DPAPI Secrets

security analysis and data recovery in DPAPI

Passcape Software

1 摘要

在本文中，我们试图分析DPAPI的操作，回顾DPAPI的未记录的结构和加密算法，理解和描述系统的内部运作。

本文提出了世界上第一个关于DPAPI运行逻辑和所有未记录的结构的完整描述（虽然没有声称是通用的），包括DPAPI blobs、主密钥和证书历史文件。

我们还试图指出最新版本的DPAPI的一些功能缺陷，以及消除这些缺陷的可能方法。此外，对与Windows 2000一起发布的第一版DPAPI的实现进行了深入的分析，发现了一些严重的漏洞，使整个系统的安全性受到质疑。

人们非常关注当用户配置文件无法加载时DPAPI数据的恢复问题。有史以来第一次，我们在不使用SAM或NTDS.DIT文件的情况下实际测试了用户登录密码恢复算法。

DPAPI的运行情况是通过一组6个实用程序进行分析的，这些程序被整合到一个最强大的审计Windows密码的应用程序中，Windows Password Recovery. 我们希望这些工具以及本文中的某些信息不仅对专家和犯罪学家感兴趣，而且对计算机安全领域的所有研究人员也感兴趣。

2 DPAPI架构和安全

2.1 什么是DPAPI

从Windows 2000开始，微软为其操作系统提供了一个特殊的数据保护接口，称为数据保护应用编程接口（DPAPI）。DPAPI目前很普遍，在许多Windows应用程序和子系统中使用。例如，在文件加密系统中，用于存储无线连接密码，在Windows凭证管理器、Internet Explorer、Outlook、Skype、Windows CardSpace、Windows Vault、Google Chrome等。DPAPI之所以在程序员中流行，首先是因为它使用简单，因为它只由几个用于加密和解密数据的函数组成，即CryptProtectData和CryptUnprotectData。

尽管它表面上很简单，但DPAPI的技术实现却相当复杂，这些函数的操作逻辑很像欢快的童谣"The House that Jack Built"。也许，这就是为什么DPAPI的内部结构和运行原理长期以来一直被关在幕后的原因。

2003年，Passcape软件公司首次对DPAPI进行了分析。2005年，该公司发布了第一个基于该恢复的商业应用程序（Outlook Password Recovery），它可以离线解密DPAPI blobs，即无需登录到所有者的帐户。Passcape提出的解密算法只是模拟DPAPI的操作，所以DPAPI系统并没有被破坏。

随着全新版本的Windows密码恢复工具的发布，离线DPAPI解密算法又有了新的突破。现在，除了解密DPAPI blobs，该程序还可以分析它们，在磁盘上找到它们，解密主密钥并验证其密码，从DPAPI中提取用户的凭证历史哈希值，以及更多。

2.2 DPAPI能保护什么？

DPAPI被用来保护以下个人数据：

- Internet Explorer、Google *Chrome中的密码和表格自动填写数据
- Outlook、Windows Mail、Windows Mail等中的电子邮件账户密码
- 内部FTP管理器的账户密码
- 共享文件夹和资源访问密码
- 无线网络账户密钥和密码
- Windows CardSpace和Windows Vault中的加密密钥
- 远程桌面连接密码，.NET Passport
- 加密文件系统（EFS）的私钥，加密邮件S-MIME，其他用户的证书，互联网信息服务中的SSL/TLS
- EAP/TLS和802.1x（VPN和WiFi认证）
- 凭证管理器中的网络密码
- 用API函数CryptProtectData编程保护任何应用程序中的个人数据。例如，在Skype、Windows权限管理服务、Windows Media、MSN messenger、Google Talk等。

使用DPAPI保护数据的一个成功而巧妙的例子是Internet Explorer中自动完成密码加密算法的实现。为了加密某个网页的登录名和密码，它调用CryptProtectData函数，在可选的熵参数中，它指定了网页的地址。因此，除非人们知道输入密码的原始URL，否则没有人，甚至IE浏览器本身，可以解密该数据回来。

2.3 DPAPI架构

从Windows 2000开始，正如前面提到的，任何应用程序都可以通过简单地调用CryptProtectData函数来保护其个人数据（例如，密码），该函数返回一个 "非透明 "的二进制结构，也被称为DPAPI blob。根据微软的定义，"非透明 "意味着除了应用程序的原始加密数据外，它还包含其他辅助内容，是解密数据的必要条件。尽管DPAPI blobs的结构没有被记录下来，但随着新版Windows Password Recovery的发布，有史以来第一次有可能对DPAPI blobs进行离线解密并彻底分析。

其他API函数，CryptUnprotectData，顾名思义，其工作原理与此类似，但是反过来，在输入端获得一个加密的DPAPI blob，并将解码的数据返回给应用程序。

图1显示了DPAPI的流程图。

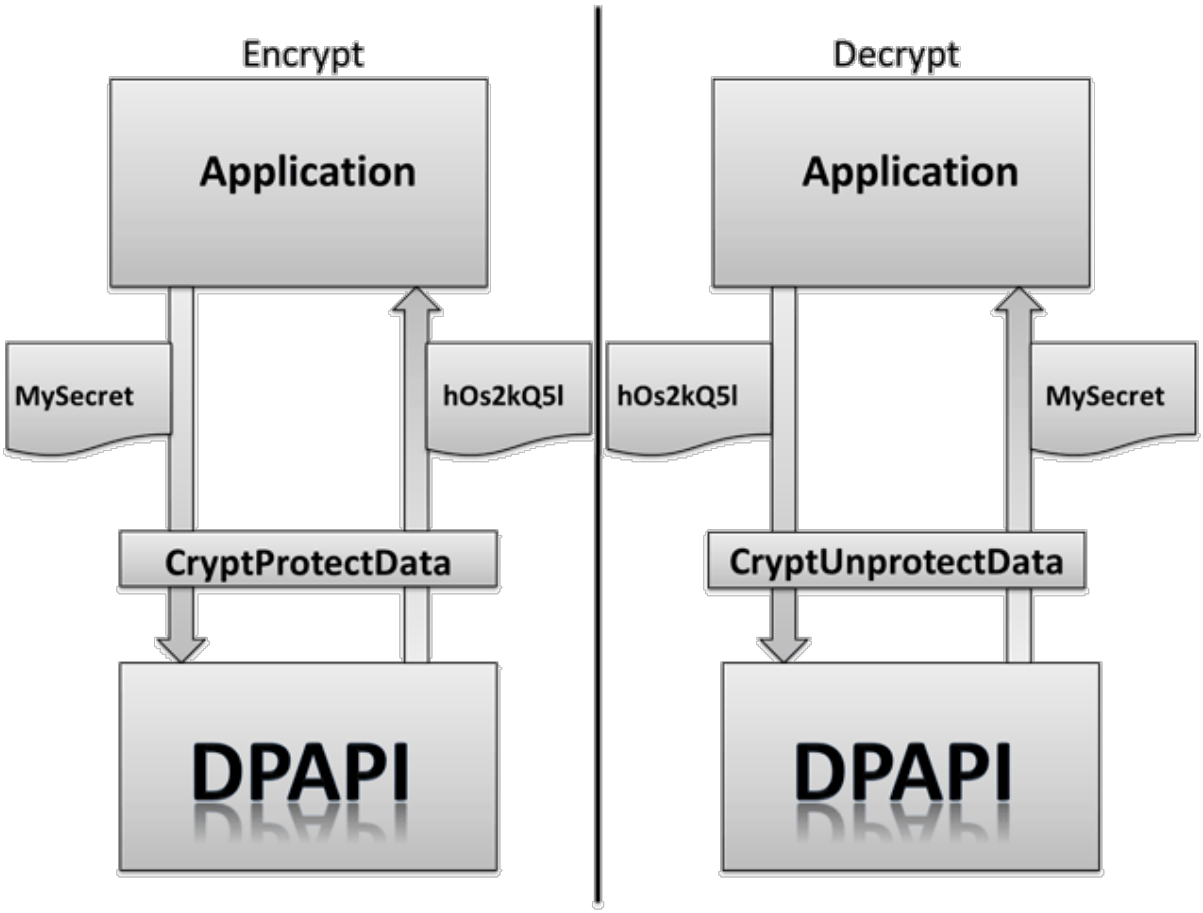

图 **1. DPAPI流程图。**

DPAPI的加密是基于用户密码的；因此，在一个账户下加密的数据不能在其他账户下被解密。此外，DPAPI允许通过设置一个额外的秘密（熵）来限制对数据的访问，甚至在一个账户内。因此，除非它知道额外的秘密，否则一个应用程序不能访问其他应用程序所保护的数据。

每个实现DPAPI接口的程序员都必须认识到，系统只对数据进行加密。一个应用程序必须提供一个解决方案来存储返回的DPAPI blobs，包括可靠地隐藏可选的熵数据，如果它被使用的话。

2.4 DPAPI安全

DPAPI的创建考虑到了安全方面的许多方面。这里列出了它区别于类似系统的主要特征：

- PAPI在CryptoAPI的基础上工作。你可以创建你自己的加密服务提供商，并将其与DPAPI一起使用。
- DPAPI使用成熟的加密算法。例如，Windows 7默认使用CBC模式的AES256加密，SHA512用于散列，PBKDF2作为基于密码的密钥衍生程序。
- 所有的算法以及它们的密钥长度都可以从注册表中进行配置，只有具有管理员权限的用户才能访问。例如，PBKDF2功能的迭代次数在以下注册表键中设置：HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Protect\Providers\%GUID%\MasterKeyIterationCount

