# Секреты DPAPI

# анализ безопасности и восстановление паролей DPAPI


Passcape Software

# 1 Введение

В этом документе мы сделали попытку анализа работы DPAPI, произвести разбор недокументированных структур и алгоритмов шифрования DPAPI, понять и описать внутреннее функционирование этой системы.

Впервые в мире дано полное описание (не претендующее, однако, на универсальность) логики работы DPAPI и всех недокументированных структур, включая объекты DPAPI, Мастер Ключи и файлы истории паролей.

Мы также попытались указать некоторые недостатки в функционировании последней версии DPAPI и возможные методы их устранения. Кроме того, более глубокий анализ реализации первой версии DPAPI, выпущенной вместе с Windows 2000, показал наличие нескольких серьезных уязвимостей и поставил под сомнение всю безопасность этой системы.

Большое внимание уделено восстановлению данных DPAPI без возможности загрузки профиля пользователя. Впервые на практике применен алгоритм восстановления пароля на вход пользователя без доступа к SAM или NTDS.DIT.

Анализ работы DPAPI был произведен с помощью набора из 6 утилит, интегрированных в одно из самых мощных приложений для аудита паролей Windows: Windows Password Recovery. Мы надеемся, что данные инструменты, а также некоторые материалы нашей статьи, будут интересны не только экспертам и криминалистам, но и всем исследователям в области безопасности.

# 2 Архитектура и безопасность DPAPI

## 2.1 Что такое DPAPI

Начиная с Windows 2000, компания Microsoft стала поставлять в своих операционных системах специальный интерфейс для защиты данных, который называется Data Protection Application Programming Interface, или сокращенно **DPAPI**. В настоящее время DPAPI получил очень широкое распространение и применяется во многих приложениях и подсистемах Windows. Например, в системе шифрования файлов, для хранения беспроводных паролей сети, в Windows Credential Manager, Internet Explorer, Outlook, Skype, Windows CardSpace, Windows Vault, Google Chrome и т.д. DPAPI стала популярной у программистов в первую очередь благодаря простоте использования, поскольку состоит всего лишь из пары функций для шифрования и расшифровки данных: **CryptProtectData** и **CryptUnprotectData**.

Несмотря на свою кажущуюся простоту, техническая реализация DPAPI довольно сложна, а логика работы этих функций во многом напоминает веселый детский стишок «Вот дом, который построил Джек». Возможно по этой причине, внутренние структуры и принципы работы DPAPI долгое время находились за закрытым занавесом.

Впервые анализ системы DPAPI был произведен фирмой Passcape Software в 2003 году. На основе этого анализа, в 2005 году вышло первое коммерческое приложение (Outlook Password Recovery), умеющее расшифровывать объекты DPAPI в режиме офлайн, т.е. без обязательного входа в учетную запись владельца. Алгоритм расшифровки, предложенный Passcape, просто имитировал работу DPAPI, поэтому система DPAPI не была скомпрометирована.

С выходом новой версии программы Windows Password Recovery, алгоритм офлайн расшифровки DPAPI получил новый виток. Теперь стало возможным не только расшифровывать DPAPI объекты, но и производить их анализ и поиск на диске, расшифровывать Мастер Ключи и проверять пароли к ним, вытаскивать хэши истории паролей пользователей из DPAPI и многое другое.

## 2.2 Что защищает DPAPI

DPAPI используется при защите следующих персональных данных:

- Пароли и данные автозаполнения форм в Internet Explorer, Google Chrome
- Пароли учетных записей почты в Outlook, Windows Mail, Windows Mail, и т.д.
- Пароли учетных записей встроенного менеджера FTP
- Пароли доступа к общим папкам и ресурсам
- Пароли и ключи учетных записей беспроводной сети
- Ключи шифрования в Windows CardSpace и Windows Vault
- Пароли соединений удаленного доступа к рабочему столу, .NET Passport
- Приватные ключи Системы Шифрования Файлов (EFS), шифрования почты S-MIME, другие сертификаты пользователя, SSL/TLS в Internet Information Services
- EAP/TLS и 802.1x (VPN и WiFi аутентификация)
- Сетевые пароли в Credential Manager
- Персональные данные любого приложения, программно защищенные при помощи API функции CryptProtectData. Например, в Skype, Windows Rights Management Services, Windows Media, MSN messenger, Google Talk и др.

Пример удачного и хитроумного способа защиты данных с использованием DPAPI, являет собой реализация алгоритма шифрования паролей автозаполения в Internet Explorer. Для шифрования пароля и логина какой-нибудь веб страницы, вызывается функция CryptProtectData, в параметре опциональной энтропии которой указывается адрес этой страницы. Таким образом, без знания исходного URL адреса, где был введен пароль, никто, даже сам браузер Internet Explorer, не сможет выполнить обратную расшифровку.

## 2.3 Архитектура DPAPI

Начиная с Windows 2000, как уже было сказано выше, любое приложение может защитить свои персональные данные (например, пароли) путем элементарного вызова функции CryptProtectData, возвращающую «непрозрачную» двоичную структуру, иначе именующуюся объектом DPAPI (DPAPI blob). «Непрозрачную», по определению Microsoft, в том смысле, что там содержатся не только исходные зашифрованные данные приложения, но и другая недокументированная служебная информация, необходимая для их расшифровки.

Аналогично функционирует API функция CryptUnprotectData, которая, как видно из ее названия, работает в обратную сторону: получая на входе зашифрованный DPAPI объект, возвращает приложению исходные декодированные данные.

На рисунке 1 схематично представлена работа DPAPI.

