DPAPI Secrets

security analysis and data recovery in DPAPI

Passcape Software

1 Abstract

In this paper, we are making an attempt to analyze the operation of DPAPI, review the undocumented structures and encryption algorithms of DPAPI, understand and describe the internal functioning of the system.

The paper presents the world's first complete (not claiming to be universal though) description of DPAPI operation logic and all the undocumented structures, including DPAPI blobs, Master Keys and credential history files.

We also attempt to point out some functional drawbacks of the latest version of DPAPI and possible ways to eliminating those. Additionally, a deeper analysis of the implementation of the first version of DPAPI, released along with Windows 2000, has revealed the presence of a number of serious vulnerabilities and put the entire security of the system into question.

Much attention is paid to the recovery of DPAPI data when user profile cannot be loaded. For the first time ever, we practically tested the user logon password recovery algorithm without using SAM or NTDS.DIT files.

The operation of DPAPI was analyzed using a set of 6 utilities, integrated into one of the most powerful applications for auditing Windows passwords, Windows Password Recovery. We hope these tools, as well as certain information from our article, is found interesting not only to experts and criminologists but to all researchers in the field of computer security.

2 DPAPI architecture and security

2.1 What is DPAPI

Beginning with Windows 2000, Microsoft ships their operating systems with a special data protection interface, known as Data Protection Application Programming Interface (**DPAPI**). DPAPI is currently widespread and used in many Windows applications and subsystems. For example, in the file encryption system, for storing wireless connection passwords, in Windows Credential Manager, Internet Explorer, Outlook, Skype, Windows CardSpace, Windows Vault, Google Chrome, etc. DPAPI has become popular among programmers first of all due to its simplicity of use, as it consists of just a couple of functions for encrypting and decrypting data, **CryptProtectData** and **CryptUnprotectData**.

Despite its apparent simplicity, the technical implementation of DPAPI is rather complicated, and these functions' operation logic is much like the cheerful childish rhyme "The House that Jack Built". Perhaps, that's the reason why the internal structures and operating principles of DPAPI had been kept behind a closed curtain for so long.

For the first time, DPAPI was analyzed by Passcape Software in 2003. In 2005, the company released the first commercial application (Outlook Password Recovery) based on that recovery, which could decrypt DPAPI blobs offline, i.e. without logging on to owner's account. The decryption algorithm proposed by Passcape merely simulated DPAPI operation, so the DPAPI system has not been compromised.

With the release of the brand-new version of Windows Password Recovery tool, the offline DPAPI decryption algorithm has gotten a new round. Now, besides decrypting DPAPI blobs, the program could

also analyze them, find them on disk, decrypt Master Keys and validate their passwords, extract users' credential history hashes from DPAPI, and much more.

2.2 What does DPAPI protect?

DPAPI is utilized to protect the following personal data:

- Passwords and form auto-completion data in Internet Explorer, Google *Chrome
- E-mail account passwords in Outlook, Windows Mail, Windows Mail, etc.
- Internal FTP manager account passwords
- Shared folders and resources access passwords
- Wireless network account keys and passwords
- Encryption key in Windows CardSpace and Windows Vault
- Remote desktop connection passwords, .NET Passport
- Private keys for Encrypting File System (EFS), encrypting mail S-MIME, other user's certificates, SSL/TLS in Internet Information Services
- EAP/TLS and 802.1x (VPN and WiFi authentication)
- Network passwords in Credential Manager
- Personal data in any application programmatically protected with the API function CryptProtectData. For example, in Skype, Windows Rights Management Services, Windows Media, MSN messenger, Google Talk etc.

An example of a successful and clever way to protect data using DPAPI is the implementation of the auto-completion password encryption algorithm in Internet Explorer. To encrypt the login and password for a certain web page, it calls the CryptProtectData function, where in the optional entropy parameter it specifies the address of the web page. Thus, unless one knows the original URL where the password was entered, nobody, not even Internet Explorer itself, can decrypt that data back.

2.3 DPAPI architecture

Beginning with Windows 2000, as it was mentioned earlier, any application can protect its personal data (e.g., passwords) by simply calling the CryptProtectData function, which returns a "non-transparent" binary structure, also known as DPAPI blob. "Non-transparent", by Microsoft's definition, means that besides the original encrypted data of the application it also contains other supplemental stuff, necessary for decrypting that data. Despite that the structure of DPAPI blobs is not documented, with the release of the new version of Windows Password Recovery, for the first time ever it has become possible to decrypt DPAPI blobs offline and thoroughly analyze them.

Other API function, CryptUnprotectData, which, as its name suggests, works similarly, but the other way around, gets an encrypted DPAPI blob at the input and returns decoded data to the application.

Figure No 1. shows the DPAPI flow chart.

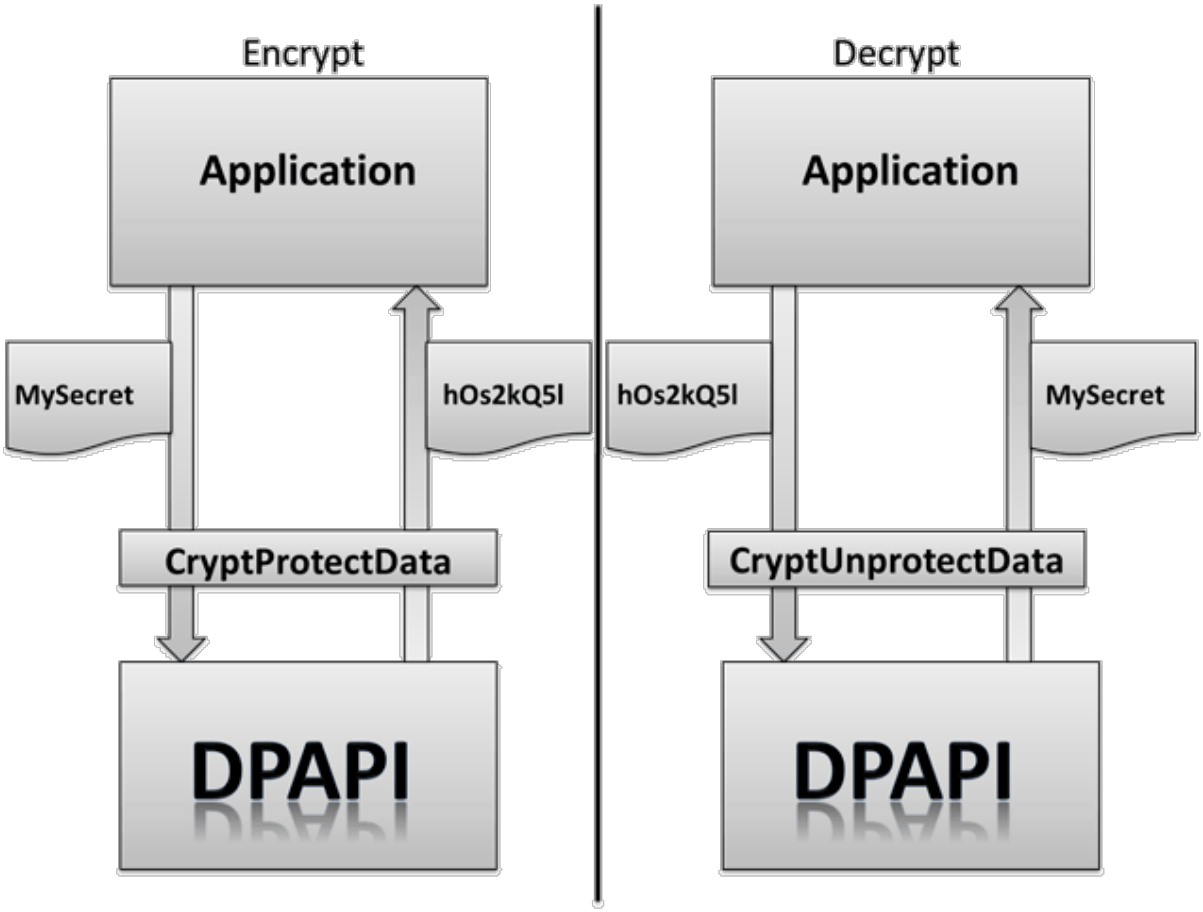

Figure 1. DPAPI flow chart.

DPAPI encryption is based upon user password; therefore, data encrypted under one account cannot be decrypted under other account. Moreover, DPAPI allows restricting access to data even within one account by setting an additional secret (entropy). Thus, unless it knows the additional secret, one application cannot access data protected in other application.

Every programmer implementing DPAPI interface must realize that the system only encrypts data. An application must provide a solution for storing the returned DPAPI blobs, including reliable hiding of optional entropy data if it is used.

2.4 DPAPI security

DPAPI was created with the view of many aspects in terms of security. Here is a list of key features that distinguish it from similar systems:

- PAPI works on top of CryptoAPI. You can create your own encryption service provider and use it with DPAPI.

- DPAPI uses proven cryptographic algorithms. For example, Windows 7 by default uses the AES256 encryption in the CBC mode, SHA512 for hashing and PBKDF2 as password-based key derivation routine.
- All the algorithms, as well as their key length, can be configured from the registry, which can be accessed only by users with the Administrator privileges. For example, the number of iterations in the PBKDF2 function is set in the following registry key: **HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Protect\Providers\%GUID% \MasterKeyIterationCount**
- All the parameters that can be configured from the registry have their restrictions and cannot be lesser than the default values. For example, in Windows XP the MasterKeyIterationCount default value is 4000.
- All critical data is protected by integrity check.
- Size of all encryption keys in all algorithms is sufficient to prevent possible brute force attack.
- For the same purpose, PBKDF2 has a rather great iteration counter. The only exception is the first implementation of DPAPI. The table below shows that the Master Key password search speed is sufficient for compromising user password in Windows 2000.
- For security reasons, the validity of the Master Key is limited. However, the implementation of that mechanism has weak points, which will be discussed below.
- All communications with the domain controller are carried out via a secure RPC channel.
- Cached data is written neither to disk, nor to swappable RAM. Intermediate keys and values are reset in the stack after running each encryption function. By the way, when analyzing the SYSKEY utility, Passcape specialists have found an interesting bug related to incorrect stack reset, which resulted in having a part of the system encryption key (SYSKEY) in some systems running under Windows 2000, Windows XP and Windows 2003 stored as plain text, available to all users of the system. Alas, it happens sometimes.

However, there are some nuances, the ignorance of which can weaken the DPAPI security system:

- **CRYPTPROTECT_LOCAL_MACHINE** - flag set in the CryptProtectData function, quite convenient in certain cases, in fact can compromise user data, not providing adequate protection. When it is set, the encryption does not depend on user's logon password. On the other hand, such behavior does not contradict the DPAPI concept and is related to peculiarities of interface implementation.
- DPAPI can be configured to run in the compatibility mode for Windows 2000, when the backup Master Key is stored locally and can be decrypted using the LSA secret. Thus, in theory, all the DPAPI blobs can be decrypted without knowing user password. That is what Windows Password Recovery demonstrates in practice. See screenshot No 2. Nevertheless, to activate that mode, one needs Administrator access to the registry key with DPAPI configuration.

