_D** software.
- “Signature Algorithm” contains customer specific information – **User ID** – supplied by 2DTG in the form of text file with special syntax. **User ID** is used for calculating **Authentication key**.
- **Authentication key** depends on the encoded data sequence - so it is different for every matrix going through the algorithm.

Data Matrix Protection Suite - Decoding SDK

- Symbol can be decoded by any 3rd party decoding software but that software will NOT BE ABLE to detect **User Signature**.

Employing this technic calls for establishing a certain key-exchange protocol (referenced here as a Main Key-Exchange Protocol) between originator (Master company) and receiver (member of protected network), as it is always a case with the symmetric-key algorithms. This single secret key (**User ID**) must be shared and kept private by both originator and receiver.

2. **Product Signature.** This method provides a Master Company with the ability to encrypt Data Matrix symbol using **Product ID** as variable parameter. Since this parameter can be changed on a regular basis, both counterfeiting and copying of symbols becomes very difficult or even meaningless for an intruder.

Main features:

- Allows for Data Matrix Symbol encryption/decryption utilizing DMPS software.
- **Encryption key** depends both on **Product ID** and **User ID** even if this mechanism is used alone, without **User signature**.
- Symbol CANNOT be decoded by any 3rd party decoding software - only by 2DTG decoding software – DMPS D.

This technic provides for establishing an Alternate Key-Exchange Protocol between Originator (Master Company) and Receiver (member of protected network). The shared secret key in this case is **Product ID**. Establishing this alternate communication protocol does not mean that the main protocol is not required anymore, because **User ID** (which is communicated within the main protocol) is still required for calculating **Encryption key**.

3. **Combined Signature.** The symbol provides the highest level of protection. In this case “**User Signature**” shall be created first and “**Product Signature**” shall be applied “on top of that”. Accordingly, symbol processing at the receiving end shall be done in the reversed order as it is depicted in the block-diagram above.

The resulting symbol images - though they are almost indistinguishable for human eyes, even when compared next to each other, - feature different protection levels:

<u>Protection Levels</u>		
User Signature	Product Signature	Both Signatures
Authentication mode	Encryption mode	Combined mode
Level «1»	Level «2»	Level «3»