- 所有可以从注册表配置的参数都有其限制，不能小于默认值。例如，在Windows XP中，MasterKeyIterationCount的默认值是4000。
- 所有关键数据都受到完整性检查的保护。
- 所有算法中的所有加密密钥的大小都足以防止可能的暴力攻击。
- 出于同样的目的，PBKDF2有一个相当大的迭代计数器。唯一的例外是DPAPI的第一个实现。下表显示，在Windows 2000中，主密钥密码搜索速度足以泄露用户密码。
- 出于安全考虑，主密钥的有效性是有限的。然而，该机制的实现也有弱点，这将在下面讨论。
- 与域控制器的所有通信都是通过一个安全的RPC通道进行的。
- 缓存数据既不写到磁盘，也不写到可交换的RAM。在运行每个加密功能后，中间密钥和值在堆栈中被重置。顺便说一下，在分析SYSKEY工具时，Passcape专家发现了一个与不正确的堆栈重置有关的有趣的错误，这导致在Windows 2000、Windows XP和Windows 2003下运行的一些系统中，系统加密密钥（SYSKEY）的一部分被存储为纯文本，供系统的所有用户使用。唉，这种情况有时会发生。

然而，有一些细微的差别，对这些差别的忽视会削弱DPAPI安全系统：

- CRYPTPROTECT_LOCAL_MACHINE - 在CryptProtectData函数中设置的标志，在某些情况下相当方便，实际上会损害用户数据，不能提供足够的保护。当它被设置时，加密不依赖于用户的登录密码。另一方面，这种行为与DPAPI的概念并不矛盾，而是与接口实现的特殊性有关。
- DPAPI可以被配置为在Windows 2000的兼容模式下运行，此时备份的主密钥被存储在本地，并可以使用LSA秘密解密。因此，在理论上，所有的DPAPI blobs可以在不知道用户密码的情况下被解密。这就是Windows Password Recovery在实践中所展示的。见第2张截图。然而，要激活这种模式，需要管理员访问具有DPAPI配置的注册表键。

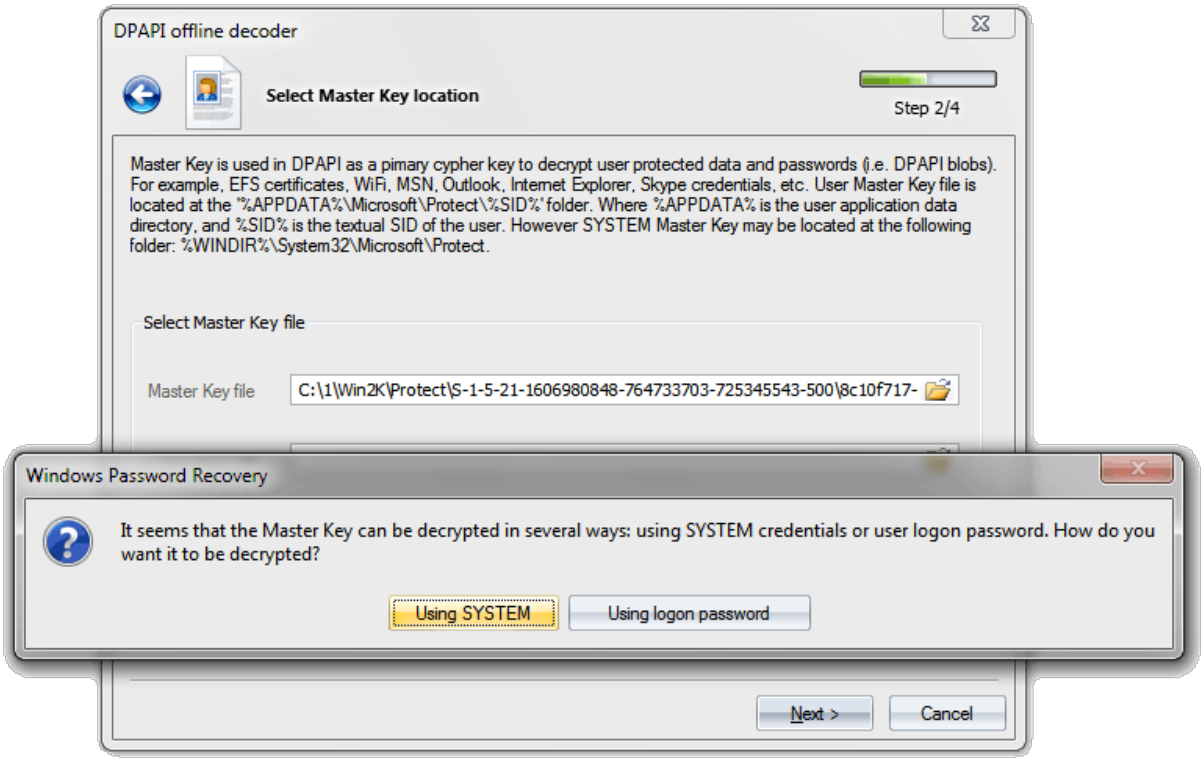

图 **2. 在Windows 2000中，在不知道原始用户登录密码的情况下解密DPAPI blob。**

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**

表1. DPAPI中使用的默认算法。

如果作为一个整体，DPAPI也许提供了迄今为止最强大的用户数据的保护和安全。

2.5 DPAPI配置

DPAPI运行在Crypto API之上，可以使用任何加密服务提供者。加密服务商的选择是在以下注册表键中定义的：

HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Protect\Providers
首选

默认情况下，系统使用微软名为df9d8cd0-1501-11d1-8c7a-00c04fc297eb的加密提供者，在psbase.dll库中实现。
加密是通过修改以下注册表键的相应值来配置的。
HKEY_LOCAL_MACHINE/Software/Microsoft/Cryptography/Protect/Providers/%GUID%。

其中%GUID%是服务提供商的唯一标识符。默认情况下，DPAPI设置键对应的是HKEY_LOCAL_MACHINE \Software\Microsoft\Cryptography\Protect\Providers\df9d8cd0-1501-11d1-8c7a-00c04fc297eb

只有拥有管理员权限的用户才能修改注册表的这个蜂巢。下面是这个蜂巢的一些键的描述。

MasterKeyIterationCount
函数PBKDF2的迭代计数器，不能比默认值小。该密钥生成算法用于加密用户的主密钥和凭证历史。

MasterKeyLegacyCompliance
将注册表键设置为1，使DPAPI在备份/恢复主密钥时的行为与Windows 2000相似。

MasterKeyLegacyNt4Domain
由于DPAPI的备份/恢复支持在NT4上不存在，DPAPI提供了一个解决方法，以避免丢失其数据，并强制客户机使用W2K风格的主密钥备份/恢复。为此，你应该把MasterKeyLegacyNt4Domain参数设置为1。

分布式备份密钥
决定备份或恢复域密钥时的兼容性设置。例如，如果域的功能级别被设置为Windows 2000本地。

恢复版本
决定备份和恢复主密钥本地副本时的版本兼容性。默认版本是2；Windows 7和Windows Server 2008 R2可以通过将DWORD注册表值设置为3来配置为使用版本3。

保护政策
默认的数据保护策略标志。与主密钥头中的dwPolicy标志相对应。

Encr Alg - Encr Alg 密钥大小
控制DPAPI中使用的加密算法类型和该算法中使用的密钥大小的一对数值。

MAC Alg - MAC Alg密钥大小
用于设置DPAPI数据完整性控制算法的密钥。

3 在DPAPI中进行数据加密

3.1 CryptProtectData和CryptUnprotectData API函数的描述

这些函数的参数基本相同，所以我们只看一下CryptProtectData，在C++中它的声明如下：

```
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
);
```

函数字段说明：

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 CryptProtectData/CryptUnprotectData函数的使用实例

为了方便运行测试，也为了提供一个示例，我们创建了两个名称不同的简单实用程序，它们实现的仅仅是对DPAPI函数的封装。因此，可以从命令行调用DPAPI函数。下面是CryptProtectData的C++源代码。这些程序编译后的EXE文件也可以在CryptProtectData, CryptUnprotectData下载。

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 在DPAPI中进行数据加密

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和CryptUnprotectData基本上只是包装器函数，因为实际的数据加密是在系统上下文中运行的。DPAPI的神奇部分发生在LSA中，并且对陌生人的眼睛是隐藏的。

加密过程漫长而枯燥;这可能会让人想起“共同生活20年后履行婚姻义务”。一开始，我们取用户的密码;然后，通过对其应用PBKDB2加密标准，我们得到主密钥加密密钥。现在，我们用这个加密密钥来解密主密钥。如果解密失败，则使用凭据历史的第一个散列CREDHIST来解密用户密码，而CREDHIST反过来也获得主密钥加密密钥，并尝试再次解密主密钥。如果该尝试也以失败告终，则解密下一个凭证历史散列，依此类推。该操作将继续进行，直到主密钥被解密或凭据历史记录的项耗尽为止。

为了避免每次调用CryptProtectData / CryptUnprotectData时提示密码，Windows会缓存密码，并在用户配置文件生命周期内将其保存在LSA的深处。这就是为什么DPAPI只在线工作，只有在用户登录到系统后。

2005年，Passcape软件公司首次推出了一种算法，该算法适用于该公司所有的产品，并模拟DPAPI的离线运行。换句话说，要解密DPAPI blob，不再需要加载用户的上下文;尽管密码是必须的。DPAPI的离线解密绝不会使系统更容易受到攻击;同时，它对研究人员、专家和犯罪学家仍然有帮助。

