XDP Design Specification

# Introduction

The Microsoft Data Protection API, DPAPI (<http://msdn.microsoft.com/en-us/library/ms995355.aspx>) is a convenient API to encrypt/decrypt data on a MS Windows machine. The API offers only 2 methods:

BOOL WINAPI CryptProtectData (

DATA\_BLOB \*pDataIn,

LPCWSTR szDataDescr,

DATA\_BLOB \*pOptionalEntropy,

PVOID pvReserved,

CRYPTPROTECT\_PROMPTSTRUCT \*pPromptStruct,

DWORD dwFlags,

DATA\_BLOB \*pDataOut)

BOOL WINAPI CryptUnprotectData (

DATA\_BLOB \*pDataIn,

LPCWSTR \*ppszDataDescr,

DATA\_BLOB \*pOptionalEntropy,

PVOID pvReserved,

CRYPTPROTECT\_PROMPTSTRUCT \*pPromptStruct,

DWORD dwFlags,

DATA\_BLOB \*pDataOut)

Note that neither method takes a key to encrypt/decrypt with, instead the methods use the current user’s Windows account password as the basis to protect the input data. ‘Protect’ in this sense means encrypting the data.

The DPAPI is very convenient for encrypting information as it eliminates the most common problem developer’s face when using encryption, that is, key management. The DPAPI does however suffering from some limitations:

* Data encrypted on one machine cannot be easily transferred to another machine. This prevents the DPAPI being used to encrypt data for transit or for replicating sensitive files e.g. configuration files with credentials, across machines.
* Data can only be encrypted to the current user’s account or the machine. This limits the ability for multiple applications running under different accounts from accessing the same encrypted information. It encourages using the DPAPI to encrypt to the machine so any account can decrypt the data. This is bad practice, as compromise of any account would expose the data.
* Data cannot be conveniently encrypted to a domain account so that the encrypted data can be decrypted on any machine that the domain account user logs onto.

This document details the design of the eXtended Data Protection API (XDP API) which overcomes the above limitations of the DPAPI but is nevertheless built on the DPAPI so benefits from all of its advantages as well.

# Goals & Non-Goals

The following are the goals of XDP:

* Provide a simple API interface for developers to protect a data buffer.
* Provide the ability to protect data to any type of account; local user account, remote user machine, or domain user account.
* Provide the ability to protect data to multiple accounts.
* Provide wrapper interfaces in numerous programming languages to promote the adoption of XDP across as many developers as possible.
* Support crypto-agility (the ability to easily upgrade the cryptographic algorithms being used to protect data).
* Provide at least the same level of security as the DPAPI. This includes secure key management and data recovery.
* Not to rely on any public-key cryptography (and hence certificates).

The following are **non-goals** of XDP:

* To stop the follow accounts from decrypting:
  + Local Administrators on the machine that encrypts data.
  + Local Administrators on a non-domain machine with a user account authorised to decrypt XDP protected data. A local Administrator can impersonate a local (non-domain) user.
  + Local Administrators on a domain machine with a domain user account *that is logged on and authorised to decrypt the data*. A local Administrator can inject into a domain user process and execute code as that user.
  + Domain Administrators.
* To provide a data protection method suitable for exchanging a high volume of encrypted data.
* To allow non-domain users to encrypt to domain users (non-domain users cannot make an authenticated connection to another domain machine).

# Dependencies

XDP depends on:

* .NET Framework.
* DPAPI.

# Design Overview

An overview of the design is given in terms of the various scenarios that XDP will support.

## Local Machine Scenario

In the Local Machine scenario UserA on the local machine wants to protect data so that they can share it with UserB on the local machine. On the local machine there is an XDP Machine Service installed and running.

1. UserA calls XDP.Protect passing in UserB’s account name and the data to protect. The XDP Machine Service (MS) creates an encryption and signature key and encrypts and signs the data. The keys are then protected by the MS using the DPAPI and stored with the encrypted data and UserB’s account name in an XDPData structure. MS returns the XDPData to UserA.
2. UserA stores the XDPData somewhere accessible to UserB.
3. UserB calls XDP.Unprotect passing the XDPData structure. The MS verifies that UserB is authorised to decrypt XDPData and unprotects the encryption ad signature keys using the DPAPI and decrypts the data, checks the signature, and returns it to UserB.