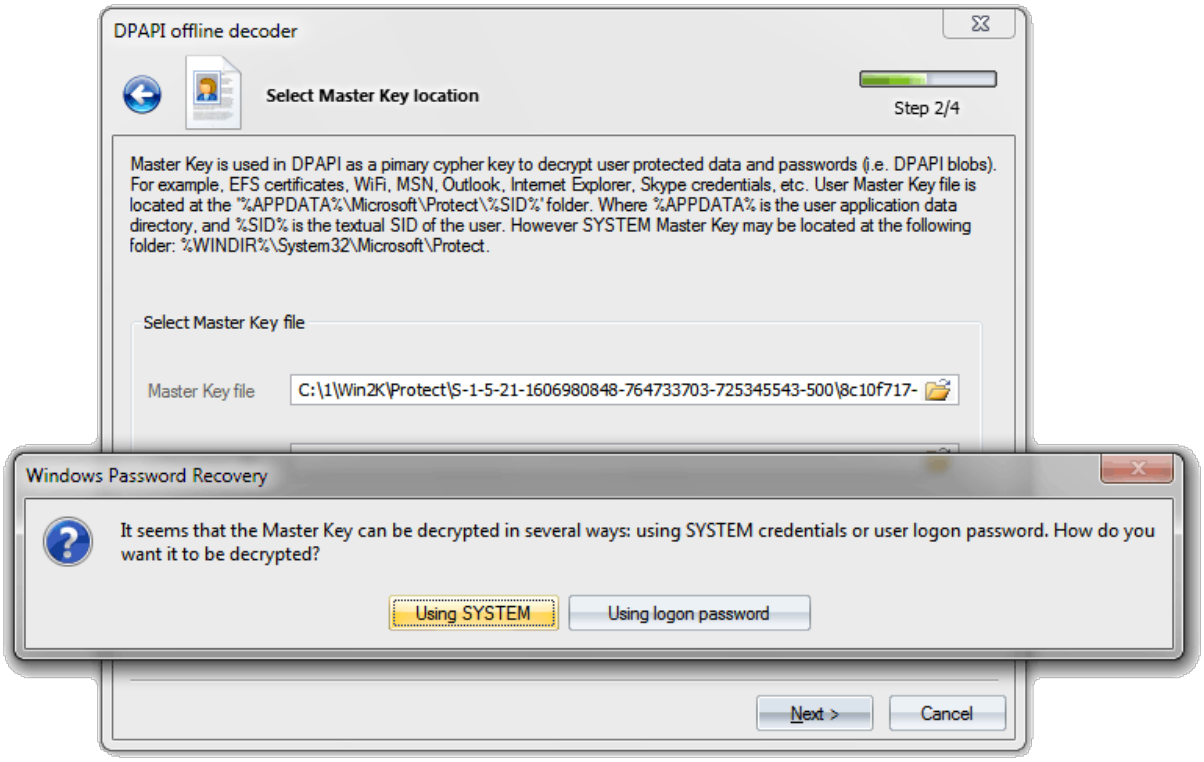

Figure 2. Decryptinng DPAPI blob without knowing the original user logon password in Windows 2000.

OS	Encryption algorithm	Hash algorithm	Number of iterations in PKCS#5 PBKDF2	Password guess speed (pwd/sec)
Windows 2000	RC4	SHA1	1	**95000**
Windows XP	3DES	SHA1	4000	**76**
Windows Vista	3DES	SHA1	24000	**12**
Windows 7	AES256	SHA512	5600	**10**
Windows 10	AES256	SHA512	8000	**<10**

Table 1. Default algorithms used in DPAPI.

If taken as a whole, DPAPI provides perhaps the strongest to date protection and safety of user data.

2.5 DPAPI configuration

DPAPI runs on top of Crypto API and can use any encryption service provider. The crypto provider is selected as defined in the following registry key:

HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Protect\Providers
Preferred

By default, the system uses Microsoft's crypto provider named **df9d8cd0-1501-11d1-8c7a-00c04fc297eb**, implemented in the psbase.dll library.
The encryption is configured by modifying the respective value in the following registry key:
HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Protect\Providers\%GUID%

Where %GUID% is the service provider's unique identifier. By default, the DPAPI settings key corresponds with
HKEY_LOCAL_MACHINE \Software\Microsoft\Cryptography\Protect\Providers\df9d8cd0-1501-11d1-8c7a-00c04fc297eb

This hive of the registry can be modified by users with Administrator rights only. Here are the descriptions of some keys of this hive.

MasterKeyIterationCount
Iteration counter for function PBKDF2, cannot be smaller than the default value. The key generation algorithm is used in the encryption of user's Master Key and credential history.

MasterKeyLegacyCompliance
Set the registry key to 1 to make DPAPI behavior similar to that of Windows 2000 when backing up/restoring Master Keys.

MasterKeyLegacyNt4Domain
Since DPAPI backup/restore support doesn't exist on NT4, DPAPI provides a workaround to avoid losing its data and force the client machine to use W2K style master key backup/restore. For that, you should set the MasterKeyLegacyNt4Domain parameter to 1.

DistributeBackupKey
Determines compatibility settings when backing up or restoring domain key. For example, if the domain functional level is set to Windows 2000 native.

Recovery Version
Determines version compatibility when backing up and restoring local copies of Master Keys. The default version is 2; Windows 7 and Windows Server 2008 R2 can be configured to use version 3 by setting the DWORD registry value to 3.

ProtectionPolicy
Default data protection policy flag. Corresponds with the **dwPolicy** flag in Master Key header.

Encr Alg - Encr Alg Key Size
Pair of values that control the type of the encryption algorithm used in DPAPI and the size of the key used in that algorithm.

MAC Alg - MAC Alg Key Size
Keys for setting the DPAPI data integrity control algorithm.

3 Data encryption in DPAPI

3.1 Description of CryptProtectData and CryptUnprotectData API functions

These functions' parameters are pretty much identical, so we are going to take a look only at CryptProtectData, which in C++ is declared as follows:

```
BOOL WINAPI CryptProtectData(
  __in      DATA_BLOB *pDataIn,
  __in      LPCWSTR szDataDescr,
  __in      DATA_BLOB *pOptionalEntropy,
  __in      PVOID pvReserved,
  __in_opt  CRYPTPROTECT_PROMPTSTRUCT *pPromptStruct,
  __in      DWORD dwFlags,
  __out     DATA_BLOB *pDataOut
);
```

Description of function fields:

DATA_BLOB *pDataIn	- pointer to a DATA_BLOB structure that refers to the original data to be encrypted.
LPCWSTR szDataDescr	- data descriptor; stored as plain text in DPAPI blob. This optional parameter is for information purposes only; it may be omitted in the function call.
DATA_BLOB *pOptionalEntropy	- as the foregoing, is an optional parameter. However, unlike the descriptor, the entropy directly affects data security, so it is not stored in the DPAPI blob. User must ensure the proper management of this secret. Some third-party applications (such as GTalk) store this secret in the registry, other Windows components (such as .Net Passport or WININET) use a hard-coded constant for the secret, third use a variable value. For example, the internal FTP client of Windows Vista creates entropy based on the server name. One way or the other, this may at least amuse the potential attacker. In our opinion, the most successful example of utilizing entropy is the one implemented in the password storage mechanism of Internet Explorer, which doesn't store the secret at all.
PVOID pvReserved	- currently not used.
CRYPTPROTECT_PROMPTSTRUCT *pPromptStruct	- when this structure is defined, user is required to enter an additional password for encrypting the DPAPI blob. Put it this way: It is used very infrequently due to the fact that the password prompt occurs in interactive mode.
DWORD dwFlags	- flags that control the encryption process. Here is the description of the most interesting of them:

CRYPTPROTECT_UI_FORBIDDEN	0x1	Used when user interface is not available. For example, when using remote access.
CRYPTPROTECT_LOCAL_MACHINE	0x4	Data is protected using local computer account. Any user of the system may be able to decrypt it.
CRYPTPROTECT_CRED_SYNC	0x8	Forces synchronizing user's credentals. Normally runs automatically upon user password change.
CRYPTPROTECT_AUDIT	0x10	Enables audit during encryption/dectyption
CRYPTPROTECT_VERIFY_PROTECTION	0x40	The flag checks security level of DPAPI blob. If the default security level is higher than current security level of the blob, the

			function returns error CRYPT_I_NEW_PROTECTION_REQUIRED as advice to reset securiry for the source data.
	CRYPTPROTECT_CRED_REGENERATE	0x80	Regenerate local computer passwords.
	CRYPTPROTECT_SYSTEM	x20000000	Indicates that only system processes can encrypt/decrypt data.
DATA_BLOB *pDataOut	- target DPAPI blob that occurs in the result of calling the function		

3.2 Examples of use for CryptProtectData/ CryptUnprotectData functions

For the convenience of running tests, as well as for the purposes of providing an example, we have created two simple utilities of the respective names, which implement mere wraps around DPAPI functions. Thus, DPAPI functions can be called from the command line. Here is the C++ source for CryptProtectData. The compiled EXE files of these programs are also available for download at: CryptProtectData, CryptUnprotectData.

Source codes : CryptProtectData , CryptUnprotectData.
Executable files : CryptProtectData , CryptUnprotectData.

CryptProtectData source code

```
// CryptProtectData.cpp : Defines the entry point for the console application.
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>

#pragma comment (lib, "Crypt32")

int _tmain(int argc, _TCHAR* argv[])
{
	if ( argc<3 || argc>5 )
	{
		_tprintf(TEXT("Syntax: %s secret output_filename [entropy_string] [flags]\n"),argv[0]);
		return 1;
	}

	//Declare variables
	DATA_BLOB DataIn;
	DATA_BLOB DataOut;
	DATA_BLOB DataEntropy;
	DWORD dwFlags;
	LPTSTR pFoo;

	//Initialize the structure
	DataOut.pbData=NULL;
	DataOut.cbData=0;
```

```
//
DataIn.pbData=(LPBYTE)(argv[1]);
DataIn.cbData=(lstrlen(argv[1])+1) * sizeof(TCHAR) ;
//
if ( argc>=4 )
{
	DataEntropy.pbData=(LPBYTE)(argv[3]);
	DataEntropy.cbData=(lstrlen(argv[3])+1) * sizeof(TCHAR) ;
}
//
if ( argc==5 )
	dwFlags=_tcstoul(argv[4],&pFoo,10);
else
	dwFlags=0;

//Protect the secret
if ( !CryptProtectData(
	&DataIn,
	TEXT("CryptProtectData by Passcape Software"), //description string to be included
	argc>=4?&DataEntropy:NULL,                     //Optional entropy
	NULL,                                          //reserved
	NULL,                                          //prompt structure, not used
	dwFlags,                                       //flags
	&DataOut) )
{
	dwFlags=GetLastError();
	_tprintf(TEXT("CryptProtectData failed with the following error code: %lu\n"),dwFlags);
	exit(1);
}

//save the output blob
FILE *f=NULL;
_tfopen_s(&f,argv[2],TEXT("wb"));
if ( !f )
{
	if ( DataOut.pbData )
	{
		LocalFree(DataOut.pbData);
		DataOut.pbData=NULL;
	}
	_tprintf(TEXT("Can't open output file for writing\n"));
	exit(2);
}

//write
if ( DataOut.pbData )
{
	size_t written=fwrite(DataOut.pbData,DataOut.cbData,1,f);
	LocalFree(DataOut.pbData);
	DataOut.pbData=NULL;
	if ( written!=1 )
	{
		_tprintf(TEXT("Can't write %lu bytes to output file\n"),DataOut.cbData);
		exit(3);
```

```
                }
        }

        fclose(f);
        return 0;
}
```

3.3 Data encryption in DPAPI

The encryption of user's personal data in DPAPI goes in three phases:

1. First, the application calls the CryptProtectData function, specifying the source data to be encrypted and the optional entropy parameter. If the latter is not specified, any other program in the context of current user may be able to decrypt the data by gaining physical access to the DPAPI blob.
2. The CryptProtectData function belongs to the CryptoAPI family and is located in Crypt32.dll. From there, the request for further processing is forwarded via a secure RPC channel to the Local Security Authority (LSA) system process, and the context of current user is switched to the system. All the further manipulations are run in the context of the local system.
3. In the LSA, the data is actually encrypted and forwarded back to Crypt32.dll via the RPC channel; from there, the data safely travels to the original application.