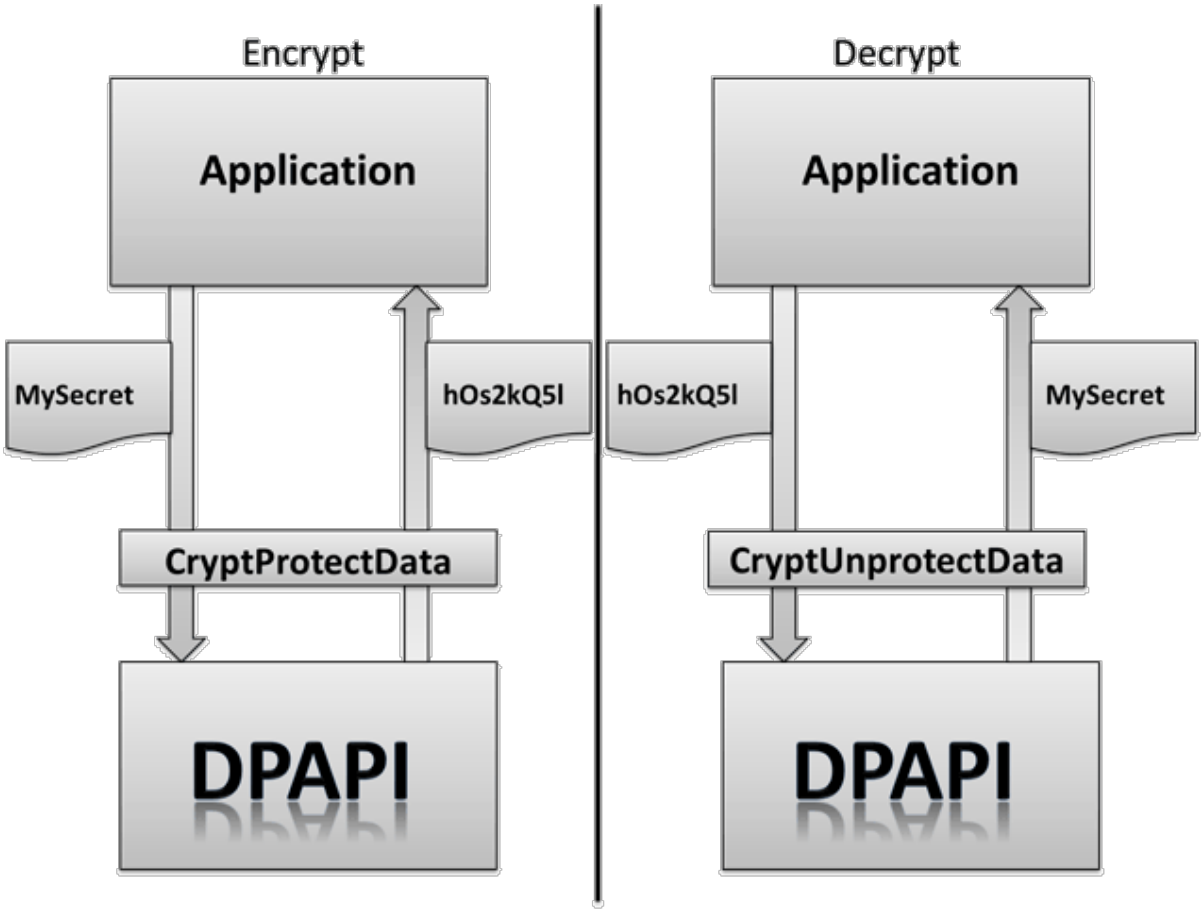


**Рис 1. Схема работы DPAPI**

Шифрование DPAPI базируется на пароле пользователя, поэтому данные, защищенные под одной учетной записью, нельзя расшифровать под другой. Более того, DPAPI предоставляет возможность ограничивать доступ к данным даже внутри одной учетной записи путем установки дополнительного секрета (энтропии). Таким образом, без знания дополнительного секрета, одно приложение не сможет получить доступ к данным, защищенным в другой программе.

Любому программисту, реализующему интерфейс DPAPI, важно понять, что эта система осуществляет только шифрование данных. Приложение должно само обеспечить механизм хранения возвращаемых объектов DPAPI, в том числе надежного сокрытия опциональной энтропии, если она используется.

## 2.4 Анализ безопасности DPAPI

DPAPI была создана с учетом многих аспектов с точки зрения безопасности. Вот список основных особенностей, отличающих ее от подобных систем:

- DPAPI работает поверх CryptoAPI. Можно даже написать свой собственный поставщик услуг шифрования и использовать его в DPAPI.

- DPAPI использует проверенные криптографические алгоритмы. Например, в Windows 7 по умолчанию используется алгоритм шифрования **AES256** в режиме CBC, для хэширования – **SHA512**, в качестве функции для генерации ключей шифрования – алгоритм **PBKDF2** PKCS #5.
- Все алгоритмы, а также длина их ключей могут настраиваться из реестра, доступ к которому имеют только пользователи с привилегий Администратора. Например, количество итераций в функции PBKDF2 задается в следующем параметре реестра:
  **HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Protect\Providers\%GUID%**
  **MasterKeyIterationCount**
- Все параметры, настраеваемые из реестра, имеют свои ограничения и не могут быть меньше значений, установленных по умолчанию. Например, в Windows XP для MasterKeyIterationCount значение по умолчанию равно 4000.
- Все критические данные защищены проверкой целостности.
- Размер всех ключей шифрования во всех алгоритмах достаточен, чтобы исключить возможную атаку перебором.
- Для этой же цели в PBKDF2 задан довольно большой счетчик итераций. Исключение составляет первая реализация DPAPI. Из таблицы 1 видно, что скорость перебора паролей Мастер Ключей в Windows 2000 достаточная для того, чтобы скомпрометировать пароль пользователя.
- В целях безопасности, срок действия Мастер Ключа ограничен. Хотя реализация этого механизма имеет слабые места, о чем будет сказано ниже.
- Все коммуникации с контроллером домена осуществляются по защищенному каналу RPC.
- Кэшированные данные не записываются ни на диск, ни в swappable RAM. Промежуточные ключи и значения обнуляются в стеке, после отработки каждой функции шифрования. Кстати, при анализе утилиты SYSKEY, специалисты Passcape обнаружили интересную ошибку, связанную именно с некорректным обнулением стека, в результате которой в некоторых системах на базе Windows 2000, Windows XP, Windows 2003, часть системного ключа шифрования (SYSKEY) лежит в открытом виде в реестре и доступна всем пользователям системы. Увы, бывает и такое.

Однако есть некоторые нюансы, незнание которых может ослабить систему защиты DPAPI:

- **CRYPTPROTECT_LOCAL_MACHINE** - флаг, который задается в функции CryptProtectData, довольно удобный при некоторых обстоятельствах, в действительности может скомпрометировать данные пользователя, не обеспечивая должной защиты. В случае его установки, шифрование не будет зависеть от пароля на вход пользователя. С другой стороны, такое поведение не противоречит концепции DPAPI и связано с особенностями реализации интерфейса.
- DPAPI может быть настроена на работу в режиме совместимости с Windows 2000, когда резервный Мастер Ключ хранится локально и может быть расшифрован при помощи LSA секрета. То есть теоретически, все объекты DPAPI можно расшифровать без знания пароля владельца (смотрите рисунок 2). Тем не менее, для активации этого режима необходим доступ Администратора к ключу реестра с конфигурацией DPAPI.

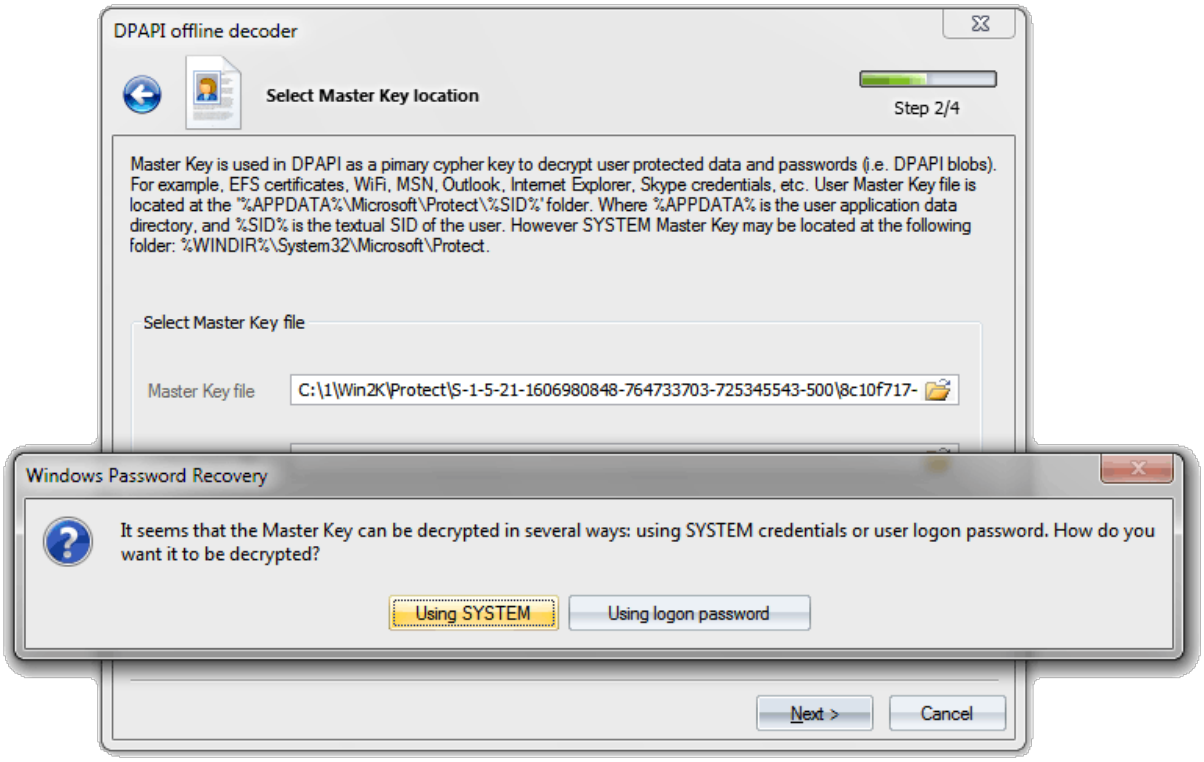


**Рис 2. Расшифровка DPAPI объекта без знания пароля пользователя в Windows 2000.**

| ОС | Алгоритм шифрования | Алгоритм хэширования | PKCS#5 PBKDF2 rounds | Скорость подбора (п/с) |
|---|---|---|---|---|
| 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** |

**Таблица 1. Алгоритмы шифрования, используемые в DPAPI**

Если же брать в целом, DPAPI обеспечивает, пожалуй, самую сильную на сегодняшний день защиту и безопасность данных пользователя.