因此，用户的主密钥驻留在一个特殊的容器文件中，该文件可以存储主密钥的额外副本(备份)。当主副本由于某种原因无法解密时，将使用主密钥备份。例如，强制重置密码后。主密钥是DPAPI密码学中的一个关键元素;因此，它的长度为512位，加密时只使用经过验证的算法。见表1。

在DPAPI的第一个实现中，主密钥的解密需要用户密码的NTLM散列。这是一个重大问题:不，这是DPAPI开发人员最大的错误，它否定了系统的安全性。这一切都是因为它在技术上允许潜在的不法分子解密主密钥，从而通过直接从SAM文件获得NTLM散列来解密任何DPAPI blob;实际的密码甚至都不需要!在第二个DPAPI的当前版本中，这个错误被修复了;现在主密钥使用SHA1散列加密。

一旦主密钥被成功解码，它的源512位数据将参与获取DPAPI blob的对称加密密钥。为此，如果指定了主密钥，则它与随机数据(然后存储在DPAPI blob中)和熵“混合”。这样，我们就实现了每个DPAPI blob的加密密钥的三维唯一性。

最后是对源数据的实际加密。Windows 7操作系统采用AES256算法对数据进行加密。这个复杂过程的所有阶段都涉及到使用“盐”，即一组随机数据，统一加密密钥，从而防止彩虹表的潜在暴力攻击。

顺便说一下，很少有人知道Windows 7的第一个版本在AES加密方面有严重的问题。简单地说，这种类型的加密根本不起作用。幸运的是，这个问题很快就解决了。

一旦数据被加密，它将与其他系统信息一起被分组到单个结构中，并作为DPAPI blob发送回应用程序。DPAPI blob的结构没有文档记录;但是，Passcape的研究人员能够完全逆向它。这就是它。

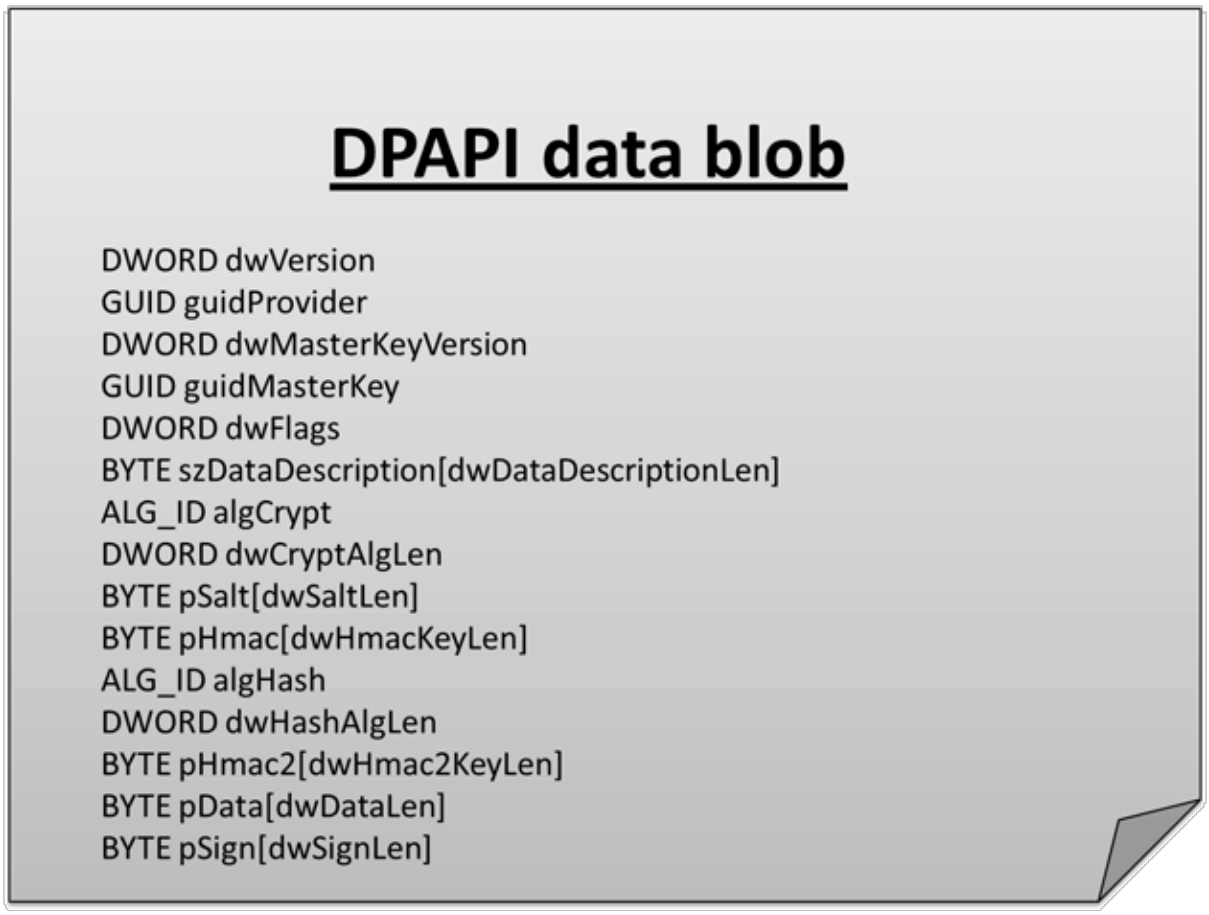

图3 DPAPI blob的无文件结构。

4 DPAPI主密钥

4.1 什么是DPAPI主密钥

用户的主密钥是一个二进制文件，它包含用于在所有DPAPI blobs中创建主加密密钥的加密数据。由于主密钥对用户的机密数据进行加密，所以密钥本身需要认真保护。用户的密码被有意义地选为保护主密钥的源数据。

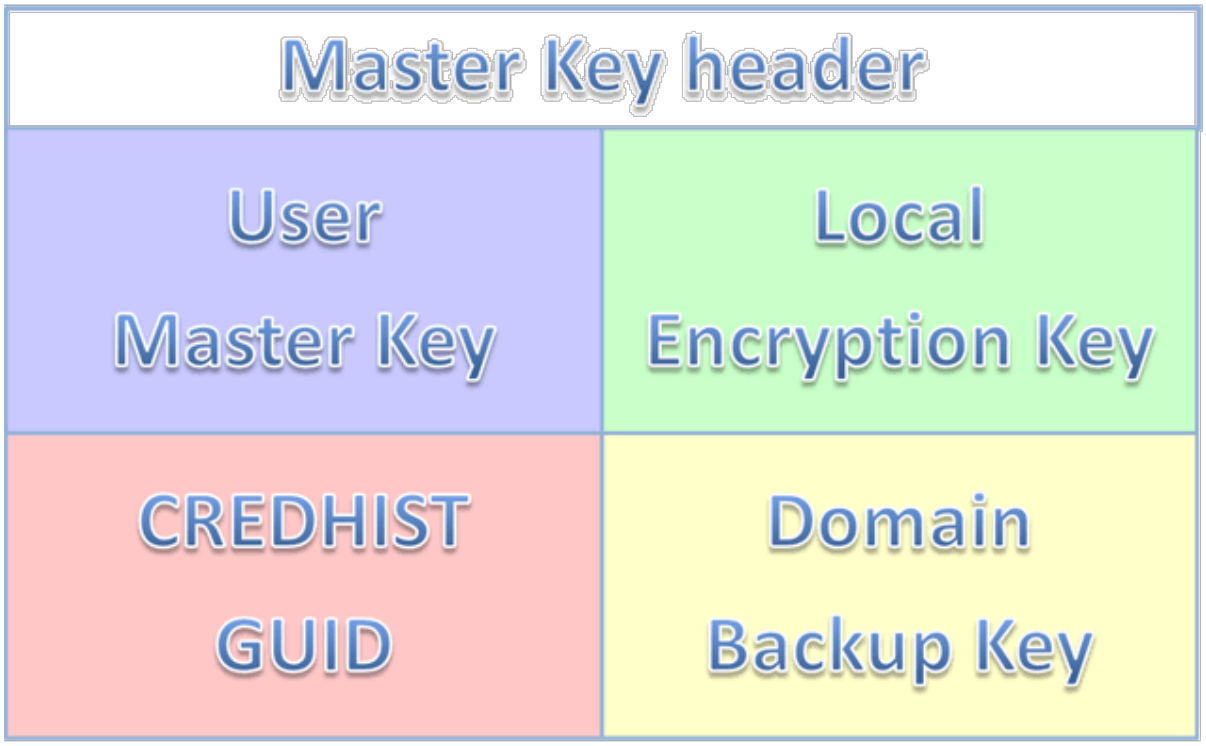

图 **4.** 主密钥结构。

4.2 主密钥结构

如图4所示，一个主密钥文件由5个单元组成。

1. 文件头和系统信息
2. 用户的主密钥
3. 本地备份加密密钥
4. 独特的CREDHIST文件标识符
5. 域主密钥备份

在Windows 2000中，系统可以存储本地主密钥备份，而不是CREDHIST标识符。这就是使DPAPI系统极其脆弱的原因。例如，Windows Password Recovery可以从其本地备份中解密主密钥，甚至不需要用户的密码就可以做到这一点。

主密钥文件的头包含以下信息(C++语法，后面是每个字段的描述)：

```
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.

标题后面是4个主键槽的序列。每个槽包含自己的标题和实际的键。

头结构对所有槽都是通用的。下面是版本1 (Windows 2000)的情况：