CryptProtectData and CryptUnprotectData are basically just wrapper functions, since the actual data encryption runs in the context of the system. The magic part of DPAPI takes place in the LSA, and is hidden from the stranger's eye.

The encryption process is lengthy and boring; it may remind of "fulfilling the marital duty after 20 years of living together". In the beginning, we take user's password; then, by applying the PBKDB2 cryptographic standard to it, we get the Master Key encryption key. Now, we use that encryption key to decrypt the Master Key. If the decryption fails, the user password is decrypted using the first hash of the credential history, CREDHIST, which in its turn also gets the Master Key encryption key and attempts to decrypt the Master Key again. If that attempt too ends up in failure, we decrypt the next credential history hash, and so on. The operation continues until either the Master Key is decrypted or the credential history runs out of its items.

To avoid password prompts every time CryptProtectData / CryptUnprotectData is called, Windows caches the password and stores it in the depths of the LSA for the duration of the user profile life. That is why DPAPI works exclusively online, only after the user has logged on to the system.

In 2005, for the first time ever, Passcape Software introduced an algorithm that works in all of the company's products and simulates the offline operation of DPAPI. In other words, to decrypt DPAPI blobs, one no longer needs to load user's context; although having the password is mandatory. In no way does the offline decryption of DPAPI makes the system more vulnerable; at the same time, it remains helpful to researchers, experts and criminalists.

So, user's Master Key resides in a special container file, which can store additional copies - backups - of the Master Key. A Master Key backup is used when the primary copy, for one or the other reason, cannot be decrypted. For example, after a forced password reset. Master Key is a critical element in DPAPI cryptography; therefore, its length is 512 bits, and only proven algorithms are used when encrypting it. See the table 1.

The decryption of Master Key in the first implementation of DPAPI required the NTLM hash of user's password. That was a major: No, that was the greatest blunder of DPAPI developers, which negated the security of the system. All because it technically allowed a potential malefactor to decrypt the Master Key and, consequently, any DPAPI blob by getting the NTLM hash directly from the SAM file; the actual password was not even necessary! In the second, current version of DPAPI, that error was fixed; now Master Key is encrypted using the SHA1 hash.

Once the Master Key is successfully decoded, its source 512-bit data participates in obtaining the symmetric encryption key for the DPAPI blob. For that purpose, the Master Key is "mixed" with random data (which would be then stored in the DPAPI blob) and entropy, if it is specified. Thus we achieve a three-dimensional uniqueness, so to say, of the encryption key for each DPAPI blob.

And, finally, there goes the actual encryption of the source data. In Windows 7, data is encrypted using the AES256 algorithm. All the phases of that intricate process involve using "salt", i.e. set of random data, unifying encryption keys and thus preventing the potential brute force attack with rainbow tables.

By the way, few people know that the first versions of Windows 7 had serious problems with AES encryption. Simply put, this type of encryption did not function at all. Fortunately, the problem was quickly eliminated.

Once the data is encrypted, it is grouped into a single structure along with other system information and sent back to the application as a DPAPI blob. The structure of a DPAPI blob is not documented; however, Passcape researches were able to fully reverse it. Here it is.

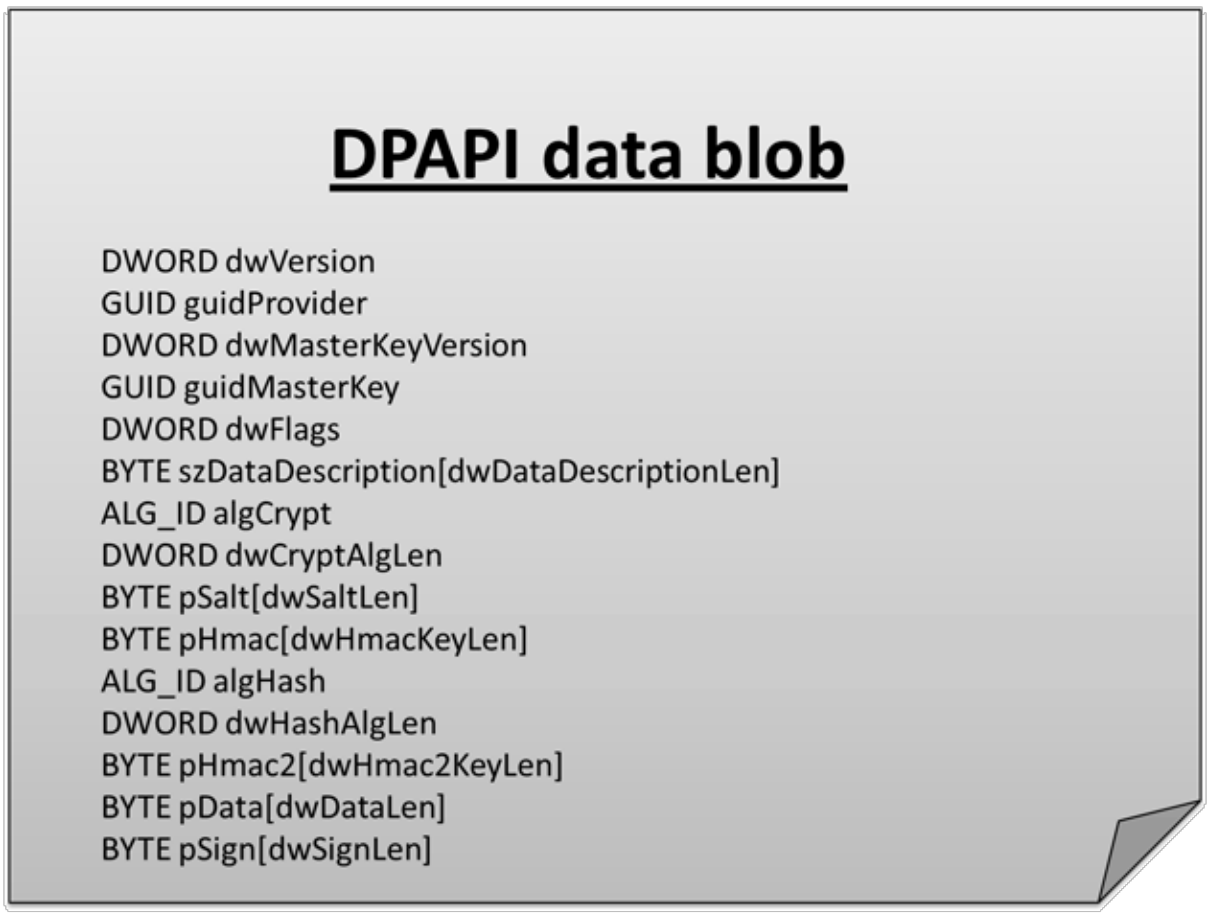

Figure 3. Undocumented structure of DPAPI blob.

4 DPAPI Master Keys

4.1 What is DPAPI Master Key

A user's Master Key is a binary file that contains encrypted data used for creating the primary encryption key in all DPAPI blobs. Since Master Key encrypts user's confidential data, the key itself requires serious protection. User's password was meaningfully chosen as source data for protecting the Master Key.

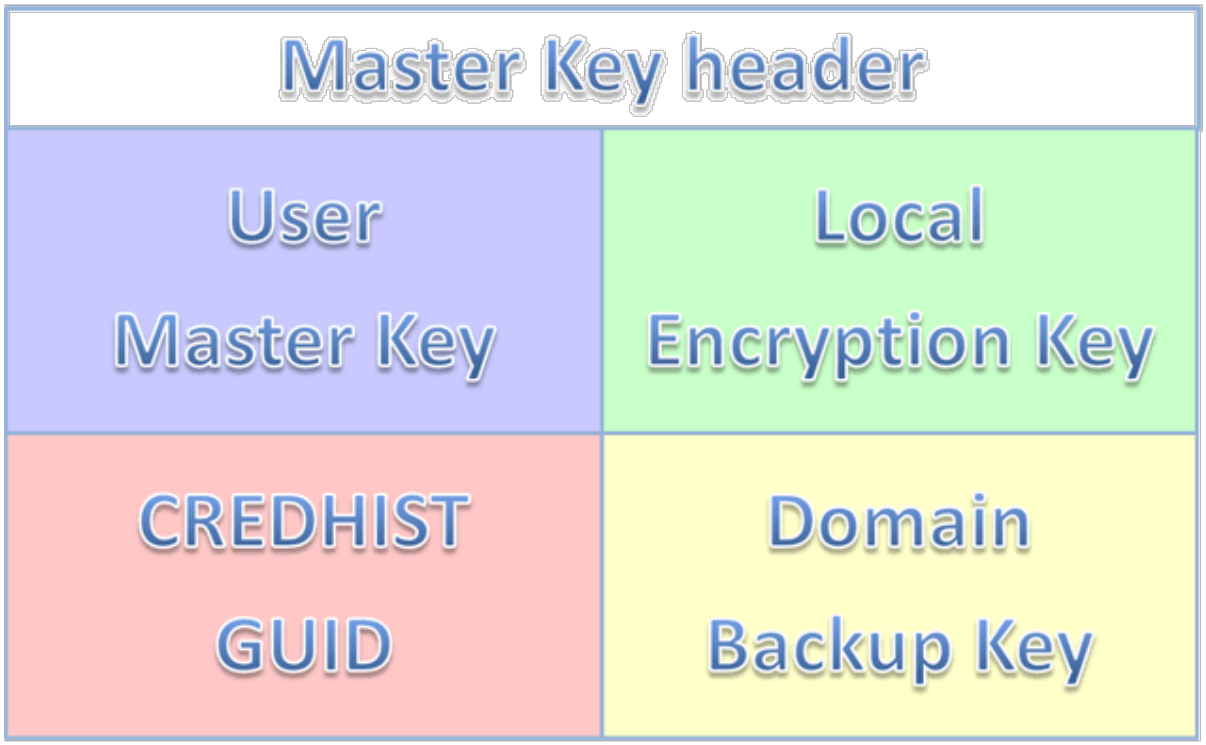

Figure 4. Master Key structure.