## 2.5 Конфигурация DPAPI

DPAPI работает поверх Crypto API и может использовать любого поставщика услуг шифрования. Выбор крипто провайдера осуществляется в соответствии со следующим ключом реестра:

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

По умолчанию, используется крипто провайдер Microsoft с именем **df9d8cd0-1501-11d1-8c7a-00c04fc297eb**, реализованный в библиотеке psbase.dll.
Настройка шифрования осуществляется изменением соответствующего значения в ключе реестра
**HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Protect\Providers\%GUID%**

Где %GUID% является уникальным идентификатором поставщика услуг. По умолчанию ключ с настройками DPAPI соответствует
**HKEY_LOCAL_MACHINE \Software\Microsoft\Cryptography\Protect\Providers\df9d8cd0-1501-11d1-8c7a-00c04fc297eb**

Доступ для изменения этой ветки реестра имеют только пользователи с правами Администратора. Ниже даны описания некоторых значений этого ключа.

**MasterKeyIterationCount**
Счетчик итераций в функции PBKDF2, не может быть меньше, чем значение, установленное по-умолчанию. Алгоритм генерации ключа используется в шифровании Мастер Ключа и истории паролей пользователя.

**MasterKeyLegacyCompliance**
Установка этого значения в 1 включает режим совместимости с Windows 2000 во время резервирования и восстановления Мастер Ключей.

**MasterKeyLegacyNt4Domain**
Поскольку резервирование и восстановление DPAPI не поддерживается в Windows NT4, установите параметр MasterKeyLegacyNt4Domain в 1, чтобы использовать режим совместимости с Windows 2000 и избежания потери данных во время резервирования/восстановления.

**DistributeBackupKey**
Ключ, отвечающий за совместимость при резервировании и восстановлении ключа домена. Например, если уровень функционирования домена установлен как Windows 2000.

**Recovery Version**
Параметр, используемый для совместимости при создании резервных копий и восстановлении локальных копий Мастер Ключей. По умолчанию используется версия 2, Windows 7 и Windows Server 2008 R2 могут быть настроены для работы в версии 3, путем установки этого DWORD значения реестра в 3.

**ProtectionPolicy**
Флаг политики безопасности, используемый по умолчанию. Соответствует флагу **dwPolicy** в заголовке Мастер Ключа.

**Encr Alg - Encr Alg Key Size**
Пара значений, манипулирующих типом алгоритма шифрования, который применяется в DPAPI, и размером ключа этого алгоритма.

**MAC Alg - MAC Alg Key Size**
Ключи для установки алгоритма контроля целостности данных в DPAPI.

# 3 Шифрование данных в DPAPI

## 3.1 Описание функций CryptProtectData и CryptUnprotectData

Параметры этих функций практически одинаковы, поэтому рассмотрим только CryptProtectData, декларация которой для C++ выглядит следующим образом:

```
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
);
```

Описание полей функции:

| | |
|---|---|
| DATA_BLOB *pDataIn | - указатель на структуру DATA_BLOB, ссылающийся на исходные данные, которые необходимо зашифровать. |
| LPCWSTR szDataDescr | - описатель данных, хранится в открытом виде в DPAPI объекте. Этот опциональный параметр носит информационный характер, его можно не задавать при вызове функции. |
| DATA_BLOB *pOptionalEntropy | - аналогично предыдущему, является опциональным параметром. Однако в отличие от описателя, энтропия напрямую влияет на безопасность данных, поэтому не хранится в объекте DPAPI. Пользователь сам должен обеспечить надлежащее управление этим секретом. Некоторые сторонние приложения (например, GTalk) хранят этот секрет в реестре, другие компоненты Windows (например, .Net Passport или WININET) в качестве секрета используют жестко прописанную константу, третьи используют переменное значение. Например, встроенный FTP клиент Windows Vista создает энтропию на основе имени сервера. Так или иначе, потенциального взломщика это может, в крайнем случае, позабавить. На наш взгляд, самый удачный пример использования энтропии реализован в механизме хранения паролей Internet Explorer, при котором секрет вообще нигде не хранится. |
| PVOID pvReserved | - в настоящий момент не используется. |
| CRYPTPROTECT_PROMPTSTRUCT *pPromptStruct | - структура, задав которую, необходимо будет ввести дополнительный пароль для шифрования объекта DPAPI. Скажем так, используется очень нечасто ввиду того, что запрос пароля происходит в интерактивном режиме. |
| DWORD dwFlags | - флаги, управляющие процессом шифрования. Вот описание наиболее интересных из них: |

| | | |
|---|---|---|
| | CRYPTPROTECT_UI_FORBIDDEN | 0x1 | Используется, когда интерфейс пользователя недоступен. Например, при удаленном доступе. |
| | CRYPTPROTECT_LOCAL_MACHINE | 0x4 | Защита данных происходит с использованием учетной записи локального компьютера. Любой пользователь системы сможет расшифровать их. |

| | | |
|---|---|---|
| CRYPTPROTECT_CRED_SYNC | 0x8 | Позволяет принудительно выполнить синхронизацию паролей. Обычно эта операция автоматически выполняется после смены пароля пользователя. |
| CRYPTPROTECT_AUDIT | 0x10 | При шифровании/расшифровки будет происходить аудит. |
| CRYPTPROTECT_VERIFY_PROTECTION | 0x40 | Этот флаг проверяет защиту объекта DPAPI. Если уровень защиты, установленный по умолчанию, выше уровня защиты для данного объекта, функция вернет ошибку CRYPT_I_NEW_PROTECTION_REQUIRED как совет того, что желательно по-новому выполнить защиту исходных данных. |
| CRYPTPROTECT_CRED_REGENERATE | 0x80 | Регенерация паролей локальной машины. |
| CRYPTPROTECT_SYSTEM | x20000000 | Индикатор того, что только системные процессы смогут выполнить шифрование/расшифровку данных. |

DATA_BLOB *pDataOut - объект DPAPI, который получается в результате вызова функции

## 3.2 Пример использования CryptProtectData/ CryptUnprotectData

Для удобства проведения тестов, а также в качестве примера, мы написали две небольшие одноименные утилиты, реализующие простые обертки вокруг функций DPAPI. Таким образом, к функциям DPAPI можно обращаться из командной строки. Ниже приведен исходный код C++ программы CryptProtectData.

Исходные коды: CryptProtectData , CryptUnprotectData.
Готовые программы: CryptProtectData , CryptUnprotectData.