```
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;
```

标题插槽字段的描述:

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.

表2显示了各种操作系统的默认主密钥加密算法，以及它们对潜在暴力攻击的保护分析。

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**

表2. 主密钥加密算法。

4.3 生成新的主密钥

让我们仔细看看创建和加密用户的新主密钥的过程:

- 首先，我们调用API函数RtlGenRandom，它返回64个伪随机生成的字节。这是用于加密DPAPI blobs的主密钥的原始材料。
- 现在我们需要保护主密钥的原始材料。为此，我们使用用户的登录密码、安全标识符(SID)和额外的16个字节的随机数据(所谓的"盐")对其进行加密。主密钥的加密过程包括两个主要步骤。首先，我们使用基于密码的密钥衍生功能(PKCS #5中描述的加密标准)，从密码、SID和盐中获得密钥。然后，我们使用该密钥获得一个对称的主密钥加密密钥。不同版本的Windows以不同方式保护主密钥。因此，在Windows 2000中，默认的加密算法是RC4，在Windows XP和Win2K3中是3DES，而Windows Vista及以上版本则使用AES256。
- 主密钥获得一个唯一的名字，即GUID。每个DPAPI blob都存储这个唯一的标识符，它通过一个温暖的友好关系与之绑定。换句话说，主密钥GUID是密钥与DPAPI blob的"链接"。
- 主密钥，用用户的密码创建和加密，与其他系统数据一起存储在主密钥存储文件夹的一个单独文件中。MKSF是磁盘上存储主密钥的一个特殊位置。用户的主密钥存储在%APPDATA %/Microsoft/Protect/%SID%，其中%APPDATA%是应用程序数据目录。例如，C:/Users/John/AppData/Roaming/Microsoft/Protect/S-1-5-21-2893984454-3019278361-1452863341-

1003, %SID%是用户的安全标识。在上面的例子中，就是S-1-5-21-2893984454-3019278361-1452863341-1003。系统的主密钥存储在%WINDIR%/System32/Microsoft/Protect中，用于解密DPAPI blobs，由本地系统账户保护。所有在CryptProtectData函数中设置了CRYPTPROTECT_LOCAL_MACHINE标志的DPAPI blobs都是用系统的主密钥保护。

- 根据操作系统的标准，主密钥的创建、加密或解密可能非常耗时。例如，在Windows 7中，在现代PC上解密一个主密钥需要0.1秒以上。但DPAPI有内部缓存，所以一旦解密，主密钥就会被放在缓存中，再次访问它不涉及新的解密途径。

4.4 重新生成主钥匙

如果你打开磁盘上的万能钥匙存储文件夹，之前允许系统显示隐藏的文件夹和文件（万能钥匙文件具有隐藏属性），你可能会看到几个万能钥匙文件。为什么有几个？- 因为DPAPI出于安全考虑，大约每90天强制创建一个新的主密钥，所以任何用户账户通常都有几个主密钥；其数量取决于用户配置文件的创建日期。

主密钥的再生期在系统中是硬编码的。它被认为是不能被修改的。然而，它有一个 "BUT"。主密钥文件夹包含一个18字节的文件，名为Preferred，其中包含系统最后创建的主密钥的名称，以及创建日期。

首选文件结构

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

每当一些DPAPI函数被调用时，系统会检查Preferred中最后一个主密钥的创建日期，如果发现它超过了90天，就会创建一个新的主密钥，同时更新Preferred。主密钥的再生过程可以被手动管理。要做到这一点，在设置CRYPTPROTECT_CRED_SYNC标志的情况下调用CryptProtectData函数。系统将重新加密所有的主密钥，并在磁盘上更新它们。该操作通常是在修改用户密码后进行的。为了防止可选的主密钥更新，Windows 7引入了一个主密钥同步机制，我们将在下面介绍。

首选文件中的数据不受任何保护，因此通过手工修改最后一个主密钥的创建日期，人们可以手动管理再生过程；例如，无限延长最后一个主密钥的寿命或创建任意数量的主密钥。但是，虽然延长万能钥匙的寿命不会给坏人任何特权，但在最后一种情况下，大量的万能钥匙和密码的改变会使系统陷入冗长的停顿，因为每次用户的密码改变时，所有的万能钥匙都要进行强制的重新加密。

现在，算算在Windows 7中处理1000个主密钥需要多少时间，比如说，处理一个主密钥需要0.3秒以上。因此，在我们看来，主密钥的再生过程需要一些额外的工作。理想情况下，增加数据完整性控制--缺乏数据完整性控制似乎是DPAPI唯一的主要漏洞，甚至很难说是漏洞--就足够了。总的来说，整个DPAPI系统是很平衡和强大的。

4.5 备份主密钥并从备份中恢复它们

备份域内电脑的主密钥

当计算机是一个域的成员时，主密钥文件存储了用户的主密钥的备份。当创建主密钥时，DPAPI向存储RSA加密私人/公共密钥对的域控制器发送请求。主密钥使用域的公钥进行加密，并作为备份与用户的主密钥存储在同一个文件中。如果在解密主密钥的主副本时发生错误，DPAPI会将备份发送给域控制器。域控制器使用其私有的RSA密钥对其进行解密，并将解密后的主密钥发送回来。

在Windows XP - Windows 7下备份独立电脑的主密钥

当涉及到家庭用户的独立电脑（Windows XP和更高版本）时，DPAPI采用了一种基于密码重置盘的不同恢复机制。密码重置盘，用户可以在任何时候在控制面板中创建，允许用户恢复遗忘的密码。前面一句话中的关键短语是用户可以。数据恢复的重任完全落在了用户的肩上。

但是，如果用户在密码重置盘创建后改变了密码，会发生什么？一切都会好起来的。这个过程很简单，同时，也像村里的衣柜一样坚固。在创建密码重置盘时，系统会创建一对公共和2048位私人RSA加密密钥。用户的当前密码使用公钥进行加密，并存储在用户的注册表键HKEY_LOCAL_MACHINE/SECURITY/Recovery/%SID%（Windows XP - Windows Vista）或HKEY_LOCAL_MACHINE/C80ED86A-0D28-40dc-B379-BB594E14EA1B（Windows 7）。无论如何，这些密钥的访问是禁止的，包括系统的所有用户，包括管理员。而解密密码所需的私钥则保存在磁盘上。因此，如果在创建密码重置盘后，用户改变了密码，新的密码会用公共RSA密钥加密，并保存在注册表中，取代旧的密码。由于公共RSA密钥与存储在磁盘上的各自的私人密钥相匹配，私人密钥可以在以后任何时候用于解密刚刚创建并存储在注册表中的新密码。顺便说一句，最新的一个版本是 Windows Password Recovery 当密码重置盘可用时，可以解密用户的明文登录密码。

在Windows 2000的独立电脑上备份主密钥

但密码重置系统只出现在Windows XP中。在这之前是什么？事实上，尽管Windows 2000没有密码重置盘，但用户的主密钥备份仍然在独立的计算机上进行。主密钥和主密钥备份都存储在同一台电脑上。这意味着，任何用DPAPI加密的数据都可以被潜在的恶意者解密，而不需要知道用户的登录密码（只需对系统有物理访问权）。

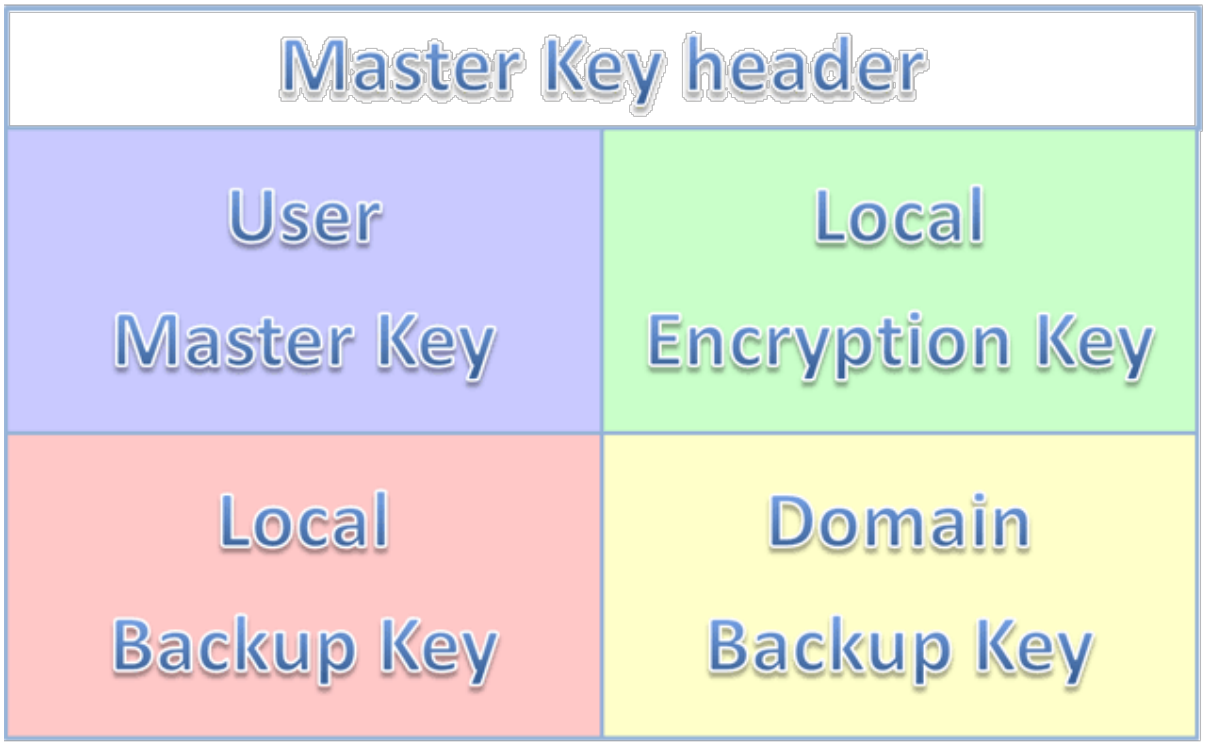

图 **5. Windows 2000中的主密钥文件结构。**

微软强烈建议将Windows 2000升级到一个更先进的版本。但是，如果你有充分的理由不这样做，我们将向你展示如何最好地保护你的好老Win2K免受潜在的损害。为此，让我们来看看本地主密钥的备份创建算法。一个Win2K主密钥文件通常由5个单元组成：

1. 标头和系统信息
2. 用户的主密钥
3. 本地备份加密密钥
4. 本地主密钥备份
5. 域主密钥备份

从本地备份中恢复用户的主密钥，首先要对本地备份的加密密钥进行解码。在解密过程中，我们使用本地计算机账户密码（哦，是的，它确实有这样一个账户），该密码存储在名为DPAPI_SYSTEM的LSA秘密中。

一旦获得本地备份的加密密钥，我们就用它来解密用户的主密钥的本地备份。这就像1-2-3一样简单。从这个角度看，Win2K对用户更友好，因为即使用户的密码被故意抹掉，DPAPI也可以随时从本地备份中恢复主密钥。希望没有必要解释这种用户友好性的代价。

任何能够访问操作系统Windows 2000下的个人电脑的用户，如果不是域成员，就可以很容易地解密其他用户的加密DPAPI数据，而不需要登录密码甚至密码散列

为了部分地防止这种暴行，我们必须改变SYSKEY的操作模式。运行SYSKEY工具（按键盘上的WIN+R快捷键，然后输入syskey.exe，再点击确定），替换SYSKEY的存储位置。现在，为了解密DPAPI_SYSTEM LSA的秘密，潜在的恶意者必须知道SYSKEY的启动密码或者拥有SYSKEY的启动盘。

5 DPAPI凭证历史

5.1 DPAPI凭证历史

DPAPI的正常运行需要保留所有用户以前的密码。所有以前的密码都以哈希值的形式存储在一个名为CREDHIST的特殊容器文件中，该文件位于%APPDATA%/Microsoft/Protect文件夹中。

5.2 CREDHIST架构

如果我们把凭证历史文件看成一条链，每个带有密码哈希的槽将作为链的一个环节出现，其二进制结构如下：