## Domain Scenario

In the Domain scenario UserA from the domain wants to protect data so that they can share it with UserB from the same domain. On the user machines there is an XDP Machine Service running, on the Domain Controller there is an XDP Domain Service running.

4



UserA@domain

UserB@domain

Domain Controller (DC)

1

2

3

5

1. UserA calls XDP.Protect passing in UserB’s account name and the data to protect. The XDP Machine Service (MS) creates an encryption and signature key and securely sends the keys and UserB’s account name to the XDP Domain Service (DS) on the DC over an encrypted and authenticated (Kerberos/NTLM) connection.
2. The DS uses the DPAPI to protect the keys and signs the protected keys along with UserB’s account name. The DS returns the signature and encrypted keys to the MS. The MS returns an XDPData structure to the user (which includes the DS signature and encrypted keys).
3. UserA communicates XDPData to UserB.
4. UserB calls XDP.Unprotect passing in XDPData. The MS impersonates UserB and sends the protected keys plus signature to the DS on the DC over an encrypted and authenticated (Kerberos/NTLM) connection.
5. The DS identifies the domain account (UserB in this case) of the incoming connection making the decryption request, verifies the signature on the protected keys and that UserB is authorised to decrypt, decrypts the keys using the DPAPI, and returns the keys to the MS. The MS decrypts the data, checks the data signature, and returns it to UserB.

## Non-Domain Scenario (*delayed to later version*)

In the non-Domain scenario UserA on MacineA wants to protect data so that they can share it with UserB on MachineB. Both machines have a XDP Machine Service running.



UserA

MachineA

MachineB

UserB

1

2

3

1. UserA on MachineA calls XDPCryptProtectData passing in the UserB@MachineB account name and the data to protect. The XDP Machine Service on MachineA (MS-A) creates an encryption and signature key, impersonates UserA and sends the key and UserB@MachineB account name to the XDP Local Server on MachineB (LS-B) over an encrypted and authenticated connection (NTLM).

* **Limitation: Requires both MachineA and MachineB to have the UserA account with the same password.**
* **MachineC may quite likely have UserA on it as well and so could MITM communications between MachineA and MachineB.**

1. The MS-B DPAPI encrypts the key and links it to the UserB@MachineB account name via an XDPInternalMachineHeader and returns this to MS-A. The MS-A creates an XDPData structure and returns this to UserA.
2. UserA communicates XDPData to UserB. UserB calls XDPCryptUnprotectData passing in XDPData. The MS-B gets the identity of the caller (in this case UserB) and confirms they are authorised to decrypt the XDP Blob. The MS-B uses the DPAPI to decrypt the encryption key in the XDPInternalMachineHeader and decrypts the data and returns it to UserB.

# Design Details

## API Methods

XDP.dll gets installed in the GAC when XDP Machine Service is installed. XDP.dll exposes the following 2 methods in the XDP namespace.

### XDP.Protect

Protects the data passed in by encrypting it so only the provided list of authorized users can decrypt the data.

public static byte[] Protect(byte[] userData, List<String> authorizedUsers)

Where:

* userData - The data to encrypt.
* authorizedUsers - The list of users who can decrypt the data. The identities should be of the form:
* hostname\username or username@hostname
* hostname\groupname or groupname@hostname
* domain\username or username@domain
* domain\groupname or groupname@domain

The method returns the encrypted data in an XDPData structure.

Some of the exceptions that can be thrown are:

* System.ArgumentNullException - The userData parameter is null.
* System.ArgumentException - The userData parameter is empty.
* XDP.XDPBadParameterException - A bad parameter was encountered.
* XDP.XDPInvalidIdentityException - An invalid identity was encountered.
* XDP.XDPException - A generic exception occurred.

### XDP.Unprotect

Unprotects data protected with ProtectData if the caller is in the list of authorizedUsers.

public static byte[] Unprotect(byte[] encryptedData)

Where:

* encryptedData - The data to decrypt.