**Исходный код программы CryptProtectData**

```
// 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 Шифрование данных в DPAPI

Шифрование персональных данных пользователя в DPAPI происходит в три этапа:

1. Первоначально приложение вызывает функцию CryptProtectData, указывая исходный материал для защиты и необязательный параметр энтропии. Если последний не задан, любая другая программа в контексте текущего пользователя может выполнить обратную расшифровку, получив физический доступ к объекту DPAPI.
2. Функция CryptProtectData относится к семейству CryptoAPI и находится в Crypt32.dll, откуда запрос на дальнейшую обработку через защищенный RPC канал перенаправляется в системный процесс Local Security Authority (LSA), и переходит смена контекста текущего пользователя на системный. Все дальнейшие манипуляции производятся в контексте локальной системы.
3. В LSA происходит сам процесс шифрования данных и отправка их обратно по каналу RPC в Crypt32.dll, откуда они благополучно возвращаются в исходное приложение.

CryptProtectData и CryptUnprotectData по сути являются простыми функциями-обертками, т.к. реальное шифрование данных протекает в контексте системы. Вся магия DPAPI происходит в LSA и скрыта от глаз сторонних наблюдателей.

Процесс шифрования долгий и скучный, напоминает супружеский долг после 20 лет совместного проживания. Вначале из пароля пользователя при помощи криптографического стандарта PBKDB2 получается ключ шифрования Мастер Ключа. Этот ключ шифрования используется для расшифровки Мастер Ключа. Если расшифровка прошла неудачно, пароль пользователя расшифровывает первый хэш истории паролей CREDHIST, из которого в свою очередь опять получается ключ шифрования Мастер Ключа и делается очередная попытка расшифровки. Если и она потерпела неудачу, расшифровывается следующий хэш истории паролей. Операция повторяется до тех пор, пока Мастер Ключ не будет расшифрован, либо не закончится вся цепочка хэшей истории.

Чтобы не спрашивать пароль каждый раз при вызове CryptProtectData / CryptUnprotectData, Windows кэширует его, храня в недрах LSA на протяжении всей жизни профиля пользователя. Вот

почему DPAPI работает исключительно в режиме онлайн, только после входа пользователя в систему.

В 2005 году Passcape Software впервые представила работающий в своих продуктах алгоритм, имитирующий офлайн работу DPAPI. Иными словами, для расшифровки DPAPI объектов не нужна загрузка контекста пользователя, хотя наличие пароля – обязательно. Офлайн расшифровка DPAPI нисколько не делает эту систему более уязвимой, в тоже время, оставаясь полезной исследователям, экспертам и криминалистам.

Итак, Мастер Ключ пользователя находится в специальном файле-контейнере, в котором могут храниться дополнительные, резервные копии Мастер Ключа. Резервная копия Мастер Ключа используется в том случае, если основная копия по той или иной причине не была расшифрована. Например, после принудительного сброса пароля. Мастер Ключ является критическим элементом в криптографии DPAPI, поэтому его длина составляет 512 бит, и при его шифровании применяются проверенные алгоритмы. Смотрите таблицу 1.

В первой реализации DPAPI, для расшифровки Мастер Ключа был необходим NTLM хэш пароля пользователя. Это был большой, нет, даже огромнейший просчет со стороны разработчиков DPAPI, который сводил на нет всю безопасность системы. А все потому, что потенциальному злоумышленнику для расшифровки Мастер Ключа и, как следствие, любого объекта DPAPI, необходимо было лишь взять NTLM хэш пароля из SAM файла, сам пароль не требовался! Во второй, текущей версии DPAPI эта ошибка была исправлена и теперь для шифрования Мастер Ключа применяется SHA1 хэш.

После того, как Мастер Ключ успешно декодирован, его исходный 512-битный материал участвует в получении симметричного ключа шифрования объекта DPAPI. Для этого Мастер Ключ «перемешивается» со случайными данными (которые будут потом храниться в DPAPI объекте) и энтропией, если она задана. Таким образом, достигается тройная уникальность, если можно так выразиться, ключа шифрования для каждого объекта DPAPI.

Ну и, наконец, происходит непосредственно шифрование исходных данных. В Windows 7 для шифрования применяется алгоритм AES256. На всех этапах этого замысловатого процесса используется «соль», т.е. набор случайных данных, унифицирующий ключи шифрования и предотвращающий таким образом потенциальную атаку перебором с помощью радужных таблиц.

Кстати, немногие знают, что в первых версиях Windows 7 имели место быть серьезные проблемы с шифрованием AES. Попросту говоря, этот тип шифрования не работал вообще. К счастью, проблему быстро устранили.

После того, как данные зашифрованы, они группируются в единую структуру вместе с остальной служебной информацией и отправляются назад приложению в виде объекта DPAPI. Структура объекта DPAPI недокументирована, но исследователям из Passcape удалось полностью ее реверсить (смотрите рис. 3).

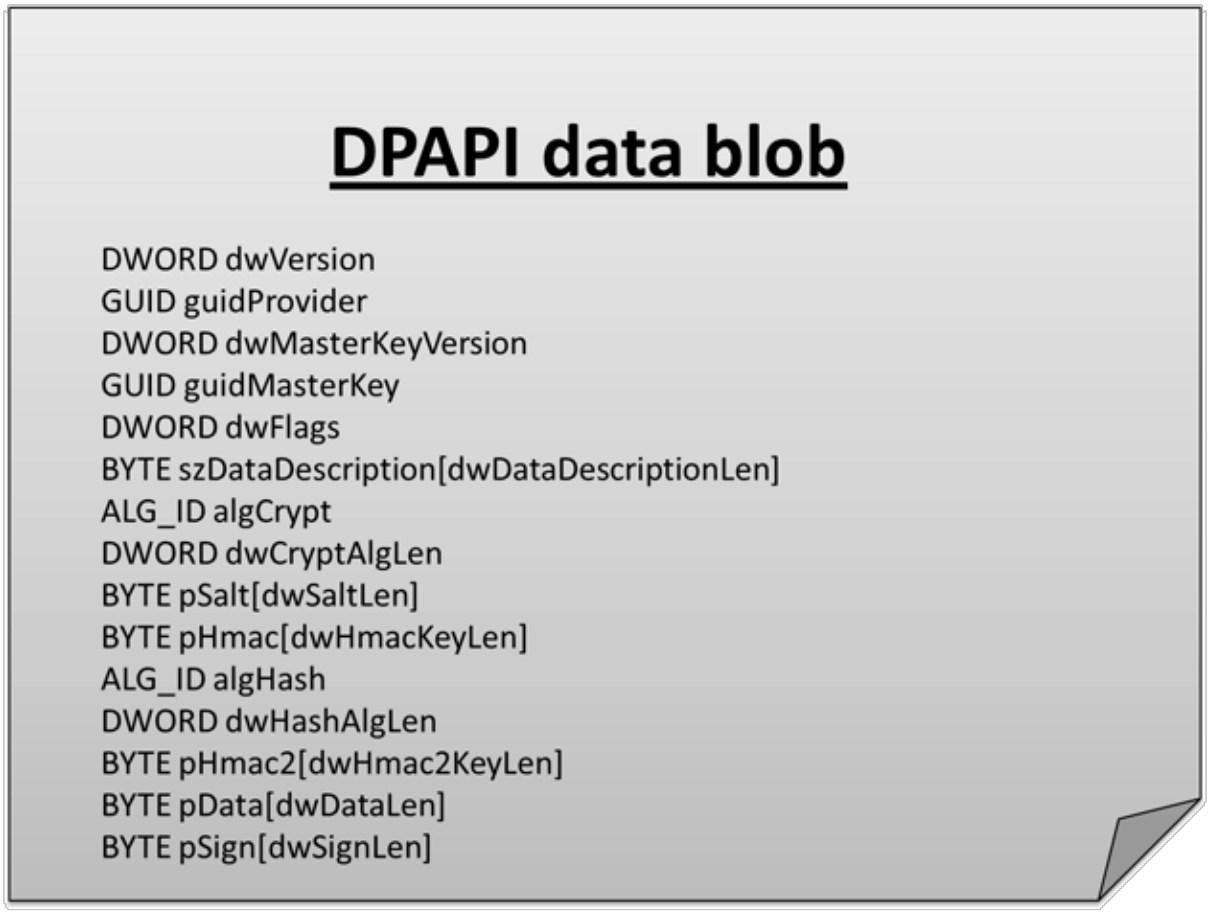


**Рис 3. Недокументированная структура двоичного объекта DPAPI.**

# 4 Мастер Ключи DPAPI

## 4.1 Что такое Мастер Ключ DPAPI

Мастер Ключ пользователя – это двоичный файл, содержащий зашифрованные данные, которые используются для создания первичного ключа шифрования во всех объектах DPAPI. Поскольку Мастер Ключ шифрует конфиденциальные данные пользователя, сам он нуждается в серьезной защите. В качестве исходного материала для защиты Мастер Ключа логично был выбран пароль пользователя.

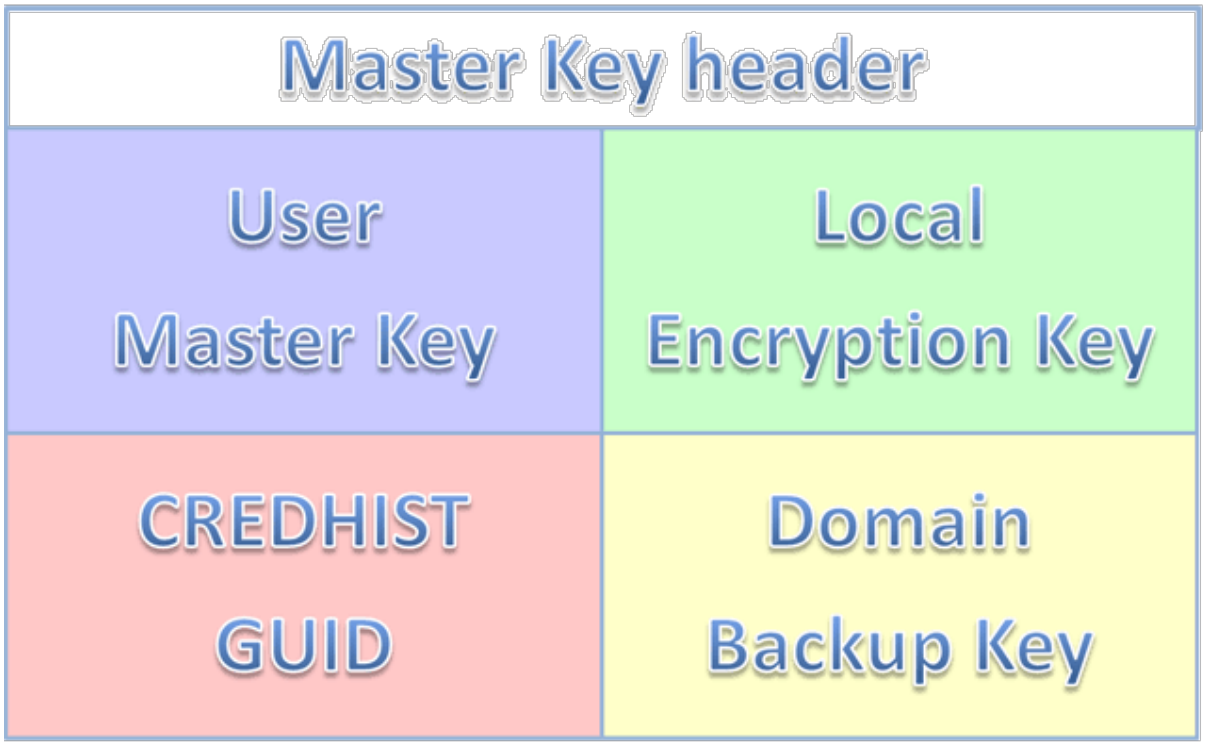


**Рис 4. Структура Мастер Ключа**

## 4.2 Структура Мастер Ключа

Как показано на рисунке 4, файл с Мастер Ключом состоит из 5 частей:
1. Заголовок и служебная информация
2. Мастер Ключ пользователя
3. Ключ шифрования локальной резервной копии
4. Уникальный идентификатор файла CREDHIST
5. Резервная копия Мастер Ключа домена

В Windows 2000 вместо идентификатора CREDHIST, могла храниться резервная локальная копия Мастер Ключа. Именно это делало систему DPAPI чрезвычайно уязвимой. Например, программа Windows Password Recovery может расшифровывать Мастер Ключ из его локальной резервной копии, для этого не требуется знание пароля пользователя.

В заголовке файла с Мастер Ключом находится следующая информация (приведен синтаксис C++, ниже дано описание каждого поля):