```
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;
```

数据字段说明：

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

图6显示了CREDHIST中使用的数据存储方案。

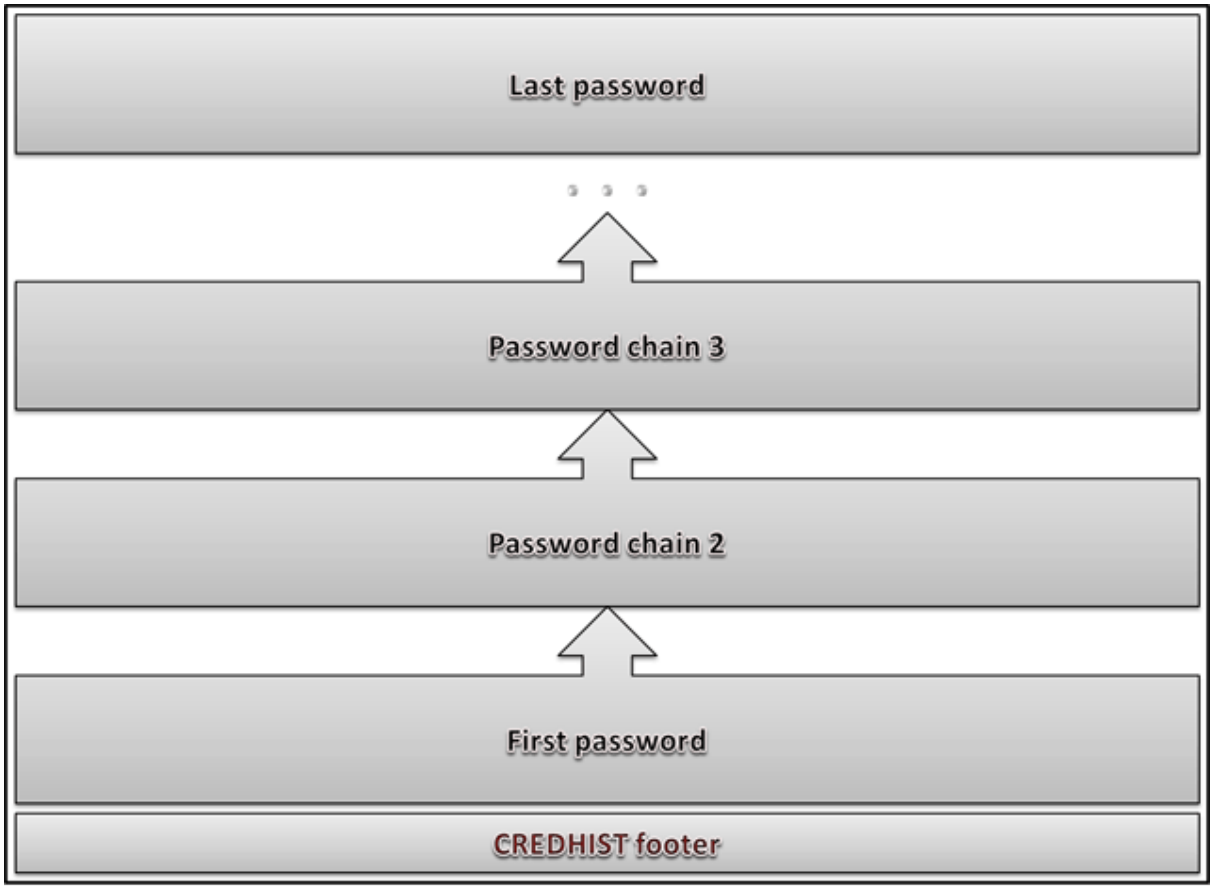

图 **6.CREDHIST**文件结构。

5.3 CREDHIST 加密

带有用户散列的插槽一个接一个地串联存储。每个密码散列用前面的散列加密，第一个散列用用户的当前密码加密。因此，要解密整个链，需要知道用户的当前密码（见图7）

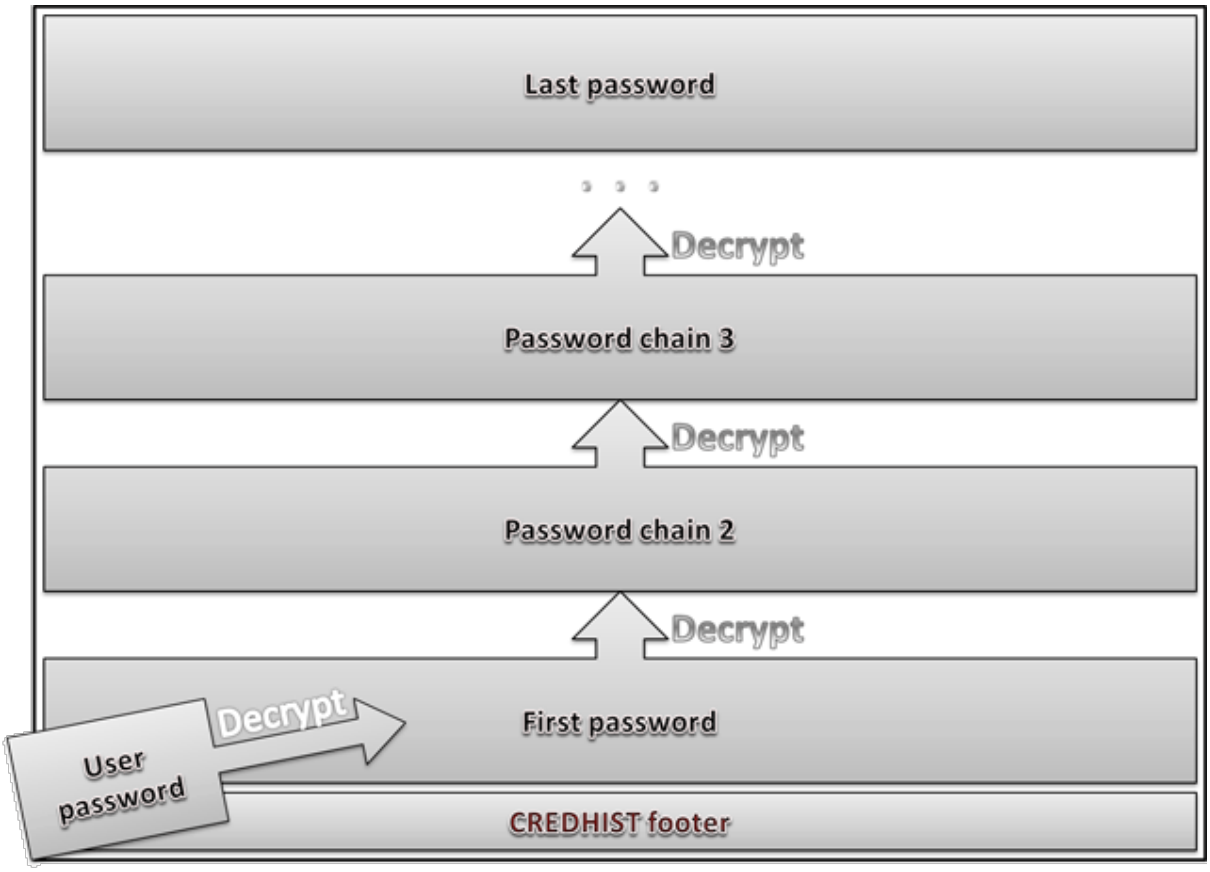

图 7.DPAPI中的凭证历史解密方案。

如果使用当前密码解密主密钥时发生错误，DPAPI将使用登录密码解密CREDHIST中的第一个哈希，并尝试使用第一个CredHi斯特哈希解密主密钥。如果主密钥仍然拒绝，则CREDHIST历史中的第一个散列将解密第二个散列，并重复该操作。

不同的操作系统使用不同的默认参数和算法来加密凭证历史记录（见下表3）。通常，加密算法类似于用于保护主密钥的加密算法。

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**

表 3.DPAPI中用于解码凭证历史记录的加密算法。

5.4 CREDHIST 漏洞

CREDHIST存储所有以前的密码哈希，即使密码相同。从理论上讲，用户有可能设置一个以前已经使用过的密码（我们称之为重复密码）。在这种情况下，要恢复当前密码，将凭证历史记录中的密码设置在当前密码和副本之间就足够了。

因此，将DPAPI与禁止输入重复密码的Windows安全策略一起使用更为方便，或者改进DPAPI，使其不允许在CREDHIST中存储重复密码。

6 改变Windows密码时会发生什么

6.1 密码更改和DPAPI

Windows中的密码更改通常分为几个关键阶段。首先，系统检查旧密码的有效性，然后检查新密码是否符合安全策略，并实际更改SAM或NTD中的密码。DIT数据库。在第二阶段，如果一切顺利，它将更新内部密码缓存。然后运行DPAPI密码更改例程。最后，它更新用户的生物特征登录凭证（Windows 7及更高版本）。

DPAPI系统足够智能，可以处理各种场景。让我们看看DPAPI内部在密码更改时发生了什么。以下是典型案例：

- 通过提供旧密码定期更改密码（Windows XP及更高版本）
- 密码重置或覆盖；e、 例如，由管理员，无需提供旧密码（Win XP和更高版本操作系统下的本地PC）
- 域用户密码重置（Windows 2000+）
- Windows 2000中本地PC用户密码重置

6.2 在Windows XP中定期更改密码 - Windows 7

Windows中的密码更改通常分为几个关键阶段。首先，系统检查旧密码的有效性，然后检查新密码是否符合安全策略，并实际更改SAM或NTD中的密码。DIT数据库。在第二阶段，如果一切顺利，它将更新内部密码缓存。然后运行DPAPI密码更改例程。最后，它更新用户的生物特征登录凭证（Windows 7及更高版本）。

DPAPI系统足够智能，可以处理各种场景。让我们看看DPAPI内部在密码更改时发生了什么。以下是典型案例：

通过提供旧密码定期更改密码（Windows XP及更高版本）

密码重置或覆盖;e、 例如，由管理员，无需提供旧密码（Win XP和更高版本操作系统下的本地PC）

域用户密码重置（Windows 2000+）

Windows 2000中本地PC用户密码重置

6.3 在Windows XP下的独立电脑上重置密码 - Windows 7

例如，管理员手动重置用户密码，或使用 密码重置程序.