4.2 Master Key structure

As shown on Figure 4, a Master Key file consists of 5 units:

1. Header and system information
2. User's Master Key
3. Local backup encryption key
4. Unique CREDHIST file identifier
5. Domain Master Key backup

In Windows 2000, instead of the CREDHIST identifier, the system could store the local Master Key backup. That is what made the DPAPI system extremely vulnerable. Windows Password Recovery, for example, can decrypt a Master Key from its local backup and wouldn't even need the user's password to do that.

The header of the Master Key file contains the following information (C++ syntax followed by description of each field):

```
typedef struct _tagMasterKey
{
        DWORD dwVersion;
        DWORD dwReserved1;
        DWORD dwReserved2;
        WCHAR szGuid[0x24];
        DWORD dwUnused1;
        DWORD dwUnused2;
        DWORD dwPolicy;
        DWORD dwUserKeySize;
        DWORD dwLocalEncKeySize;
        DWORD dwLocalKeySize;
        DWORD dwDomainKeySize;
} MASTERKEY, *PMASTERKEY;
```

DWORD dwVersion	- version of the Master Key file used for the purposes of compatibility check. For Windows 2000 this field is set to 1; for other operating systems - to 2.
WCHAR szGuid[0x24]	- string value with a unique Master Key identifier, normally matching the file name. Any DPAPI blob stores this identifier, which links it to a specific (and only one) Master Key.
DWORD dwPolicy	- field with various flags. For example, if bit 3 of this field is set, the decryption key for the Master Key (from user's password) will be created using the SHA1 algorithm. In Windows 2000, this flag is always cleared; i.e. it uses NTLM hash.
DWORD dwUserKeySize	- contains the size of user's Master Key.
DWORD dwLocalEncKeySize	- size of backup copy encryption key.
DWORD dwLocalKeySize	- size of local backup key or CREDHIST GUID field
DWORD dwDomainKeySize	- size of domain's Master Key backup.

The header is followed by a sequence of 4 Master Key slots. Each slot contains its own header and the actual key.
The header structure is common for all slots. Here is what it looks like for version 1 (Windows 2000):

```
typedef struct _tagMasterKey1Base
{
        DWORD dwVersion;
        BYTE pSalt[0x10];
        BYTE pKey[];
} MASTERKEY1BASE, *PMASTERKEY1BASE;
```

In other versions of OS, the structure is complemented with new fields:

```
typedef struct _tagMasterKey2Base
{
```

```
        DWORD dwVersion;
        BYTE pSalt[0x10];
        DWORD dwPBKDF2IterationCount;
        ALG_ID HMACAlgId;
        ALG_ID CryptAlgId;
        BYTE pKey[];
} MASTERKEY2BASE, *PMASTERKEY2BASE;

typedef struct _tagMasterKey3Base
{
        DWORD dwVersion;
        GUID guidCredhist;
} MASTERKEY3BASE, *PMASTERKEY3BASE;
```

Description of header slot fields:

DWORD dwVersion	slot version.
BYTE pSalt[0x10]	salt, i.e. set of random data for unifying Master Key and avoiding possible rainbow table attack.
DWORD dwPBKDF2IterationCount	iteration counter in the PBKDF2 key generation algorithm.
ALG_ID HMACAlgId	data integrity check algorithm.
ALG_ID CryptAlgId	slot encryption algorithm identifier.
BYTE pKey[]	encrypted Master Key or slot data.
GUID guidCredhist	unique credentials history file identifier.

The table 2 shows the default Master Key encryption algorithms for various operating systems, as well as the analysis of their protection from a potential brute force attack.

OS	CryptAlgId	HMACAlgId	dwPBKDF2IterationCount	Password guess speed (pwd/sec)
Windows 2000	RC4	SHA1	1	**95000**
Windows XP	3DES	SHA1	4000	**76**
Windows Vista	3DES	SHA1	24000	**12**
Windows 7	AES256	SHA512	5600	**10**
Windows 10	AES256	SHA512	8000	**<10**

Table 2. Master Key encryption algorithms.

4.3 Generating new Master Key

Let's take a closer look at the process of creating and encrypting user's new Master Key:

- First, we call the API function **RtlGenRandom**, which returns 64 pseudo-randomly generated bytes. That's the raw material for the Master Key to be used for encrypting DPAPI blobs.
- Now we need to protect the raw material for the Master Key. For that purpose, we encrypt it using the user's logon password, security identifier (SID) and additional 16 bytes of random data (so-called "salt"). The Master Key encryption process consists of two major steps. First, we use the Password-Based Key Derivation function (crypto standard described in PKCS #5) and get the key out of the password, SID and salt. Then we use that key to obtain a symmetric Master Key encryption key.

Different versions of Windows protect Master Key different ways. Thus, in Windows 2000 the default encryption algorithm is **RC4**, in Windows XP and Win2K3 it's **3DES**, and Windows Vista and higher use **AES256**.

- Master Key obtains a unique name, **GUID**. Each DPAPI blob stores that unique identifier, which it is bound to via a warm friendly relationship. In other words, Master Key GUID is the key's "link" to the DPAPI blob.
- Master Key, created and encrypted with user's password, is stored in a separate file in the Master Key storage folder along with other system data. MKSF is a special location on disk where Master Keys are stored. User's Master Keys are stored in **%APPDATA%/Microsoft/Protect/%SID%**, where **%APPDATA%** is the Application Data directory. For example, C:/Users/John/AppData/Roaming/Microsoft/Protect/S-1-5-21-2893984454-3019278361-1452863341-1003, and %SID% is user's security identifier. In the above example that's S-1-5-21-2893984454-3019278361-1452863341-1003. System's Master Keys are stored in **%WINDIR%/System32/Microsoft/Protect** and used for decrypting DPAPI blobs, protected under a local system account. All the DPAPI blobs created with the CRYPTPROTECT_LOCAL_MACHINE flag set in the CryptProtectData function are protected with the system's Master Keys.
- Master Key creation, encryption or decryption can be extremely time-consuming, by the standards of the operating system. For example, in Windows 7, it takes over 0.1 of a second to decrypt a Master Key on a modern PC. But DPAPI has internal caching, so once decrypted, a Master Key is placed in the cache, and accessing it again does not involve a new decryption route.

4.4 Regenerating Master Keys

If you open Master Key Storage Folder on your disk, having previously allowed the system to show hidden folders and files (Master Key files have the Hidden attribute), you will likely see several Master Key files. Why several? - Because DPAPI, for safety reasons, forces the creation of a new Master Key approximately every 90 days, so any user account normally has several Master Keys; the number depends on the user profile creation date.

The Master Key regeneration period is hard-coded in the system. It is assumed that it cannot be modified. However, there is a "BUT" to it: The Master Key folder contains an 18-byte file named Preferred, which contains the name of the last Master Key created by the system, as well as the creation date.

Preferred file structure

```
typedef struct _tagPreferredMasterKey
{
        GUID guidMasterKey;
        FILETIME ftCreated;
} PREFERREDMASTERKEY, *PPREFERREDMASTERKEY;
```

Every time some DPAPI function is called, the system checks the last Master Key creation date in Preferred and, if finds it older than 90 days, creates a new Master Key, updating Preferred along the way. The Master Key regeneration process can be managed manually. To do that, call the CryptProtectData function with the CRYPTPROTECT_CRED_SYNC flag set. The system will re-encrypt all the Master Keys and update them on the disk. That operation is normally performed after changing user's password. To prevent the optional Master Key updates, Windows 7 introduced a Master Key synchronization mechanism, which we are going to cover below.

The data in the Preferred file is not protected by any means, so by modifying the last Master Key creation date by hand, one can manually manage the regeneration process; for example, infinitely extend the life of the last Master Key or create any number of Master Keys. But while the extension of life for a Master Key does not give malefactor any privileges, in the last case, a great number of Master Keys followed by password change can put the system in a lengthy screeching halt, as all Master Keys undergo a mandatory re-encryption every time user's password is changed.

Now, figure out how much time it would take to process, for example, 1000 Master Keys in Windows 7 if processing a single Master Key takes over 0.3 of a second. Thus, in our opinion, the Master Key regeneration process requires a bit of additional work. Ideally, adding data integrity control-the absence of which appears to be just about the only major vulnerability of DPAPI, and even that can hardly be called vulnerability-would be sufficient. Overall, the entire DPAPI system is well balanced and robust.

4.5 Backing up Master Keys and restoring them from backups

Backing up Master Keys of a domain PC

When the computer is a member of a domain, the Master Key file stores the backup of user's Master Key. When creating a Master Key, DPAPI sends a request to the domain controller that stores the RSA encryption private/public key pair. The Master Key is encrypted using the domain's public key and stored as backup in the same file with the user's Master Key. If an error occurs while decrypting the primary copy of the Master Key, DPAPI sends the backup to the domain controller. The domain controller decrypts it using its private RSA key and sends the decrypted Master Key back.

Backing up Master Key of a stand-alone PC under Windows XP - Windows 7

When it comes to stand-alone computers of home users (Windows XP and higher), DPAPI employs a different recovery mechanism based on password reset disk. Password reset disk, which user can create at any time in the control panel, allows user to recover a forgotten password. The key phrase in the previous sentence is user can. The burden of data recovery rests entirely on user's shoulders.

But what happens if user changes the password after the password reset disk has been created? Everything is going to be just fine. The process is simple and, at the same time, robust as a village closet. When creating a password reset disk, the system creates a pair of the public and 2048-bit private RSA encryption keys. User's current password is encrypted using the public key and stored in user's registry key **HKEY_LOCAL_MACHINE/SECURITY/Recovery/%SID%** (for Windows XP - Windows Vista) or in **HKEY_LOCAL_MACHINE/C80ED86A-0D28-40dc-B379-BB594E14EA1B** (for Windows 7). Anyway, access to these keys is forbidden to all users of the system, including Administrators. And the private key that is required for the decryption of the password is saved on the disk. Thus, if after creating a password reset disk user changes the password, the new password is encrypted with the public RSA key and is saved in the registry, replacing the old one. Since the public RSA key matches the respective private key stored on the disk, the private key can be used any time later for decrypting the new password that has just been created and is stored in the registry. By the way, one of the latest versions of Windows Password Recovery can decrypt user's plaintext logon password when the password reset disk is available.

Backing up Master Keys on a stand-alone PC in Windows 2000

But the password reset system came only in Windows XP. What was before that? Indeed, despite the fact that Windows 2000 did not have password reset disk, user's Master Key backup was still performed on stand-alone computers. Both the primary Master Key and the Master Key backup were stored in the same! That means that any data encrypted with DPAPI could be decrypted by a potential malefactor WITHOUT knowing user's logon password (by just having physical access to the system).