The method returns the decrypted data.

Some of the exceptions that can be thrown are:

* System.ArgumentNullException - The encryptedData parameter is null.
* System.ArgumentException - The encryptedData parameter is empty.
* XDP.XDPBadParameterException - The encryptedData parameter contained a bad parameter.
* XDP.XDPSignatureVerificationException - The encryptedData parameter did not have a valid signature.
* XDP.XDPAuthorizationException - The caller was not authorized to unprotect the data
* XDP.XDPInvalidIdentityException - An invalid identity was encountered.
* XDP.XDPException - A generic exception occurred.

## Data Formats

### XDPData

This is the format of the byte array that is returned by XDP.Protect.

struct XDPData

{

unsigned char[] Magic = “XDP”; // Magic value

byte flags; // Flags if XDPHeader/Data is compressed

unsigned int cbXDPHeader; // Length in bytes of XDPHeader

unsigned char[] XDPHeader; // All XDP information

unsigned char[] Data // Encrypted Data

}

### XDP.xsd and XDPMessages.xsd

XDP.xsd contains the definitions for all the XDP data formats, including:

* XDPHeader – Contains all other header elements
* XDPInternalCommonHeader – Contains the Encryption Algorithm, the Encryption Mode, the Encryption IV, the Signature Algorithm and the Signature of the unencrypted Data
* XDPInternalMachineHeader – Contains the hostname of the machine, XDPAuthorizedIdentities and XDPEncryptedKeys.
* XDPInternalDomainHeader – Contains the Domain name, XDPAuthorizedIdentities and XDPEncryptedKeys.
* XDPAuthorizedIdentities – The SIDs that can decrypt the data, these are relative to a Machine or Domain
* XDPEncryptedKeys –DPAPI encrypted XDPKeys.

XDPMessages.xsd contains the definitions for all the XDP message formats, including:

* XDPKeys – Contains the encryption key (XDPEncryptionKey) and signature key (XDPSignatureKey).
* XDPRequestDomainHeader – The request to the DS used during encryption. It includes XDPInternalCommonHeader, XDPAuthorizedIdentities and XDPKeys.
* XDPResponseDomainHeader – The response from the DS for a XDPRequestDomainHeader message. It includes XDPInternalDomainHeader and XDPInternalHeaderDomainSignature.
* XDPRequestDecryptionKey – The request to the DS used during decryption. It includes XDPInternalCommonHeader, XDPInternalDomainHeader and XDPInternalHeaderDomainSignature.
* XDPResponseDecryptionKey – The response from the DS for a XDPRequestDecryptionKey message. It includes XDPKeys.
* XDPException – The response to any request that causes an exception on the DS.

### Design of XDPData Format

A simplified graphical layout can be seen on the next page.

Green Border = Signed using XDPSignatureKey

Red Box Background = Encrypted using DPAPI

Purple Box Background = Encrypted using XDPEncryptionKey

Blue Box Background = Not encrypted

We can also represent XDP (logically) in symbols as (where || is concatenation):

XDP-E - XDP encryption key

XDP-S - XDP signature key

Info - Cryptographic algorithm related information

Data - The data to be encrypted

XDPHeader = Info || SigXDP-S(Data) || EncDPAPI(XDP-E, XDP-S)

XDPData = XDPHeader || Sig­XDP-S(XDPHeader) || EncXDP-E(Data)

The XDPData structure is designed to hold all the information that an XDP component needs in order to verify the XDPHeader, decrypt the encrypted data and verify that the data has not been altered.

The goals of the XDPData structure are as follows:

* The encrypted data should leak no information about the unencrypted data apart from its length.
* The signature of the unencrypted data in XDPInternalCommonHeader can be used to validate that the encrypted data has not been altered.
* The signature of XDPInternalCommonHeader and XDPInternalMachineHeader, or XDPInternalCommonHeader and XDPInternalDomainHeader can be used to validate that the headers have not been altered. XDPInternalMachineHeader and XDPInternalDomainHeader contain the identities that are authorized to decrypt the encrypted data.
* XDPEncryptedKeys contains XDPEncryptionKey and XDPSignatureKey encrypted using the DPAPI which ensures only the XDP Service (Machine or Domain) that created the XDPEncryptedKeys can decrypt it.
* XDPEncryptionKey is used to encrypt the Data.
* XDPSignatureKey is used for all the signatures.