尽管用户保持登录系统的能力，但使用DPAPI加密的所有数据不再可用（例如，网络密码、EFS密钥、邮件证书等），因为用户的旧密码未知，DPAPI无法解密主密钥。

如果在强制密码重置后再次设置旧密码，DPAPI将返回其原始状态，并且所有DPAPI Blob将再次可用。

这是因为DPAPI谨慎地保留主密钥的所有副本，即使它们无法解密。恢复对系统的完全访问（强制密码重置后）的另一种方法是使用先前创建的密码重置磁盘。

6.4 强制重设域用户密码

如果在Active Directory环境（Windows 2000+）中使用DPAPI，则主密钥存储两个备份副本。第一个副本受用户当前密码保护，第二个副本使用属于域控制器的公钥RSA加密。

在强制用户密码重置时，DPAPI无法解密主密钥的第一个副本。然而，将使用第二个备份副本恢复对所有DPAPI Blob的访问。工作站将主密钥的加密备份副本发送到域控制器。域控制器使用其专用RSA密钥解密备份副本，并将解密的主密钥发回。工作站使用新的用户密码对主密钥进行加密，并使用副本更新文件，并同步其他文件。

6.5 在Win2K的独立电脑上重置密码

在Win2K下运行的单机计算机上，不使用DPAPI密码重置磁盘。主密钥文件存储两个副本。第一个副本使用用户密码进行加密，第二个副本使用DPAPI_SYSTEM的LSA密钥，存储在注册表中，仅对系统帐户可用。

一旦用户密码被强制重置，DPAPI将无法解密主密钥的第一个副本，因此系统将自动解密第二个副本。然后用用户的新密码保护解密副本，并将其写回文件。所有其他主密钥也进行同步。

从最终用户的角度来看，访问受DPAPI保护的机密数据的过程是完全连续的。为确保DPAPI在Windows 2000下运行的单机PC上充分保护机密数据，建议修改系统密钥设置，以便系统需要输入系统密钥启动密码或在系统启动时提供系统密钥启动软盘。

7 从DPAPI blobs中恢复数据

7.1 从DPAPI blobs中恢复数据

使用DPAPI的一组工具 Windows密码恢复提供了广泛的机会，可以在离线模式下恢复使用DPAPI加密的密码和数据，即不加载用户配置文件。让我们回顾一下使用该软件的典型场景。

7.2 查找磁盘上的DPAPI blobs

在windows中，DPAPI blob可以存储在任何地方：二进制或文本文件、数据库、注册表、Active Directory等。在开始解密DPAPI Bloc之前，我们必须首先将其“拉出”并以适当的形式放入。为此，我们需要一个搜索工具来查找磁盘上的DPAPI Blos。

启动 该工具 并选择要搜索的源文件夹。. 假设源文件夹是C:/Program Data/Microsoft/Wlansvc。它包含Windows 7中的无线连接密码。
使用DPAPI加密的密码以文本格式存储在XML文件中；因此，我们分别设置搜索文本块（默认情况下启用该选项）。选择存储找到的文件的文件夹后，点击开始按钮。

找到的DPAPI Blob将以各自的名称存储在目标文件夹中。例如，如果程序在文件xxx中找到DPAPI blob。xml，它将被转换为二进制格式，并保存在目标文件夹中为xxx.xml.001。如果在文件yyy中。如果程序找到两个blob，它们将保存为yyy.xml。002和yyy.xml.003。

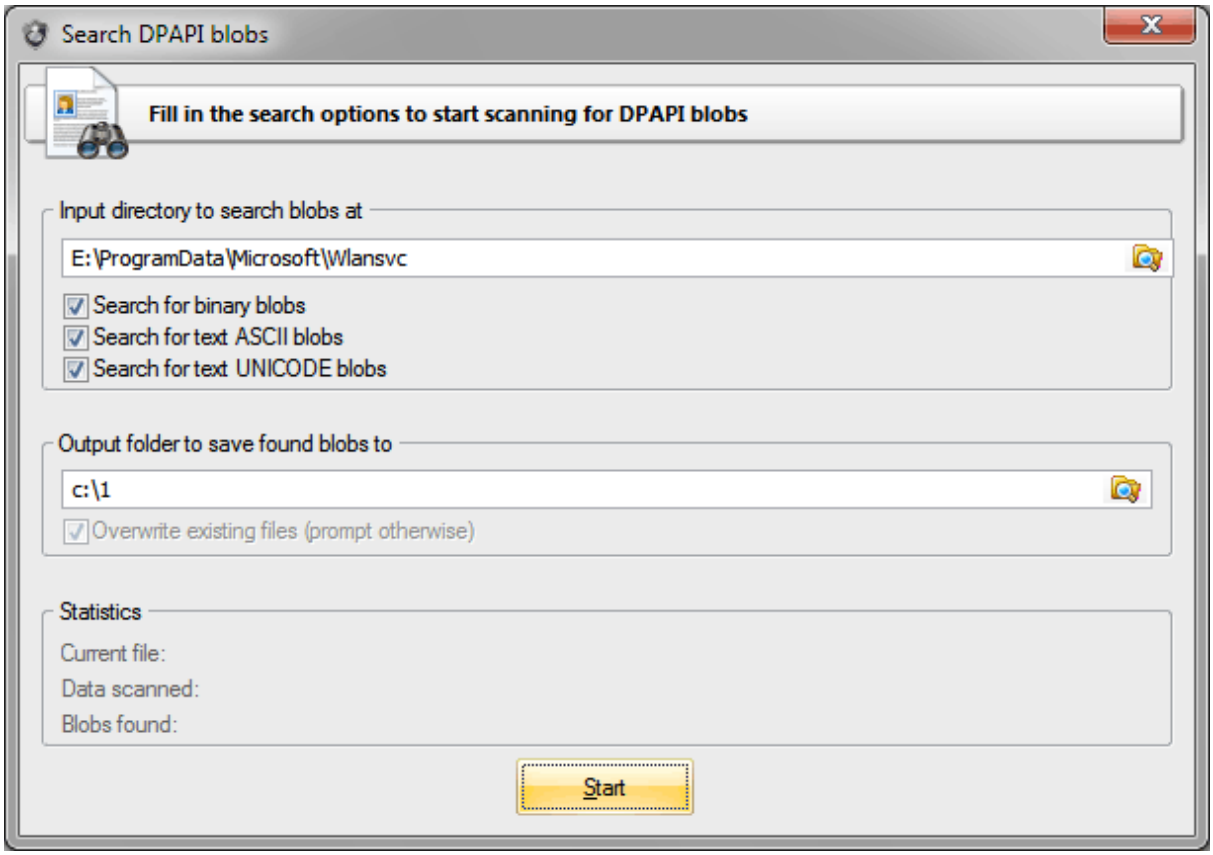

图 **8.** 寻找 **WiFi**密码。

7.3 在注册表和活动目录中寻找DPAPI blobs

在磁盘上寻找DPAPI blobs并不难。但如果你需要扫描当前用户的注册表或活动目录数据库呢？对这些文件的访问是被禁止的，包括管理员。

这很难，但幸运的是，并非不可能。为此，我们利用了一个方便的 注册表和活动目录备份工具，即使是新手也很容易操作。

启动它，然后选择我们要备份的内容和地点：注册表或活动目录。注意，解析活动目录数据库可能需要很长的时间。现在我们可以进入DPAPI blob搜索工具，并指定我们保存注册表或ntds.dit数据库的文件夹的路径。