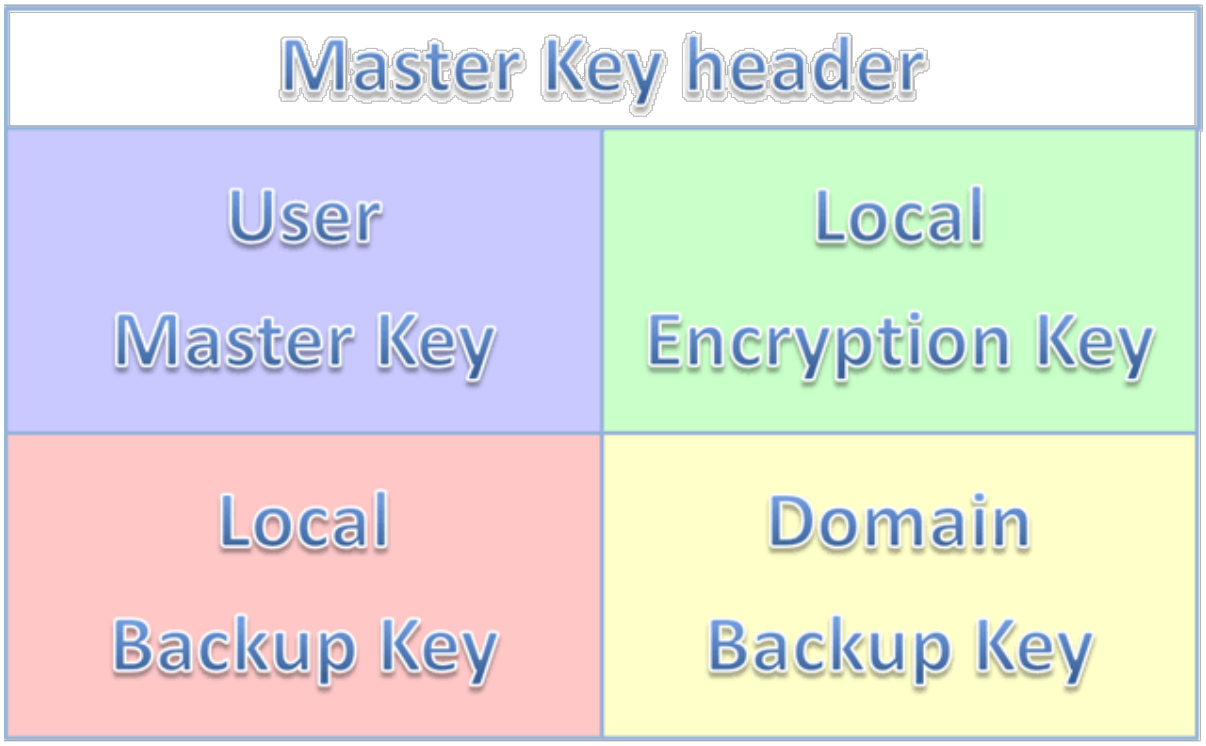

Figure 5. Master Key file structure in Windows 2000.

Microsoft strongly recommends upgrading Windows 2000 to a more current version. But if you have a good reason for not doing it, we are going to show you how to best protect your good old Win2K from a potential compromising. For that purpose, let's take a look at the local Master Key backup creation algorithm. A Win2K Master Key file conventionally consists of 5 units:

1. Header and system information
2. User's Master Key
3. Local backup encryption key
4. Local Master Key backup
5. Domain Master Key backup

Restoring user's Master Key from a local backup of begins with decoding the local backup encryption key. In the decryption process, we use the local computer account password (oh yes, it does have such an account), which is stored in the LSA secret named **DPAPI_SYSTEM**.

Once the local backup encryption key is obtained, we use it to decrypt the local backup of user's Master Key. It's as easy as 1-2-3. From this point of view, Win2K is more user-friendly, as even if user's password is deliberately erased, DPAPI can always restore the Master Key from the local backup. Hopefully, there's no need to explain the cost of that user-friendliness:

Any user that can access a PC under OS Windows 2000, which is not a domain member, can easily decrypt other user's encrypted DPAPI data, with no need for the logon passwords or **even password hash**!

To partially prevent that outrage, we must change the operating mode for SYSKEY. Run the SYSKEY utility (press the WIN + R shortcut on the keyboard, then type in syskey.exe and then click OK), replacing the SYSKEY storage location. Now, to decrypt the DPAPI_SYSTEM LSA secret, a potential malefactor would have to either know the SYSKEY bootup password or have the SYSKEY startup disk.

5 DPAPI Credential History

5.1 DPAPI Credential History

The proper functioning of DPAPI assumes keeping all of user's previous passwords. All the previous passwords are stored as hashes in a special container file named **CREDHIST**, located in the **%APPDATA%/Microsoft/Protect** folder.

5.2 CREDHIST Structure

If we look at the credential history file as a chain, each slot with a password hash would appear as a link of the chain with the following binary structure:

```
typedef struct _tagCREDENTIAL_HISTORY
{
        DWORD dwVersion;
        GUID guidLink;
        DWORD dwNextLinkSize;
        DWORD dwCredLinkType;
        ALG_ID algHash;
        DWORD dwPbkdf2IterationCount;
        DWORD dwSidSize;
        ALG_ID algCrypt;
        DWORD dwShaHashSize;
        DWORD dwNtHashSize;
        BYTE pSalt[0x10];
} CREDENTIAL_HISTORY, *PCREDENTIAL_HISTORY;
```

Data fields description:

DWORD dwVersion	current link version of the credential history chain
GUID guidLink	unique link identifier
DWORD dwNextLinkSize	size of next chain link
DWORD dwCredLinkType	type of password used for decrypting current link
ALG_ID algHash	hashing algorithm identifier in PBKDF2

DWORD dwPbkdf2IterationCount	number of iterations in PBKDF2
DWORD dwSidSize	security identifier size of the data owner
ALG_ID algCrypt	used data encryption algorithm
DWORD dwShaHashSize	size of SHA1 hash
DWORD dwNtHashSize	size of NTLM hash
BYTE pSalt[0x10]	random set of binary data used for encrypting
BYTE pSid[]	owner SID
BYTE pShaHash[]	SHA hash
BYTE pNtHash[]	NTLM hash

Figure 6 shows the data storage scheme used in CREDHIST.

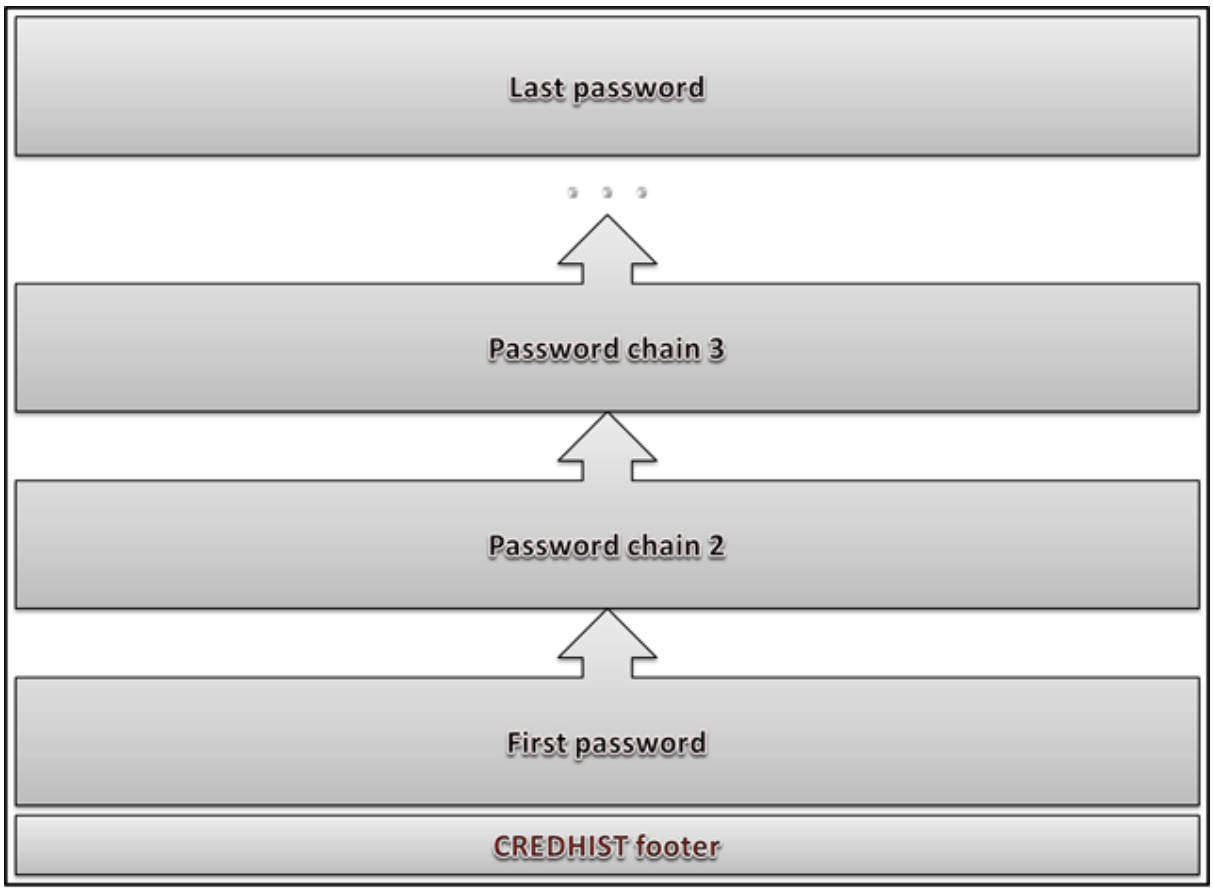

Figure 6. CREDHIST file structure.

5.3 CREDHIST Encryption

Slots with user's hashes are stored in series, one after another. Each password hash is encrypted with the preceding hash, and the first hash is encrypted with user's current password. Therefore, to decrypt the entire chain, one needs to know user's current password (see fig. 7.)