```
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 | - версия файла с Мастер Ключом, используемая для проверки совместимости. В Windows 2000 значение этого поля было равно 1, для остальных операционных систем – 2. |
| WCHAR szGuid[0x24] | - строковое значение с уникальным идентификатором Мастер Ключа, обычно совпадающим с именем файла. Любой объект DPAPI хранит этот идентификатор, привязывающий его к определенному (и только одному) Мастер Ключу. |
| DWORD dwPolicy | - поле с различными флагами. Например, если установлен бит 3 в этом поле, то для создания ключа расшифровки Мастер Ключа (из пароля пользователя) будет применен алгоритм SHA1. В Windows 2000 этот флаг всегда сброшен, т.е. используется NTLM хэш. |
| DWORD dwUserKeySize | - содержит размер Мастер Ключа пользователя |
| DWORD dwLocalEncKeySize | - размер ключа шифрования резервных копий |
| DWORD dwLocalKeySize | - размер локального резервного ключа или поля с CREDHIST GUID |
| DWORD dwDomainKeySize | - резервная копия Мастер Ключа домена |

После заголовка последовательно идут 4 слота с Мастер Ключами. Каждый слот содержит свой собственный заголовок и непосредственно сам ключ.
Структура заголовка общая для всех слотов. Для версии 1 (Windows 2000) она выглядит следующим образом:

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

В остальных версиях ОС, структура дополнена новыми полями:

```
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;
```

Описание полей заголовка слотов:

| | |
|---|---|
| DWORD dwVersion | версия слота |
| BYTE pSalt[0x10] | соль, т.е. набор случайных данных для унификации Мастер Ключа и исключения возможной атаки по радужным таблицам |
| DWORD dwPBKDF2IterationCount | счетчик итераций в алгоритме генерации ключа PBKDF2 |
| ALG_ID HMACAlgId | алгоритм проверки целостности данных |
| ALG_ID CryptAlgId | идентификатор алгоритма шифрования слота |
| BYTE pKey[] | зашифрованный Мастер Ключ или данные слота |
| GUID guidCredhist | уникальный идентификатор файла с историей паролей |

В таблице 2 показаны используемые по умолчанию алгоритмы шифрования Мастер Ключей для разных операционных систем, а также анализ защищенности от потенциальной атаки методом грубой силы.

| ОС | CryptAlgId | HMACAlgId | dwPBKDF2IterationCount | Скорость подбора (п/с) |
|---|---|---|---|---|
| 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** |

**Таблица 2. Алгоритмы шифрования Мастер Ключа**

## 4.3 Процесс создания нового Мастер Ключа

Рассмотрим подробнее процесс создания и шифрования нового Мастер Ключа пользователя:

- Сначала вызывается API функция **RtlGenRandom**, возвращающая 64 псевдо-случайно сгенерированных байт. Это и есть исходный материал Мастер Ключа, с помощью которого будут шифроваться объекты DPAPI.
- Исходный материал Мастер Ключа необходимо защитить. Для этого происходит шифрование с использованием пароля на вход пользователя, описателя безопасности (SID) и еще 16 байт случайных данных (т.н. «соль»). Процесс шифрования Мастер Ключа состоит из двух этапов: сначала с помощью функции PBKDF2 (криптографического стандарта, описанного в PKCS #5), из пароля, SID и соли получается ключ. Затем этот ключ используется для получения симметричного ключа шифрования Мастер Ключа. Разные версии Windows, защищают Мастер Ключ по-разному. Если в Windows 2000 по умолчанию применяется криптологический алгоритм **RC4**, то в Windows XP и Win2K3 – **3DES**, а в Windows Vista и выше – **AES256**.
- Мастер Ключу присваивается уникальное имя – **GUID**. Каждый объект DPAPI хранит этот уникальный идентификатор, с которым его связывают теплые дружеские отношения. Иными словами, GUID Мастер Ключа является его «привязкой» к объекту DPAPI.
- Созданный и зашифрованный с помощью пароля пользователя Мастер Ключ вместе с другими служебными данными сохраняется в отдельный файл в папке для хранения Мастер Ключей (Master Key Storage Folder). Master Key Storage Folder – это специальное место на диске, где хранятся и складируются Мастер Ключи. Мастер Ключи пользователя хранятся в каталоге **%APPDATA%\Microsoft\Protect\%SID%**, где **%APPDATA%** - каталог Application Data. Например, C:\Users\John\AppData\Roaming\Microsoft\Protect\S-1-5-21-2893984454-3019278361-1452863341-