7.4 恢复Windows 7中的无线连接密码

解密无线连接密码之前，要在上述的C:/Program Data/Microsoft/Wlansvc文件夹中查找相应的DPAPI blobs。所以，这里我们要跳过它的又一次描述。

启动DPAPI解密工具并指定提取的DPAPI blob的路径。无线连接密码受到CRYPTPROTECT_LOCAL_MACHINE标志的保护，所以要恢复它们，我们需要E:/Windows/System32/Microsoft/Protect/S-1-5-18/User文件夹中的系统主密钥。

在该文件夹中，找到带有主密钥的所需文件。默认情况下，打开文件对话框只显示我们需要的文件。在我们的例子中，CREDHIST的路径是没有必要的，所以我们把这个字段留空。

一旦移动到应用程序的第三步，我们标记出我们不需要用户的密码。相反，我们需要两个注册表文件，SYSTEM和SECURITY。好吧，我们也指定它们。注意，你必须首先解锁当前的注册表文件（即备份它们）。在用户的SID字段中，程序会自动放置系统的SID: S-1-5-18。离开该字段，通过获取Windows 7中的原始明文无线连接密码完成解密。

容易吗？让我们更进一步。

7.5 恢复存储在Internet Explorer中的Facebook密码

现在的问题有点复杂了。让我们尝试在Internet Explorer中恢复Facebook账户的密码。要做到这一点，我们可能要做相当多的"手鼓舞"，所以提前储备好耐心。

Internet Explorer的自动完成密码存储在注册表的HKCU/Software/Microsoft/Internet Explorer/IntelliForms/Storage2中。运行Regedit并打开该目录。在那里，你可能会看到一些二进制记录。我们只需要其中一条，名为F6FFE33B9EF4D7CB8F5A2F41F3222D21E131ED787A。

如果你没有这样的记录，可以尝试在Facebook上创建一个测试密码：打开页面http://www.facebook.com/（www是必须的），输入任何电子邮件地址和测试密码，然后点击登录按钮。当IE友好地提出保存新密码时，就这样做。当然，Facebook网站会返回一个错误，但这并不重要，密码已经被保存了。

图 **9.** 创建 **Facebook**测试密码。

切换回regedit，按F5更新数据。存储IE密码的注册表键必须得到一个新的记录，名为F6FFE33B9EF4D7CB8F5A2F41F3222D21E131ED787A。这个狡猾的名字不过是网站名称的SHA哈希值。

当你打开它时，你会看到一个DPAPI blob，里面有该网站的加密登录和密码。让我们把DPAPI blob导出到一个文本文件。要做到这一点，打开cmd并运行以下命令。

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

引号是必须的！我们的密码将以ASCII格式保存到c:test文件夹。你可以自由地设置任何其他文件夹；这其实并不重要。现在，我们需要做的是，从ASCII文件fb.txt中提取我们需要的二进制DPAPI blob。

启动搜索工具，指定源文件夹c:test，设置搜索文本文件选项，在输出端得到二进制文件fb.txt.001。最后，我们可以继续进行实际的密码恢复。

打开相应的工具，将路径设为fb.txt.001。第二步，指定主文件，该blob是用该文件加密的。所有用户的主密钥都存储在%AppData%/Roaming/Microsoft/Protect/%SID%文件夹中。再往上一层，是CREDHIST文件。我们可能也能使用它。