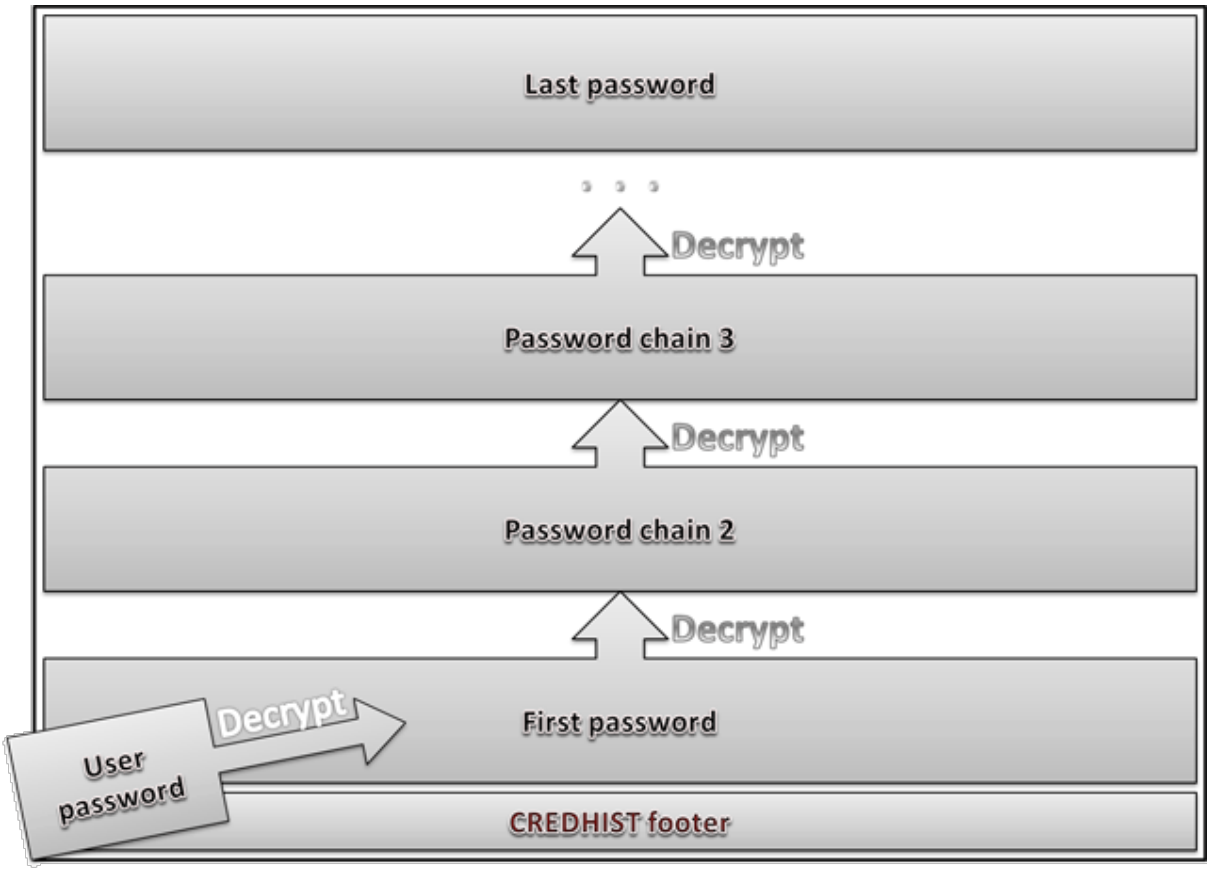

Figure 7. Credentials history decryption scheme in DPAPI.

If an error occurs when decrypting the Master Key using current password, DPAPI uses the logon password to decrypt the first hash in CREDHIST and attempts to decrypt the Master Key with the first CREDHIST hash. If the Master Key still resists, the first hash in the CREDHIST history decrypts the second hash, and the operation repeats.

Different operating systems use different default parameters and algorithms for encrypting credential history (see the table 3 below). In general, the encryption algorithm is similar to the one that is used for protecting Master Keys.

OS	Block cypher	Hash algorithm	PBKDF2 counter	Password guess speed (pwd/sec)
Windows XP	3DES	SHA1	4000	**76**
Windows Vista	3DES	SHA1	24000	**12**
Windows 7	AES256	SHA512	5600	**10**
Windows 10	AES256	SHA512	8000	**<10**

Table 3. Encryption algorithms used to decode credential history hases in DPAPI.

5.4 CREDHIST vulnerability

CREDHIST stores all the previous password hashes, even though the passwords are identical. Theoretically, there is a chance that user sets a password that has already been used before (let's call it duplicate). In that case, to recover current password, it would be sufficient to have the password from credential history, set between the current password and the duplicate.

Therefore, it is more expedient to use DPAPI with a Windows security policy that forbids entering duplicate passwords or improve DPAPI, so that it does not allow storing duplicates in CREDHIST.

6 What happens when changing Windows password

6.1 Password change and DPAPI

Password change in Windows is conventionally divided into several critical phases. First, the system checks the validity of the old password, then checks the new password for compliance with security policies and physically changes the password in the SAM or NTDS.DIT databases. In the second phase, if all goes well, it updates the internal password cache. Then there runs the DPAPI password change routine. And, finally, it updates the biometric logon credential for the user (Windows 7 and higher).

The DPAPI system is smart enough to handle various scenarios. Let's see what takes place inside DPAPI on password change. Here are the typical cases:

- Regular password change with supplying the old password (Windows XP and later)
- Password reset or overwrite; e.g., by administrator, without supplying old password (local PC under Win XP and later OSes)
- Domain user password reset (Windows 2000 +)
- Local PC user password reset in Windows 2000

6.2 Regular password change in Windows XP - Windows 7

Windows security policy assumes the regular change of user password, and DPAPI must provide user with the same level of access to personal encrypted data as before the password change.

That is achieved by a three-step synchronization of all user's Master Keys, on the first step of which all the Master Keys undergo the decryption with the old user password, on the second step all the keys are encrypted with the new password and saved to disk, and, finally, they are backed up if the respective option is selected.

The re-encryption of all Master Keys may take very long time; therefore, it is carried out in a separate thread within the LSA. An interesting Master Key synchronization gear has been added in Windows 7: DPAPI hashes all the Master Keys within user's profile and writes the hash obtained as the result of that process to the **SYNCHIST** file. The next time, to determine whether any changes took place, DPAPI again hashes the entire pack of the Master Keys and compares the obtained hash with the value stored in SYNCHIST, thus preventing the unnecessary and time-consuming operation of re-encryption.

The update of the CREDHIST file precedes the operation of synchronizing the Master Keys. The old user password hash is encrypted with the new one and placed to the beginning of the hash history chain in CREDHIST. If the credential history is empty, the system creates the first link of the hash chain.

In the end, the system updates the password reset disk if it was created earlier. To do so, the system looks for the public RSA key in the certificate storage, encrypts the new password with it and writes it to user's registry, replacing the old one.

6.3 Password reset on a stand-alone PC under Windows XP - Windows 7

This is the case when, for example, Administrator manually resets user's password, or the password is zeroed out using a password reset program.

Despite that the user maintains the ability to log on to the system, all data encrypted with DPAPI is no longer available (for example, network passwords, EFS keys, mail certificates, etc.), as the user's old password is unknown, and DPAPI is unable to decrypt the Master Key.

If after a forced password reset the old password is set again, DPAPI returns to its original state, and all the DPAPI blobs are available again.

This happens because DPAPI prudently keeps all copies of Master Keys even if they cannot be decrypted. Another way to recover the full access to the system (after a forced password reset) is using an earlier created password reset disk.

6.4 Forced domain user password reset

If DPAPI is used in the Active Directory environment (Windows 2000 +), the Master Keys store two backup copies. The first copy is protected by user's current password, and the second one is encrypted with the public RSA key that belongs to the domain controller.

On a forced user password reset, DPAPI is unable to decrypt the first copy of the Master Key. Nevertheless, access to all the DPAPI blobs will be recovered using the second backup copy. The workstation sends the encrypted backup copy of the Master Key to the domain controller. The domain controller decrypts the backup copy using its private RSA key and sends the decrypted Master Key back. The workstation encrypts the Master Key all over using the new user password and updates the file with both the copies and synchronizes the other files.

6.5 Password reset on a stand-alone PC in Win2K

On stand-alone computers running under Win2K, the DPAPI password reset disk is not used. The Master Key file stores two copies. The first copy is encrypted using user's password, and the second one is using the LSA secret of DPAPI_SYSTEM, stored in the registry and available only to the system account.

Once user password is forcibly reset, DPAPI is be unable to decrypt the first copy of the Master Key, so the system decrypts the second copy automatically. Then the decrypted copy is protected with user's new password and written back to the file. All the other Master Keys also undergo the synchronization.

From the end-user's point of view, the process of accessing confidential data, protected by DPAPI, is completely continuous. To ensure the adequate protection of confidential data by DPAPI on a stand-alone PC running under Windows 2000, it is recommended to modify the SYSKEY settings, so that the system would require entering the syskey startup password or provide a syskey startup diskette on system boot.

7 Recovering data from DPAPI blobs

7.1 Recovering data from DPAPI blobs

A set of tools for working with DPAPI available in the Windows Password Recovery offers broad opportunities for recovering passwords and data encrypted with DPAPI in the off-line mode, i.e. without loading user's profile. Let's review the typical scenarios of using the software.

7.2 Finding DPAPI blobs on disk

In windows, DPAPI blobs can be stored anywhere: in binary or text files, databases, registry, Active Directory, etc. Before beginning to decrypt a DPAPI blob, we must first "pull" it out and put in a proper form. To do so, we would need a search tool for finding DPAPI blobs on disk.

Launch the utility and select the source folder to be searched. Suppose, the source folder is C:/Program Data/Microsoft/Wlansvc. It contains wireless connection passwords in Windows 7.

Passwords encrypted with DPAPI are stored in the text format in XML files; therefore, we make the respective setting to search for text blobs (the option is enabled by default). Once the folder for storing found files is selected, hit the Start button.

Found DPAPI blobs will be stored in the target folder under the respective names. For example, if the program finds a DPAPI blob in file xxx.xml, it will be converted to the binary format and saved in the target folder as xxx.xml.001. If in file yyy.xml the program finds two blobs, they will be saved as yyy.xml.002 and yyy.xml.003.

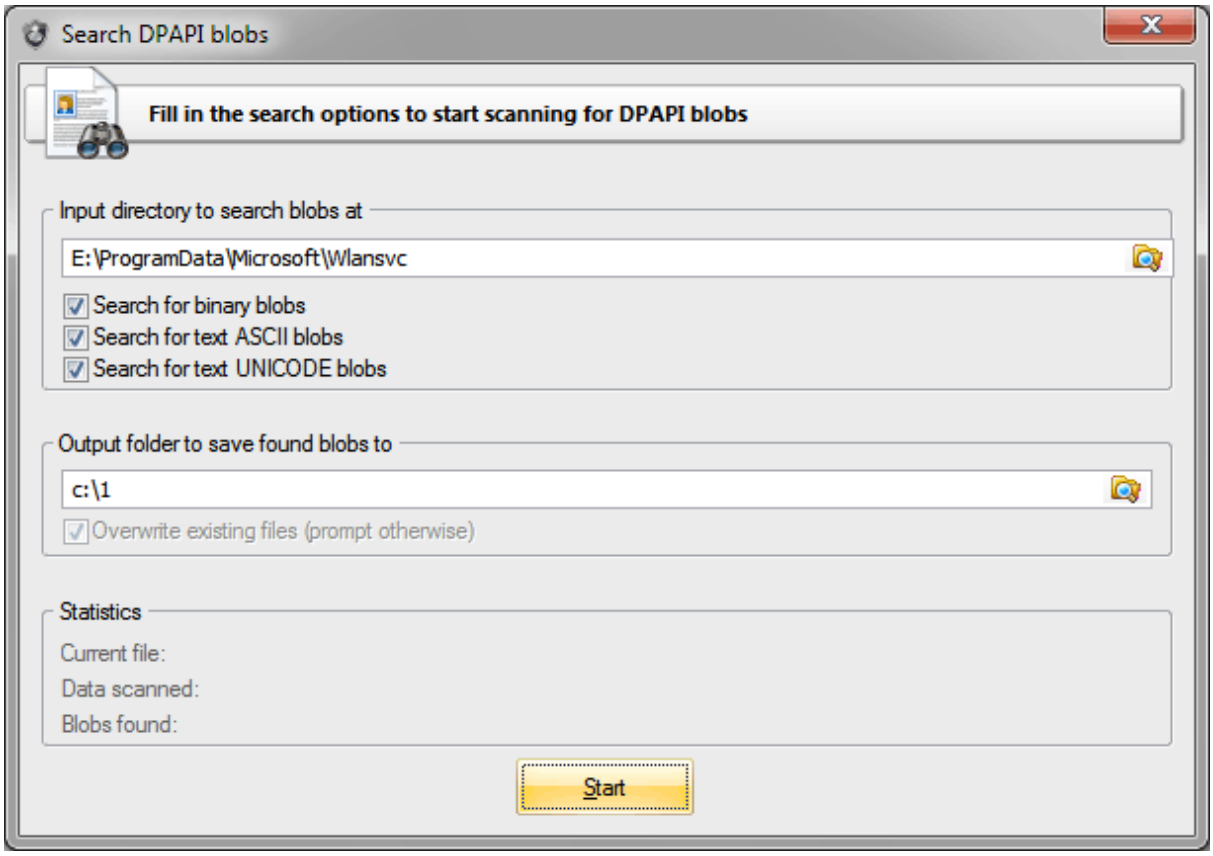

Figure 8. Looking for WiFi passwords.