1003, а **%SID%** - текстовый описатель безопасности пользователя. В приведенном выше примере это S-1-5-21-2893984454-3019278361-1452863341-1003. Мастер Ключи системы хранятся в каталоге **%WINDIR%\System32\Microsoft\Protect** и используются для расшифровки DPAPI объектов, защищенных из-под учетной записи локального компьютера. Все DPAPI объекты, во время создания которых в функции CryptProtectData был установлен флаг CRYPTPROTECT_LOCAL_MACHINE, защищены Мастер Ключами системы.

- Создание, шифрование или расшифровка Мастер Ключа может занять очень много времени по меркам операционной системы. Например, в Windows 7 расшифровка Мастер Ключа занимает более 0,1 секунды на современном ПК. Однако внутри DPAPI существует кэширование. Однажды расшифрованный, Мастер Ключ помещается в кэш так, что последующий доступ к нему не требует повторной расшифровки.

## 4.4 Регенерация Мастер Ключей

Если вы откроете Master Key Storage Folder у себя на диске, предварительно разрешив системе показ спрятанных папок и файлов (файлы с Мастер Ключом имеют атрибут Скрытый), то наверняка увидите несколько файлов с Мастер Ключами. Почему несколько? А потому что DPAPI в целях безопасности принудительно создает новый Мастер Ключ примерно каждые 90 дней, поэтому любая учетная запись пользователя имеет, как правило, несколько Мастер Ключей, количество которых зависит от даты создания профиля пользователя.

Период регенерации Мастер Ключа жестко прописан в системе. Подразумевается, что его нельзя изменить. Однако есть одно «Но»… В каталоге с Мастер Ключами находится 18 байтовый файл с именем **Preferred**, содержащий имя последнего созданного системой Мастер Ключа, а также дату его создания.

**Структура файла Preferred**

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

Каждый раз, при вызове одной из функций DPAPI, система проверяет дату создания последнего Мастер Ключа в Preferred и, если прошло более 90 дней, создает новый Мастер Ключ, обновляя при этом и Preferred. Регенерацией Мастер Ключей можно управлять принудительно. Для этого вызовите функцию CryptProtectData с установленным флагом CRYPTPROTECT_CRED_SYNC. Система выполнит перешифровку всех Мастер Ключей и обновит их на диске. Обычно эта операция выполняется после смены пароля пользователя. Для предотвращения необязательного обновления Мастер Ключей в Windows 7 был введен механизм синхронизации Мастер Ключей, о котором будет сказано ниже.

Данные в файле Preferred ничем не защищены, поэтому, вручную изменив последнюю дату создания Мастер Ключа, можно самостоятельно управлять процессом регенерации. Например, бесконечно продлить жизнь последнему Мастер Ключу или создать любое количество новых Мастер Ключей. И если продление жизни Мастер Ключу не дает никаких привилегий потенциальному злоумышленнику, то в последнем случае, большое количество Мастер Ключей с последующей сменой пароля может ввести систему в долгий ступор, поскольку при смене пароля пользователя, все Мастер Ключи подвергаются обязательной перешифровке.

Теперь прикиньте самостоятельно, сколько сможет занять времени обработка, допустим, 1000 Мастер Ключей в Windows 7, если на один Мастер Ключ затрачивается более 0.3 секунды. Итак, на наш взгляд, процесс регенерации Мастер Ключей требует небольшой доработки. В идеале, достаточно контроля целостности данных, отсутствие которого является едва ли единственной большой уязвимостью DPAPI, да и уязвимостью это вряд ли можно назвать. В целом, вся система DPAPI очень хорошо сбалансирована и продумана.

## 4.5 Резервирование Мастер Ключей восстановление их из резервных копий

### Резервирование Мастер Ключей на ПК, являющихся членами домена

В случае, когда компьютер является членом домена, файл с Мастер Ключом хранит резервную копию Мастер Ключа пользователя. При создании Мастер Ключа, DPAPI отправляет запрос контроллеру домена, в котором находится пара приватный/публичный ключей шифрования RSA. Мастер Ключ шифруется публичным ключом домена и сохраняется в виде резервной копии в том же файле, что и Мастер Ключ пользователя. Если во время расшифровки основной копии Мастер Ключа произошла ошибка, DPAPI высылает резервную копию контроллеру домена. Контроллер домена расшифровывает ее с помощью своего приватного RSA ключа и отправляет расшифрованный Мастер Ключ обратно.

### Резервирование Мастер Ключей на автономных ПК под руководством Windows XP - Windows 7

Когда речь идет об автономных компьютерах домашних пользователей, DPAPI задействует другой механизм восстановления на основе дискеты сброса паролей. Дискета сброса пароля, которую пользователь может создать в любой момент из панели управления, позволяет восстановить забытый им пароль. Ключевое слово в предыдущем предложении – может. Забота о восстановлении данных целиком лежит на плечах пользователя.

А что будет, если пользователь сменил пароль после того, как была создана дискета сброса пароля? Все будет нормально. Процесс одновременно продуман и прост, как деревенский клозет. При создании дискеты сброса пароля, система создает пару из публичного и 2048 бит приватного ключа шифрования RSA. Текущий пароль пользователя шифруется при помощи публичного ключа и сохраняется в ключе реестре пользователя **HKEY_LOCAL_MACHINE\SECURITY\Recovery\%SID%** (в Windows XP – Windows Vista), либо в ключе реестра **HKEY_LOCAL_MACHINE\C80ED86A-0D28-40dc-B379-BB594E14EA1B** (Windows 7). В любом случае, доступ к этим ключам запрещен всем пользователям системы, включая Администраторов. А приватный ключ, который требуется при расшифровке пароля, записывается на дискету. Таким образом, если после создания дискеты сброса пароля, пользователь сменил пароль, его новый пароль шифруется публичным ключом RSA и сохраняется в реестре вместо старого. Поскольку публичный RSA ключ соответствует приватному ключу, хранимому на дискете, то с помощью приватного ключа можно восстановить и вновь созданный, хранящийся в реестре пароль. В одной из последних версий программа Windows Password Recovery может расшифровывать пароль пользователя при наличии дискеты сброса пароля.

### Резервирование Мастер Ключей на автономных ПК под руководством Windows 2000

Но система сброса пароля появилась только в Windows XP. А до этого? Действительно, несмотря на то, что в ОС Windows 2000 не было дискеты сброса пароля, на автономных компьютерах также осуществлялось резервное хранение Мастер Ключа пользователя. При этом и сам ключ шифрования и резервная копия Мастер Ключа хранились в одном файле с основным Мастер Ключом (смотрите рисунок 5). Это значит, что любые данные, зашифрованные при помощи DPAPI, могли быть расшифрованы потенциальным злоумышленником БЕЗ знания пароля пользователя (при условии физического доступа к системе). Что можно сказать о первой реализации DPAPI в Windows 2000? “Понять и простить”.