For a discussion of the security of the encrypted and signed data see section 6.

## Registry Settings

The registry settings will be stored under HKLM and ACLed such that the account running the service has read/write access.

The following are common settings:

|  |  |  |  |
| --- | --- | --- | --- |
| **Key Name** | **Key Type** | **Possible Values** | **Default Value** |
| Encryption Algorithm | String | Aes, DES, TripleDES | AesManaged |
| Signature Algorithm | String | HMACSHA256, HMACSHA384, HMACSHA512 | HMACSHA256 |
| Encryption Mode | String | CBC, ECB, OFB, CFB, CTS | CBC |
| Communications Security | String | Kerberos, KerberosOrNTLM | Kerberos |
| Domain Port | UInteger | 1-65535 | 3483 |
| Network Timeout | UInteger | 0-(10\*60) | 30 |
| Service Min Thread Pool Size | UInteger | 5-20 | 5 |

The following registry options are specific to XDP Machine Service:

|  |  |  |  |
| --- | --- | --- | --- |
| **Key Name** | **Key Type** | **Possible Values** | **Default Value** |
| XDP Domain Service Account | String | \* | XDPDSHostname |
| Domain Hostname | String | \* |  |

Where:

* XDP Domain Service Account – The name of the Domain account running the XDP Domain Service. The XDPMachine Service needs to know this to set up an authenticated Kerberos connection.
* Domain Hostname – The name of the host where the XDP Domain Service is installed.

The following registry options are specific to the XDP Domain Service

|  |  |  |  |
| --- | --- | --- | --- |
| **Key Name** | **Key Type** | **Possible Values** | **Default Value** |
| Data Recovery Group Name | String |  |  |
| Update Client Crypto | Boolean | True, False | True |

Where:

* Data Recovery Group Name – This is the name of a Domain group that will be checked for membership for domain decryption requests so authorized data recovery users can decrypt data.
* Update Client Crypto – Decides whether or not the DS will try to update the client cryptographic algorithms if the client passes through different algorithms than the DS has configured.

If the registry values are set to a value that is not in the range of possible values, then the default value is used and the value in the registry is updated.

## XDP Machine Service

The XDP Machine Service will be running under a unique account created at install time with a strong random password. The password chosen will be sufficiently complicated (GUID) that it will never need to be changed. When the service starts it will drop any permission it does not need.

If an exception occurs (either expected or unexpected) it will be written to a log file where the service is installed.

### Setup

The XDP Machine Service will create 2 named pipes:

* \\.\pipe\XDPProtectData

Takes a serialised XDP.MachineService.Messages.XDPRequestProtectData object (defined in XDPMachineServiceMessages.xsd) to encrypt data. Will respond with a serialised XDP.MachineService.Messages.XDPResponseProtectData object.

* \\.\pipe\XDPUnprotectData

Takes a serialised XDP.MachineService.Messages.XDPRequestUnprotectData object (defined in XDPMachineServiceMessages.xsd) to encrypt data. Will respond with a serialised XDP.MachineService.Messages.XDPResponseUnprotectData object.

Whilst any client could invoke the services of these pipes directly, XDP.dll is a .Net client that is installed in the GAC when XDP Machine Service is installed (the API is detailed in 5.1).

### Encryption Process

This section discusses how the encryption process creates XDPData.