7.3 Finding DPAPI blobs in registry and Active Directory

Finding DPAPI blobs on disk is not that difficult. But what if you need to scan current user's registry or an Active Directory database? Access to those files is forbidden to all, including administrators.

It's difficult, but, fortunately, not impossible. For that purpose, we take advantage of a handy tool for registry and Active Directory backup, which is easy to operate even for newbies.

Launch it and then select what and where we to be backed up: registry or Active Directory. Note that parsing an Active Directory database may take long time. Now we can proceed to the DPAPI blob search utility and specify the path to the folder where we have saved the registry or ntds.dit database.

7.4 Recovering wireless connection passwords in Windows 7

Decrypting wireless connection passwords is preceded by looking up the respective DPAPI blobs in the above described **C:/Program Data/Microsoft/Wlansvc** folder. So, here we are going to skip its yet another description.

Launch the DPAPI decryption utility and specify the path to the extracted DPAPI blob. Wireless connection passwords are protected with the CRYPTPROTECT_LOCAL_MACHINE flag, so to recover them we would need the system Master Key from folder **E:/Windows/System32/Microsoft/Protect/S-1-5-18/User**.

In that folder, find the required file with the Master Key. By default, the open file dialog shows only the file we need. Path to CREDHIST in our case is not necessary, so we leave that field empty.

Once moved to the third step of the application, we mark that we don't need the user's password. Instead, we need two registry files, **SYSTEM** and **SECURITY**. Well, we specify them too. Note that you must first unlock current registry files (i.e. back them up). In the user's SID field, the program automatically places the system's SID: S-1-5-18. Leave that field and finish the decryption by obtaining the original plain-text wireless connection password in Windows 7.

Easy? Let's go further.

7.5 Recovering Facebook passwords stored in Internet Explorer

Now the problem is a bit more complicated. Let's try to recover the password for a Facebook account in Internet Explorer. To do so, we may have to do quite a bit of "tambourine dancing", so stock up on patience in advance.

Internet Explorer auto-complete passwords are stored in the registry hive **HKCU/Software/Microsoft/Internet Explorer/IntelliForms/Storage2**. Run Regedit and open that hive. There, you will probably see a number of binary records. We are going to need just one of them, named **F6FFE33B9EF4D7CB8F5A2F41F3222D21E131ED787A**.

If you do not have such a record, try creating a test password on Facebook: open the page at http://www.facebook.com/ (the www is mandatory), enter any e-mail address and the test password and then click the sign in button. When IE kindly offers to save the new password, do that. Naturally, the Facebook website will return an error, but that doesn't matter, the password has been saved.

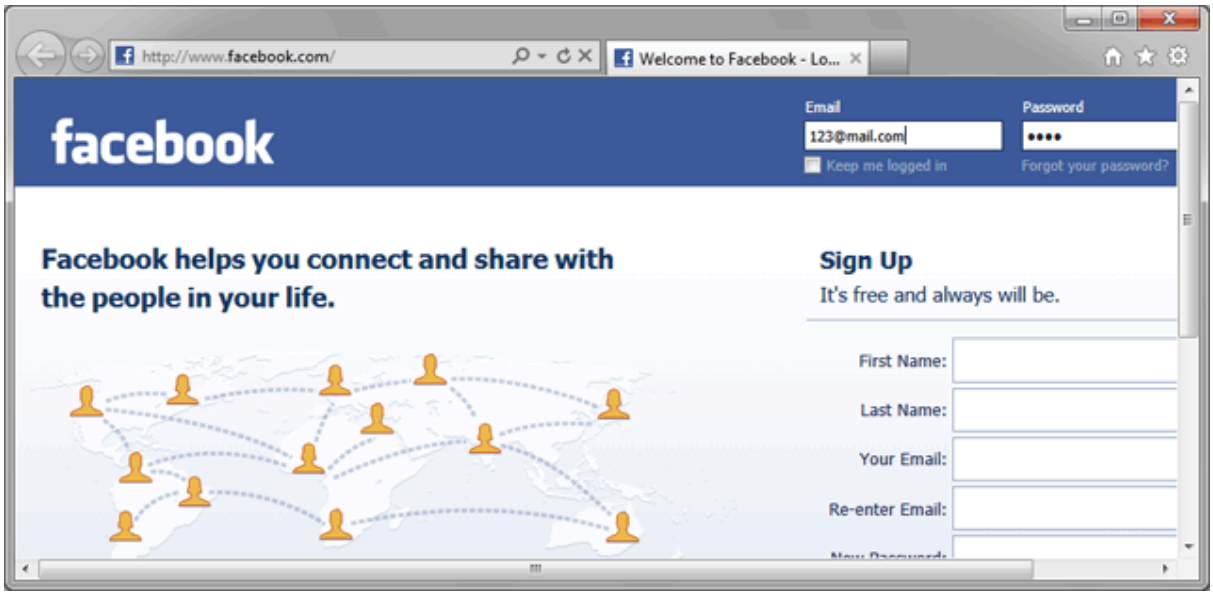

Figure 9. Creating Facebook test password.

Switch back to regedit and press F5 to update the data. The registry key that stores IE passwords must get a new record named F6FFE33B9EF4D7CB8F5A2F41F3222D21E131ED787A. This tricky name is nothing but a SHA hash of the website name.

When you open it, you will see a DPAPI blob with encrypted login and password to the website. Let's export the DPAPI blob to a text file. To do so, open cmd and run the following command:

REG QUERY "HKCU\Software\Microsoft\Internet Explorer\IntelliForms\Storage2" /v F6FFE33B9EF4D7CB8F5A2F41F3222D21E131ED787A > c:\test\fb.txt

The quotes are mandatory! Our password will be saved in the ASCII format to folder c:test. You are free to set any other folder; that doesn't really matter. Now, all we need to do is to extract the binary DPAPI blob we need from the ASCII file fb.txt.

Launch the search tool, specify the source folder c:test, set the search text files option and get the binary file fb.txt.001 at the output. Finally, we can move on to the actual password recovery.

Open the respective utility and set the path to fb.txt.001. On the second step, specify the Master File, which the blob was encrypted with. All the user's Master Keys are stored in the folder **%AppData %/Roaming/Microsoft/Protect/%SID%**. One level higher, there is the **CREDHIST** file. We may be able to use it too.

The most interesting stuff begins when we move on to the third step of the application's wizard. To decrypt the Facebook password, we will need to properly set the following three parameters: user's **SID**, his **logon password** and the secret the DPAPI blob was encrypted with. The program normally extracts user's SID automatically from path to the Master Key. If that hasn't happened, you will have to obtain that parameter by hand (for example, by analyzing the CREDHIST file). Concerning user's logon password everything is clear too. But what are we supposed to do about the secret?

For the secret, when encrypting password, IE uses the name of the web page the password was entered on. The page must be entered in lowercase, as UNICODE, and terminate with a zero. If you have a HEX editor at hand and have sufficient skills to comfortably work in it, try creating that secret manually. Otherwise, simply download it here.

So, having specified the path to the file that contains the secret in the entropy setting, we finally finish our epic of recovering a Facebook password. Having decrypted it, you will see that the password, login and other system information are merged into a single data structure (Fig. 10). Don't let this bother you.

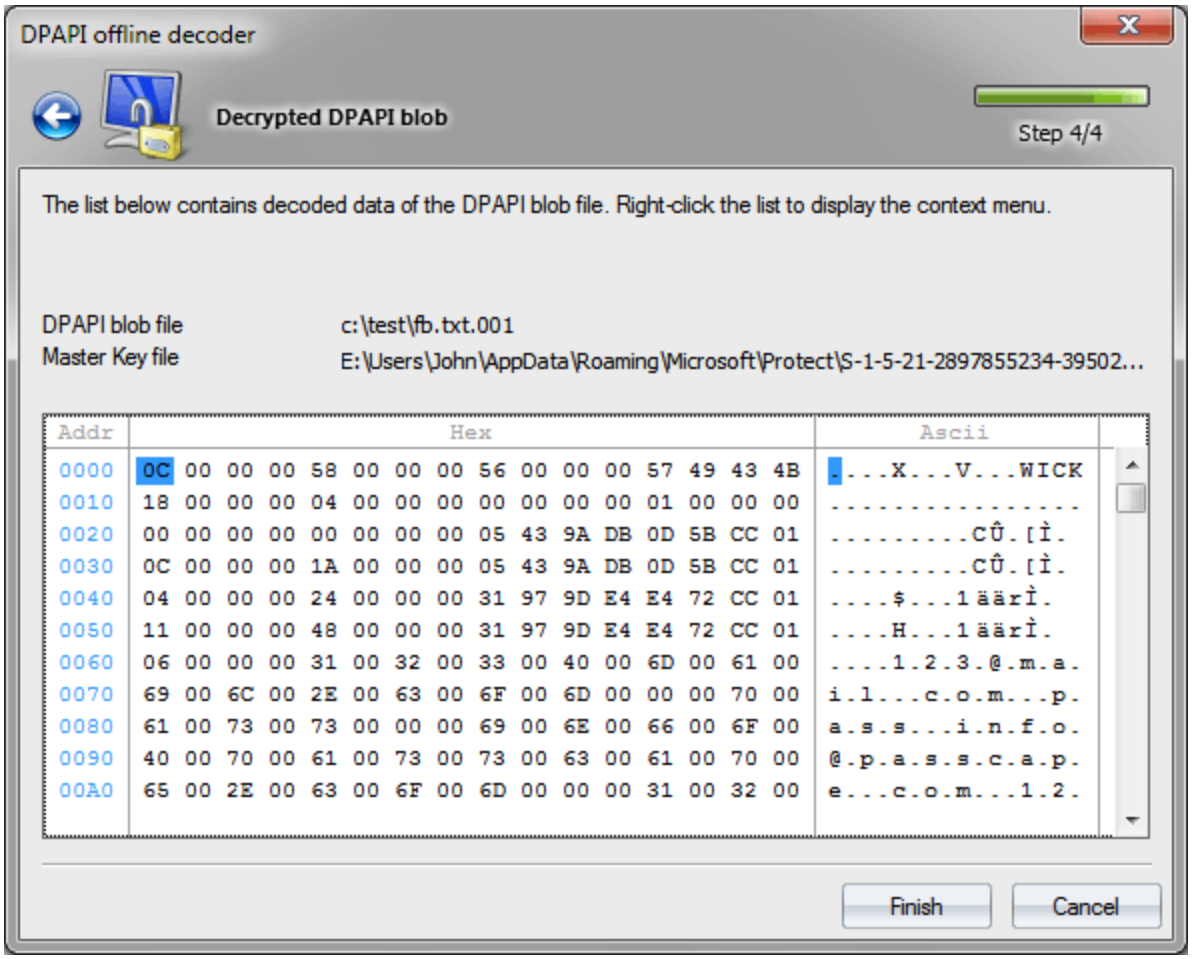

Figure 10. Decrypted Facebook password.