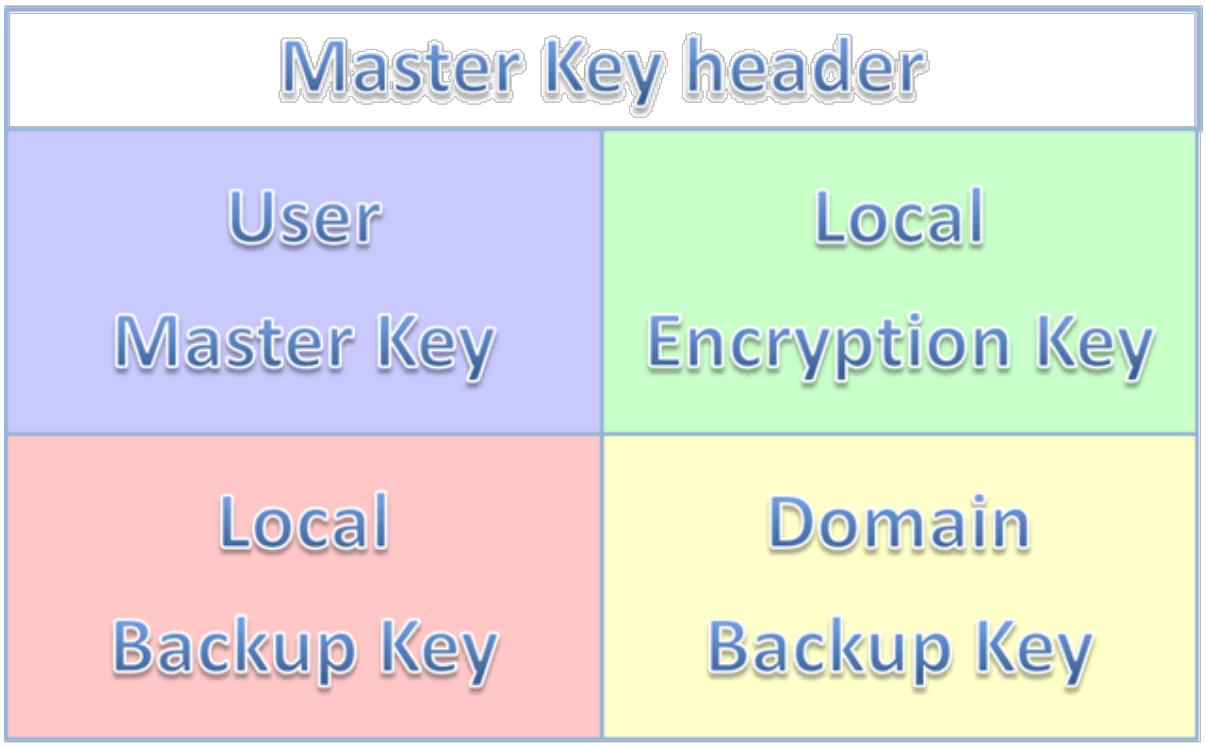


**Рис 5. Структура файла с Мастер Ключом на ОС Windows 2000.**

Компания Microsoft настоятельно рекомендует обновить Windows 2000 до более актуальной версии. Однако не спешите делать это. Мы расскажем вам, как максимально обезопасить старую добрую Win2K от потенциального компрометирования системы. Для этого рассмотрим алгоритм создания локальной резервной копии Мастер Ключа. Файл с Мастер Ключом в Win2K условно состоит из 5 частей (рис. 5):

1. Заголовок и служебная информация
2. Мастер Ключ пользователя
3. Ключ шифрования локальной резервной копии
4. Локальная резервная копия Мастер Ключа
5. Резервная копия Мастер Ключа домена

Восстановление локальной резервной копии Мастер Ключа пользователя начинается с декодирования ключа шифрования локальной резервной копии. В расшифровке используется пароль учетной записи локального компьютера (да, да, есть и такая), который хранится в LSA секрете с именем **DPAPI_SYSTEM**. После того, как получен ключ шифрования локальной резервной копии, им расшифровывается локальная резервная копия Мастер Ключа. Все просто, как 1-2-3.

С этой точки зрения, система Win2K более дружелюбна пользователю, поскольку если даже пароль пользователя будет принудительно стерт, DPAPI всегда сможет восстановить Мастер Ключ из локальной резервной копии. Каким путем достигается такая дружелюбность, объяснять, наверное, не надо: любой пользователь, имеющий доступ к не-доменному ПК под руководством ОС Windows 2000, сможет легко расшифровать зашифрованные DPAPI данные другого пользователя без знания текстового пароля и даже **без знания хэша пароля**!

Чтобы частично воспрепятствовать такому безобразию, нужно изменить режим функционирования SYSKEY. Запустите утилиту SYSKEY (нажмите одновременно клавиши WIN + R, введите syskey.exe и нажмите OK), сменив место хранения SYSKEY. Тогда для того, чтобы расшифровать LSA секрет DPAPI_SYSTEM, потенциальному злоумышленнику необходимо будет знать либо пароль SYSKEY (SYSKEY bootup password), либо иметь дискету загрузки SYSKEY (SYSKEY startup disk).

# 5 История паролей DPAPI

## 5.1 История паролей DPAPI

Корректное функционирование DPAPI подразумевает хранение всех предыдущих паролей пользователя. Все предыдущие пароли хранятся в виде хэшей в специальном файле-контейнере, который называется **CREDHIST** и находится в каталоге **%APPDATA%\Microsoft\Protect**.

## 5.2 Структура CREDHIST

Если файл истории паролей представить в виде цепи, то каждый слот с хэшем пароля представляет собой одно звено этой цепи со следующей двоичной структурой:

```
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;
```

Описание полей структуры.

| | |
|---|---|
| DWORD dwVersion | версия текущего звена цепочки истории паролей |
| GUID guidLink | уникальный идентификатор звена |

| | |
|---|---|
| DWORD dwNextLinkSize | размер следующего звена |
| DWORD dwCredLinkType | тип пароля, который используется в расшифровке текущего звена |
| ALG_ID algHash | идентификатор алгоритма хэширования в функции PBKDF2 |
| DWORD dwPbkdf2IterationCount | количество итераций в PBKDF2 |
| DWORD dwSidSize | размер идентификатора безопасности владельца |
| ALG_ID algCrypt | используемый алгоритм шифрования данных |
| DWORD dwShaHashSize | размер SHA1 хэша |
| DWORD dwNtHashSize | размер NTLM хэша |
| BYTE pSalt[0x10] | случайный набор данных, используемый при шифровании |
| BYTE pSid[] | SID владельца |
| BYTE pShaHash[] | SHA hash |
| BYTE pNtHash[] | NTLM hash |

На рисунке 6 схематично представлена структура хранения хэшей истории в CREDHIST.