1. UserA serialises a XDPRequestProtectData object and writes it to the XDPProtectData pipe.
2. The XDP Machine Service (MS) creates a new thread to handle the request and impersonates UserA.
3. The MS deserialises the request and calls XDPData.Encrypt.
4. The AuthorizedIdentities are split into local and domain users. The local users are transformed into SIDs. All local users must exist otherwise an exception is thrown.
5. Encryption and signature keys are generated, these are cryptographically strong random bytes (the number of bytes depends on the algorithm being used).
6. Separate threads are created to sign and encrypt the data.
7. The MS creates an XDPInternalCommonHeader. This contains all the crypto settings used and the signature of the encrypted data.
8. If AuthorizedIdentities contained any domain users, then a request is made to the XDP Domain Service for an XDPInternalDomainHeader (see 5.5.1).
9. The MS reverts the impersonation context.
10. If AuthorizedIdentities contains any local users MS creates an XDPInternalMachineHeader. This contains the local machine AuthorizedIdentities and DPAPI encrypted encryption and signature keys.
11. The MS creates XDPInternalHeaderSignatures of the XDPInternalCommonHeader, XDPInternalDomainHeader and XDPInternalMachineHeader.
12. The XDPData object is now fully populated and serialised to a byte array.
13. An XDPResponseProtectData object is created with the byte array and returned to the caller via the XDPProtectData pipe.

### Decryption Process

This section discusses how the decryption process uses XDPData to recover the data.

1. UserA serialises a XDPRequestUnprotectData object and writes it to the XDPUnprotectData pipe.
2. The XDP Machine Service (MS) creates a new thread to handle the request and impersonates UserA.
3. The MS deserialises the request and calls XDPData.Decrypt.
4. If UserA is a domain user, the MS makes a request to the XDP Domain Service to get the XDP Keys (see 5.5.2).
5. If UserA is a local user, the MS gets the XDPInternalMachineHeader
6. The MS verifies that UserA is in the list of AuthorizedIdentities (or a member if there are groups).
7. The MS decrypts the XDPEncryptedKeys to get the encryption and signature keys.
8. The MS uses the signature key to verify the signature on the XDPInternalCommonHeader and XDPInternalMachineHeader.
9. The MS decrypts the encrypted data.
10. The MS verifies the signature on the unencrypted data.
11. The MS creates an XDPResponseUnprotect object and populates it with the decrypted data and serialises the object to the XDPUnprotectData pipe.

### Error Handling

Exceptions will be thrown when bad parameters are found, when signatures do not verify and when the caller is not authorized to decrypt.

## XDP Domain Service

The XDP Domain Service will be running under a unique account created at install time with a strong random password (a GUID). The password chosen will be sufficiently complicated that it will never need to be changed. When the service starts it will drop any permission it does not need.

If an exception occurs (either expected or unexpected) it will be written to a log file where the service is installed.

### Encryption Process

1. Receive XDPRequestDomainHeader (which includes XDPInternalCommonHeader, XDPAuthorizedIdentities and XDPKeys).
2. Read the XDP settings from the registry.
3. Check if XDPInternalCommonHeader references the same cryptographic information as the local XDP settings. If it doesn’t
4. Update XDPInternalCommonHeader with the XDP settings.
5. Return an XDPException/XDPUpdateCommonHeader (which includes the updated XDPInternalCommonHeader).
6. Stop processing this request.
7. Verify that each identity in XDPAuthorizedIdentities belongs to the right domain. If any don’t:
8. Return XDPException/XDPUnknownIdentity.
9. Stop processing this request.
10. Create XDPEncryptedKeys by DPAPI encrypting XDPKeys.
11. Create XDPInternalDomainHeader (from XDPAuthorizedIdentities and XDPEncryptedKeys).
12. Create XDPInternalDomainHeaderSignature of XDPInternalCommonHeader combined with XDPInternalDomainHeader, using XDPSignatureKey.
13. Securely delete XDPKeys.
14. Return XDPResponseDomainHeader (which includes XDPInternalDomainHeader and XDPInternalDomainHeaderSignature)

### Decryption Process

1. Receive XDPRequestDecryptionKey (which includes XDPInternalCommonHeader, XDPInternalDomainHeader and XDPInternalDomainHeaderSignature)
2. Get the identity of the caller.
3. Confirm the identity of the caller is in (or a member of) the identities listed in XDPAuthorizedIdentities in XDPInternalDomainHeader. This is solely done for the sake of efficiency. If it isn’t:
4. Return XDPException/XDPNotAuthorized.
5. Stop processing this request.
6. DPAPI decrypt XDPInternalDomainHeader/XDPEncryptedKeys to recover XDPKeys (which includes XDPEncryptionKey and XDPSignatureKey).
7. Validate XDPInternalDomainHeaderSignature by re-creating the signature of XDPInternalCommonHeader combined with XDPInternalDomainHeader using XDPSignatureKey. If it does not validate:
8. Return XDPException/XDPBadSignature.
9. Stop processing this request.
10. If the signature is valid, then this implies the authorisation check in step 3 is accurate.
11. Create and return XDPResponseDecryptionKey which includes XDPKeys.