当我们进入应用程序向导的第三步时，最有趣的东西开始了。为了解密Facebook的密码，我们将需要正确设置以下三个参数：用户的SID、他的登录密码和DPAPI blob加密的秘密。该程序通常会从主密钥的路径中自动提取用户的SID。如果没有这样做，你将不得不通过手工获得该参数（例如，通过分析CREDHIST文件）。关于用户的登录密码，一切也都很清楚。但是我们应该如何处理这个密钥呢？

对于这个密钥，在加密密码时，IE使用密码输入的网页名称。该网页必须以小写字母输入，作为UNICODE，并以0结尾。如果你手头有一个HEX编辑器，并且有足够的技能来适应它的工作，可以尝试手动创建该秘密。否则，只需在 这里下载。

因此，在熵设置中指定了包含秘密的文件的路径后，我们终于完成了恢复Facebook密码的史诗。解密后，你会看到密码、登录和其他系统信息被合并成一个数据结构（图10）。不要让这个问题困扰你。

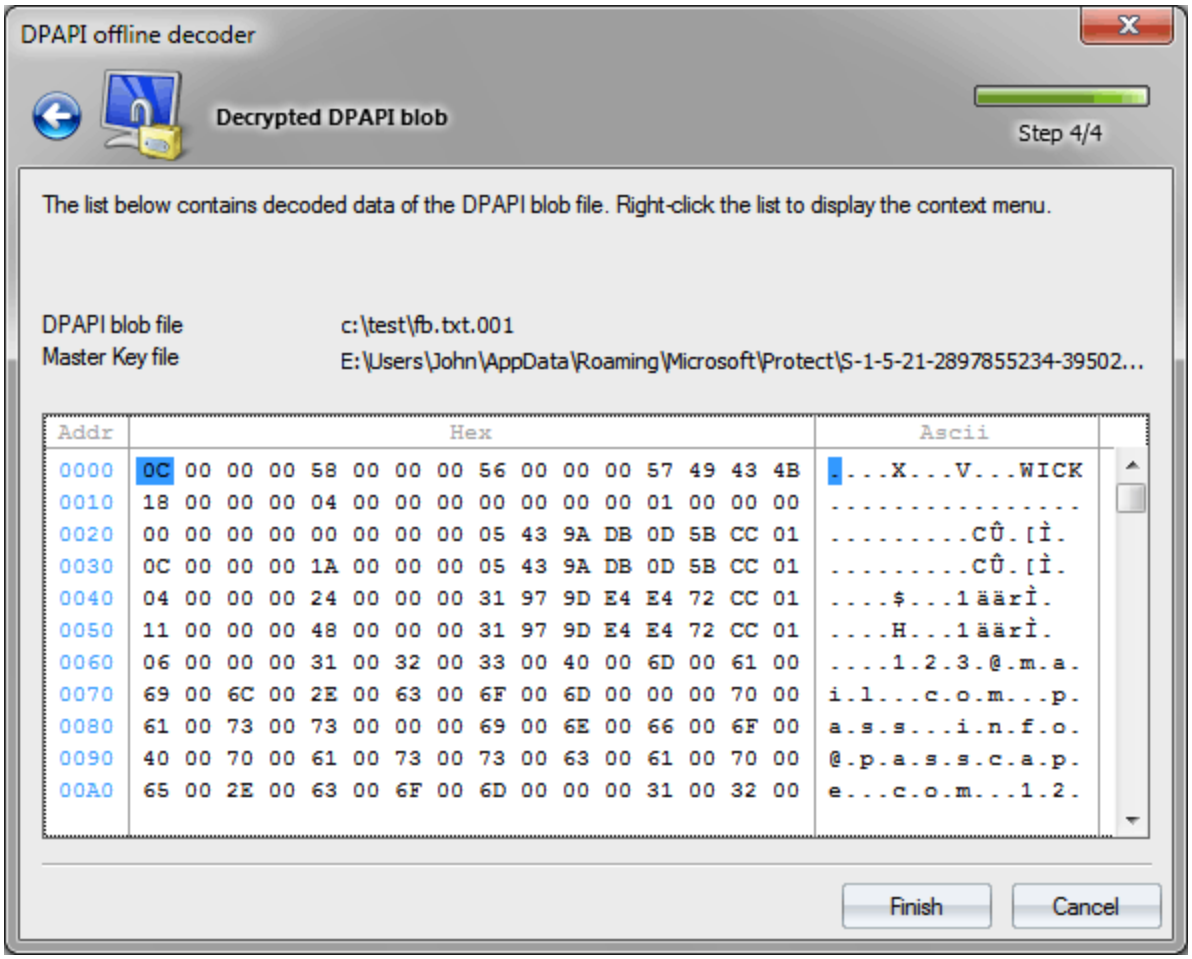

图 **10.** 解密的**Facebook**密码。

7.6 无需从SAM/NTDS.DIT加载哈希值即可恢复用户密码

有史以来第一次，Windows Password Recovery允许恢复登录密码，而不需要从SAM或NTDS.DIT加载用户的哈希值。这之所以成为可能，是因为用户的密码参与了主密钥的加密过程。因此，主密钥可以被用来验证密码。

为了说明上述情况，运行 主密钥分析工具并 并指定一个带有主密钥的文件的路径。在成功解析主密钥后，该程序将显示其内部结构。右键单击它，然后选择使用字典验证密码项目，并指定任何字典的路径。

尽管它具有革命性的特点，但从万能钥匙中恢复用户的密码并没有任何实际好处。例如，在Windows 7中，密码验证的速度大约为10 p/s。理论上说，通过优化加密算法和让GPU参与验证，速度可以提高几个数量级。但即使在那之后，与NTLM哈希相比，万能钥匙的密码挖掘速度似乎是可笑的。在安全性方面，DPAPI已经准备就绪

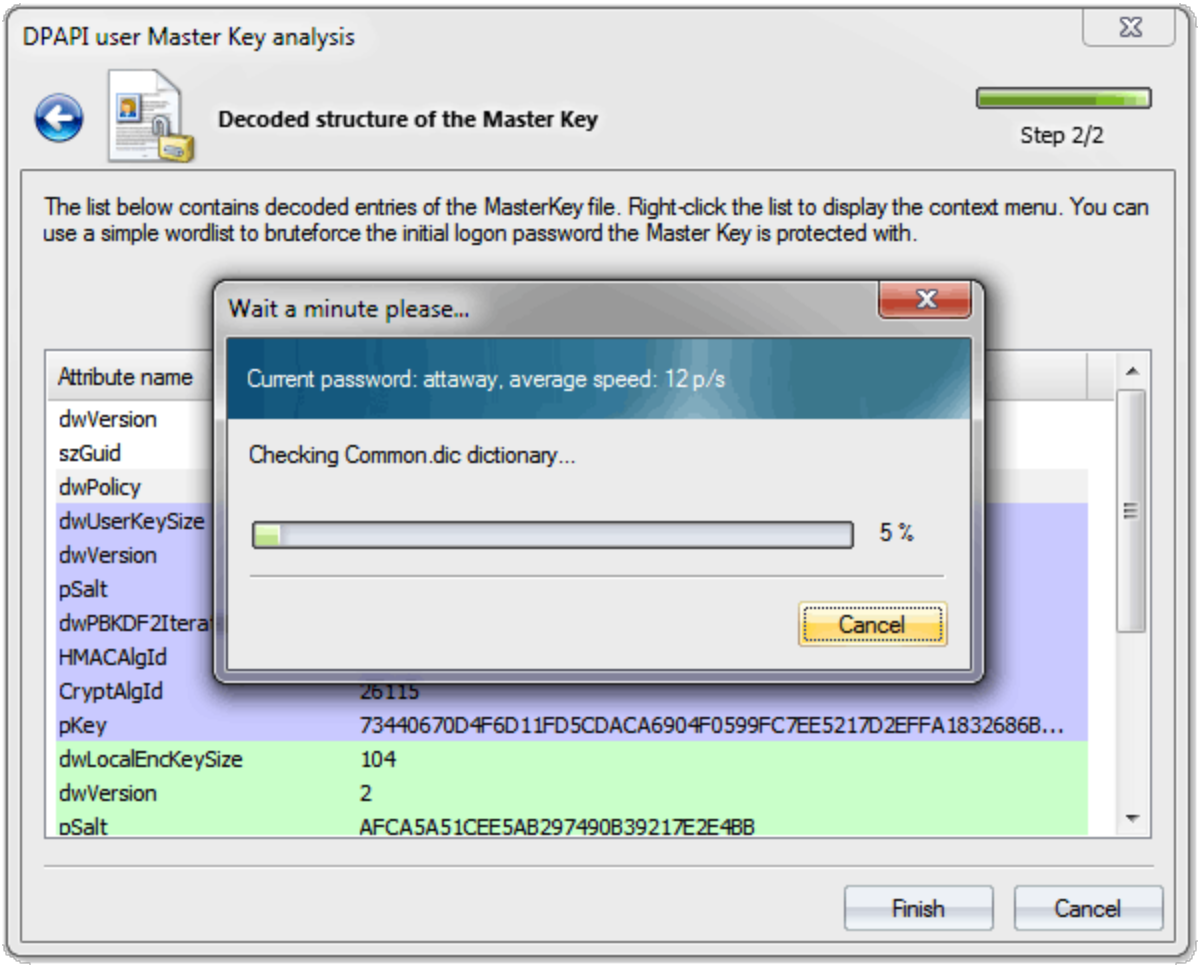

图 **11.** 在 **Windows Vista**中通过分析 **DPAPI**主密钥恢复用户的登录密码。

7.7 从DPAPI中解密凭证历史哈希值

打开凭证历史转储工具，指定CREDHIST文件的路径，该文件存储在%AppData %/Roaming/Microsoft/Protect文件夹中。如果该文件存储了以前的用户密码，"CREDHIST密钥链"字段应该有一个非零值。

以前的用户密码是以SHA1和NTLM哈希值的形式存储。该应用程序有一个相应的开关。如果你选择当前用户的CREDHIST文件，就不需要输入登录密码，程序会自动从Windows缓存中获取密码，只需进行相应的设置。否则，你将不得不手动输入当前用户的密码。

该工具支持部分转储。这意味着，在当前用户密码未知，但其中一个旧用户密码已知的情况下，该程序可以解密早期使用的哈希值，即在输入旧密码之前。

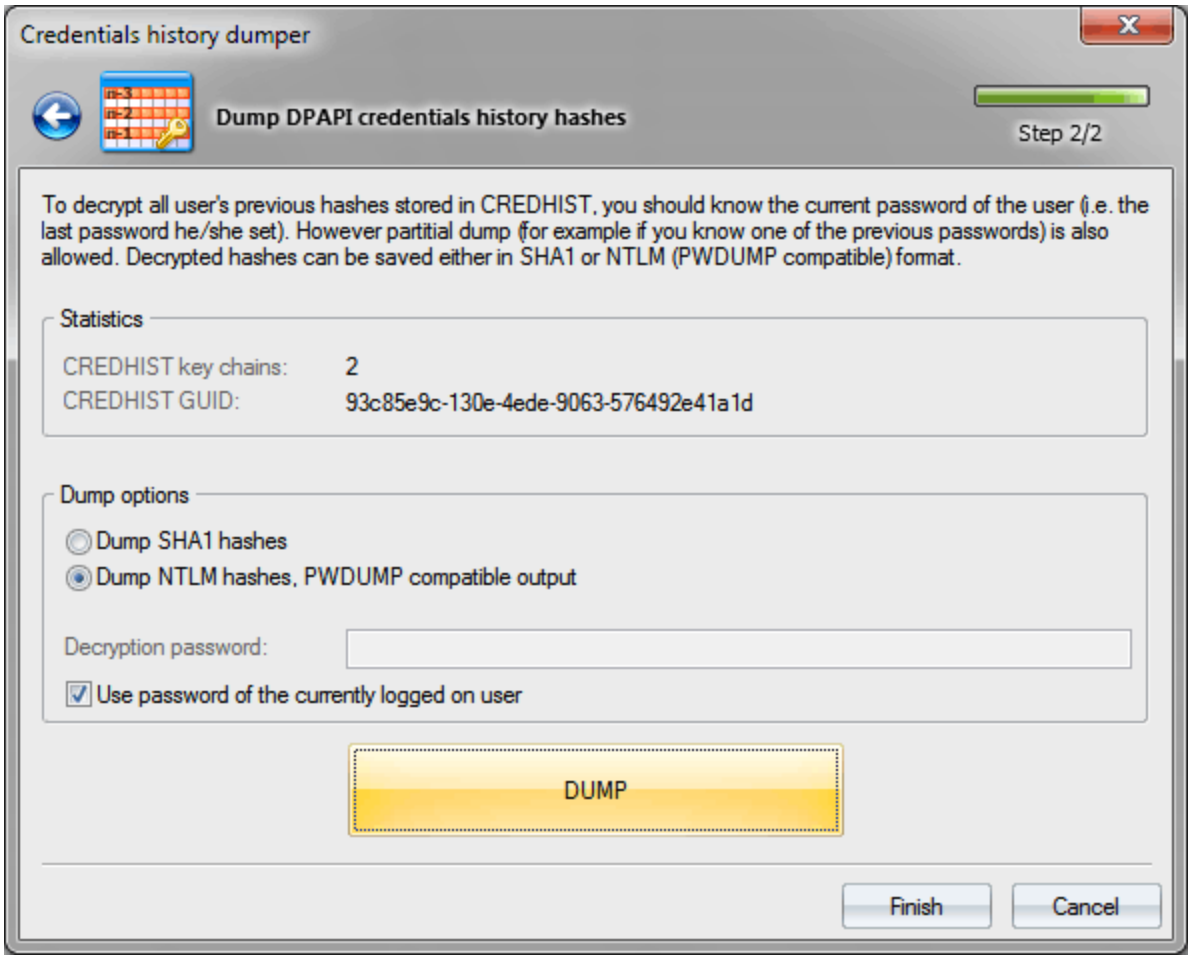

图 **12.** 转储DPAPI凭证历史密码。

8 总结

DPAPI值得如此密切关注，至少是因为它是唯一的基于密码的系统，为用户的个人数据提供了适当的和彻底的保护。没有一个操作系统有比DPAPI更可行的替代方案！

我们也许应该提到，DPAPI的第一个实现有许多严重的缺陷，这可能使潜在的恶意者轻易地破坏由DPAPI保护的用户数据。

众所周知，第一张饼总是疙疙瘩瘩的。在所有的续集操作系统中，从Windows XP开始，这些漏洞不仅仅是被消除了，整个DPAPI系统都经历了一次重大的修订。特别是，它采用了新的加密算法；这使得主密钥的密码查询速度慢了大约1000(！)倍。有可能允许任何用户访问任何由EFS加密的文件的主密钥加密错误已被修复。本地主密钥备份系统已经被密码重置盘等所取代。

总的来说，DPAPI加密系统已经变得更加强大、有力，满足了对密码安全的严格要求。