7.6 Recovering user password without loading hashes from SAM/NTDS.DIT

For the first time ever, Windows Password Recovery allows to recover the logon password without loading user's hashes from SAM or NTDS.DIT. That was made possible because user's password participates in the Master Key encryption process. Consequently, the Master Key can be used for validating the password.

To illustrate the above, run the Master Key analysis tool and specify the path to a file with a Master Key. After a successful parsing of the Master Key, the program will show its internal structure. Right-click on it, then select the validate password using dictionary item and specify the path to any dictionary.

Despite its revolutionary character, the recovery of user's password from the Master Key is not of any practical benefit. For example, in Windows 7, the password validation goes at the speed of about 10 p/s. Theoretically, the speed can be increased by a few orders of magnitude by optimizing the encryption

algorithms and engaging GPU in the validation. But even after that, the speed of password mining by Master Key compared to NTLM hash would seem ludicrous. In terms of security, DPAPI is all set.

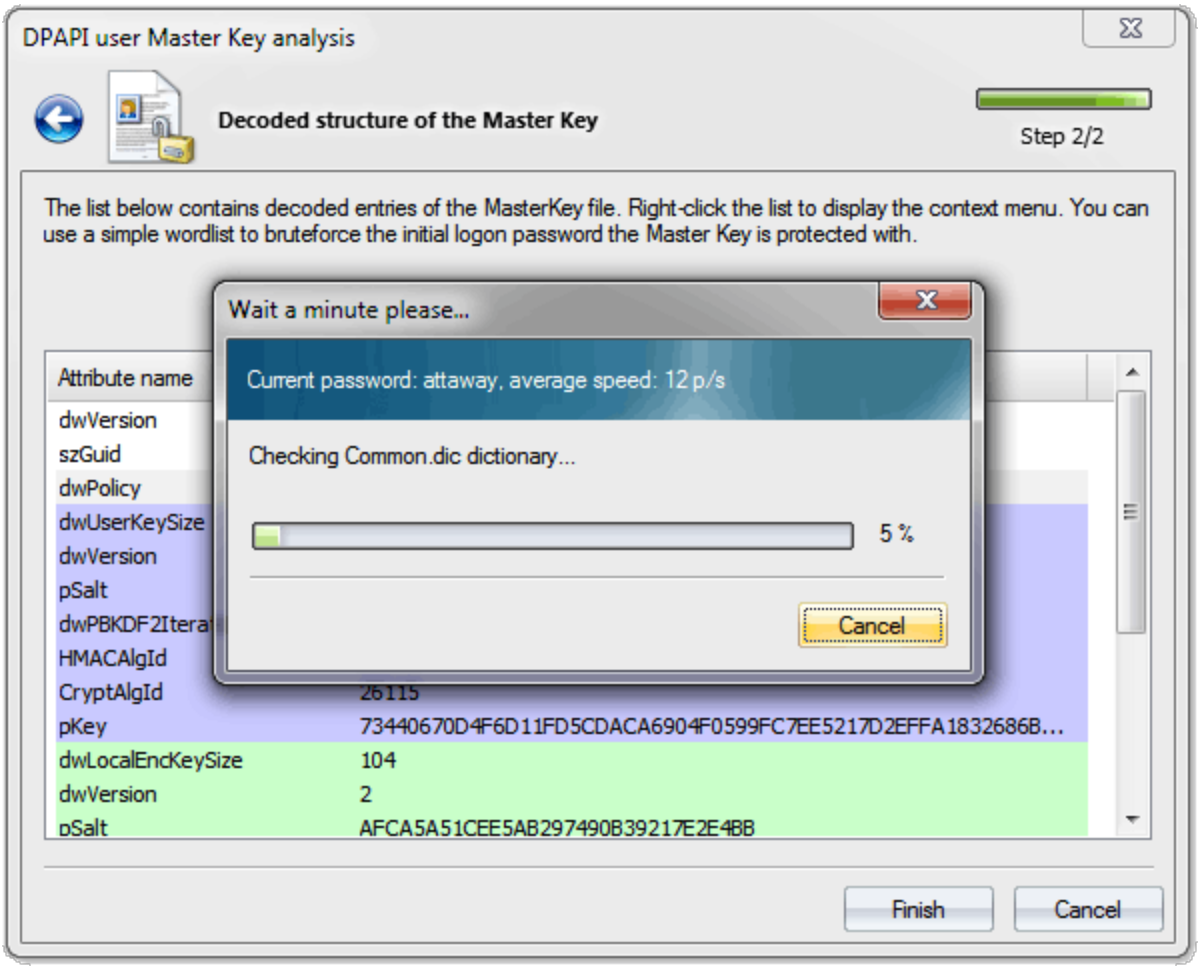

Figure 11. Recovering user's logon password by analysing DPAPI Master Key in Windows Vista.

7.7 Decrypting credential history hashes from DPAPI

Open the credential history dump tool and specify the path to the CREDHIST file, stored in the **%AppData%/Roaming/Microsoft/Protect** folder. If the file stores the previous user passwords, the "**CREDHIST key chains**" field should have a non-zero value.

Previous user passwords are stored as SHA1 and NTLM hashes. The application has a respective switch. If you select current user's CREDHIST file, entering the logon password isn't required; the program will automatically fetch it from Windows cache, simply make the respective setting. Otherwise, you would have to enter current user's password manually.

The utility supports partial dump. That means that in case the current user password is unknown, but one of the old user password is known, the program can decrypt the hashes that were used earlier, i.e. before the old password was entered.

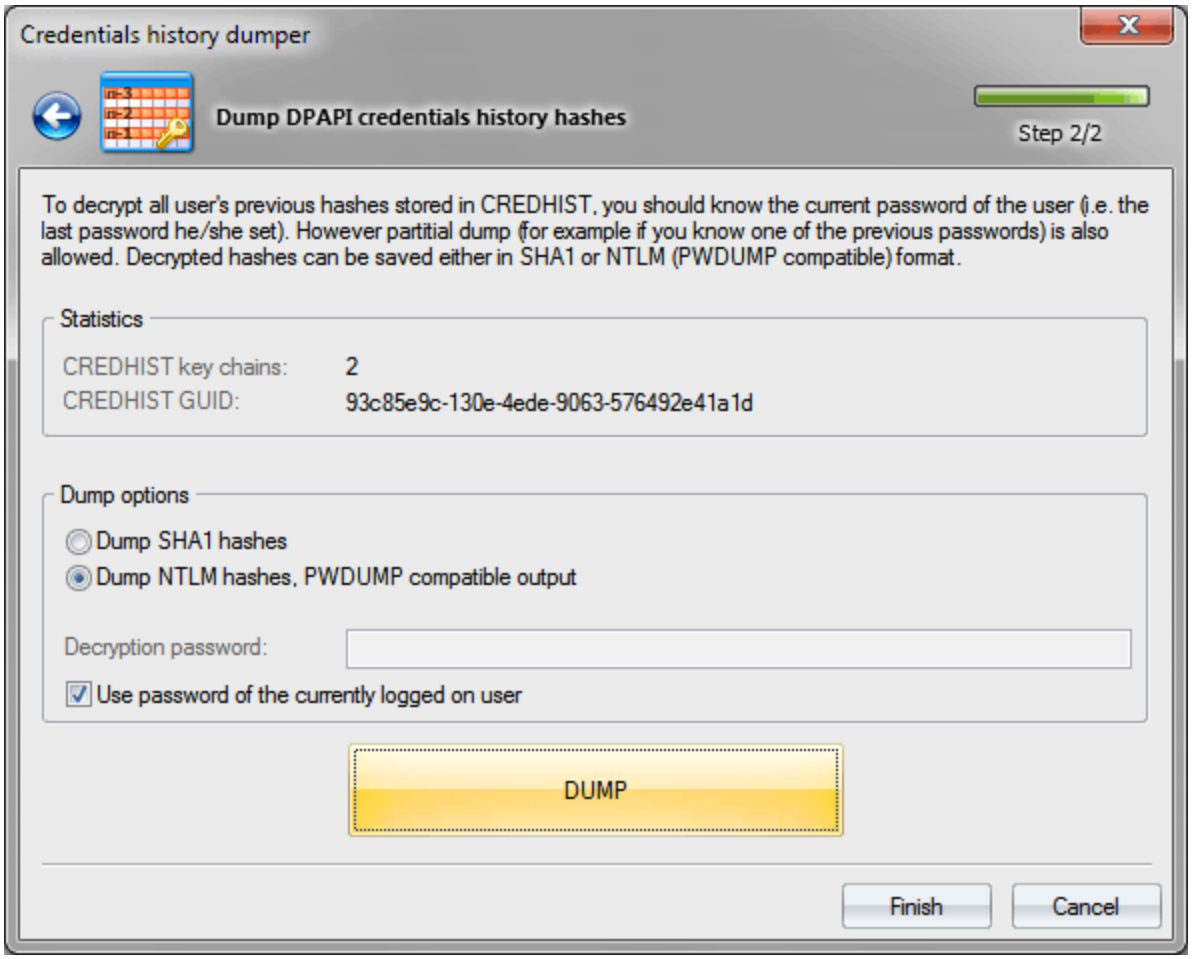

Figure 12. Dumping DPAPI credential history passwords.

8 Conclusion

DPAPI deserves such close attention at least for the fact that it's the only password-based system that provides appropriate and thoroughly thought out protection of user's personal data. None of the operating systems has a more viable alternative to DPAPI!

We should, perhaps, mention that the first implementation of DPAPI had a number of serious flaws, which could enable a potential malefactor to easily compromise user's data protected by DPAPI.

The first pancake is known to be always lumpy. In all the sequel operating systems, beginning with Windows XP, those vulnerabilities have not merely been eliminated; the entire DPAPI system has undergone a major revision. In particular, it has adopted new encryption algorithms; that has made the Master Key password lookup speed about 1000 (!) times slower. Master Key encryption errors that potentially allowed any user to gain access to any files encrypted by EFS were fixed. The local Master Key backup system has been replaced with the password reset disk, etc.

Overall, the DPAPI encryption system has become more robust, powerful, meeting the stringent requirements of password security.