### Error Handling

The possible error messages returned by the XDP Domain Service are encapsulated by the XDPException messages in XDPMessages.xsd. The encryption and decryption processes list the specific exception messages that occur is different situations, however generic messages are possible as well.

XDPException/XDPBadParameter is returned whenever an input parameter is considered bad, and includes the name of the parameter and the reason it is bad.

XDPException/XDPGeneralException is for any error that is unexpected or does not have a specific exception message handling it.

# Security

## Security of the encrypted data

The security of the encrypted data directly relates to the security of the encryption algorithm. Encryption algorithms are designed to be secure against an attacker that is allowed to request encryptions and decryptions by the target key. For XDP, an attacker is unable to request encryptions as the key is always randomly generated, and they cannot request decryptions without first creating a forged signature (see next section).

An attacker can see a symmetric signature of the unencrypted data, so this might leak some information about the data. This leads us to the following assumption:

Assumption: The symmetric signature does not leak any information about the unencrypted data.

This is a reasonably assumption given that most symmetric signature algorithms are either based on encryption algorithms or on keyed-hash functions which are designed not to leak information, especially without knowledge of the signature key.

## Security of signed information

Typically when considering an attacker against a signature scheme the attacker is allowed to choose arbitrary messages, get them signed and is tasked with producing a signature for a message they did not ask the signature for. The attacker is trying to forge a signature.

This scenario is somewhat different; every message an attacker submits to be signed, is signed using a different key, and it may be possible for an attacker to influence the signature key that is used to verify a signature (as there is a circular reference, the signature is on data that contains the (encrypted) signature key).

All the sensitive information in XDPData is signed. If we assume an attacker is unable to influence the signature key, then any signature scheme that is secure against forgery attacks would suffice to ensure the integrity of the information in XDPData.

An attacker is however able to influence the signature key, using one of three methods:

1. Alter the bits of XDPEncryptedKeys of the target XDPData they are trying to alter.
2. Replace XDPEncryptedKeys (with a valid DPAPI encrypted XDPEncryptedKeys obtained from somewhere), they may or may not alter the bits of this.
3. Replace XDPEncryptedKeys (with a valid DPAPI encrypted XDPEncryptedKeys) where they know the values of the encrypted keys.

Method 3 is pointless as all this would give the attacker would be the encrypted data decrypted under a key of their choosing, but they clearly don’t need to ask an XDP Service to do this, they can just decrypt the encrypted data using that key themselves.

Method 2 is pointless since replacing XDPEncryptedKeys also replaces the encryption key, and so even if a successful signature forgery was achieved, the attacker would only get the encrypted data decrypted under a key they do not know. If they do not know the key, then it was randomly generated (by an XDP Service), so it’s reasonable to assume they gain no more information than if they were to decrypt using a random key they generated.

For an attacker to succeed using Method 1, they need to alter XDPEncryptedKeys in such a way that they can alter the signed data and/or the signature value so that an XDP Service will verify the signature on the headers. Actually the attacker needs to do more since the same signature key is used to sign the data, so if they change the signature key they also need to update the signature on the data so it is valid, however the attacker does not know the value of the unencrypted data to create a forged signature.

It would be difficult to prove that such an attack is impossible, but we will assume this type of attack is not feasible.

Even if it were feasible, it could be reasonably argued that changing the signature key by changing XDPEncryptedKeys will also change the encryption key. So the outcome of a successful attack would be to get the encrypted data decrypted under a key related to the actual encryption key. This is called a ‘related-key attack’ and so it would suffice that the encryption algorithm be secure against this type of attack (this may not even apply as the way the keys are related would not be known).