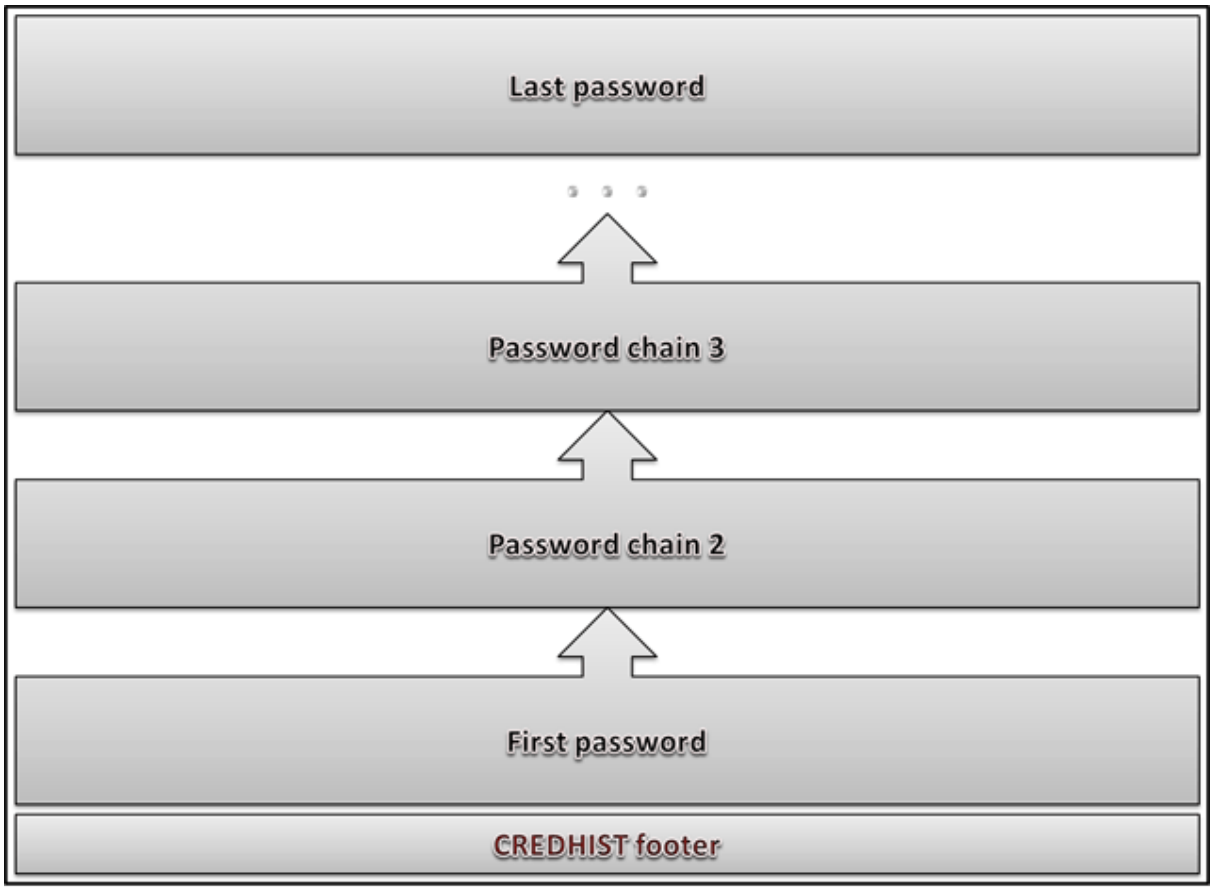


**Рис 6. Структура файла CREDHIST**

## 5.3 Шифрование CREDHIST

Слоты с хэшами пользователя хранятся последовательно, друг за другом. Каждый хэш пароля зашифрован предыдущим хэшем, а первый хэш – текущим паролем пользователя. Поэтому, для расшифровки всей цепочки необходимо знать текущий пароль пользователя (рисунок 7).

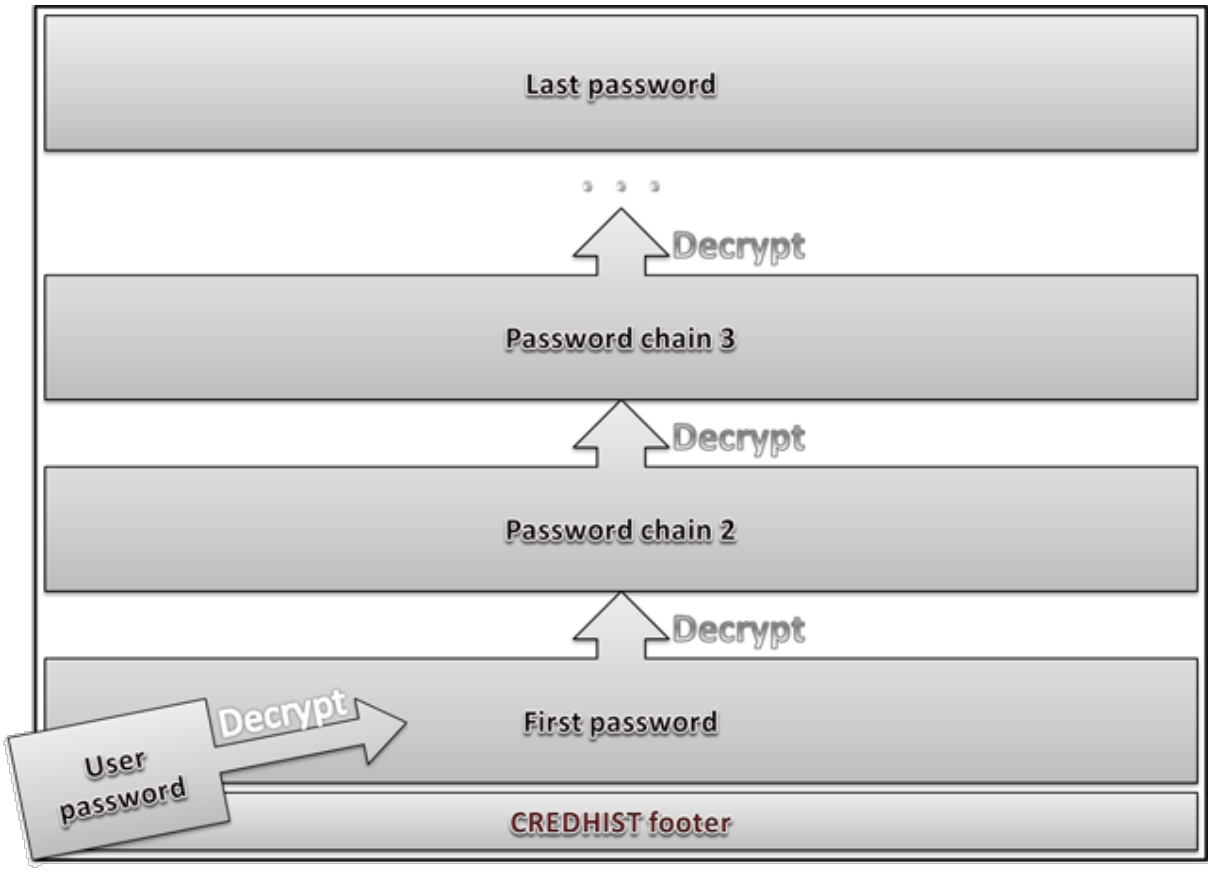


**Рис 7. Схема расшифровки хэшей истории паролей DPAPI**

Если при расшифровке Мастер Ключа паролем пользователя произошла ошибка, DPAPI при помощи текущего пароля расшифровывает первый хэш в CREDHIST и делает попытку расшифровать им Мастер Ключ. Если Мастер Ключ снова не поддался, первый хэш истории CREDHIST расшифровывает второй хэш, и операция повторяется.

Разные операционные системы по умолчанию используют различные параметры и алгоритмы шифрования истории паролей. Смотрите таблицу внизу. В целом, алгоритм шифрования похож на тот, который используется при защите Мастер Ключей.

| ОС | Алгоритм шифрования | Алгоритм хэширования | Счетчик PBKDF2 | Скорость подбора (п/с) |
|---|---|---|---|---|
| Windows XP | 3DES | SHA1 | 4000 | **76** |
| Windows Vista | 3DES | SHA1 | 24000 | **12** |
| Windows 7 | AES256 | SHA512 | 5600 | **10** |
| Windows 10 | AES256 | SHA512 | 8000 | **<10** |

**Таблица 3. Алгоритмы шифрования истории паролей в различных ОС**

## 5.4 Недостатки CREDHIST

В CREDHIST хранятся все предыдущие хэши паролей, даже если эти пароли были одинаковые. Чисто теоретически не исключена ситуация, когда пользователь мог задать пароль, который уже был ранее (назовем его дубликат). Тогда для того, чтобы вычислить текущий пароль, достаточно узнать один из паролей истории, который был задан в промежутке между текущим паролем и дубликатом.

Поэтому, целесообразнее использовать DPAPI с политикой безопасности Windows, запрещающей введение повторных паролей, либо усовершенствовать систему DPAPI таким образом, чтобы исключить возможность хранения дубликатов в CREDHIST.

# 6 DPAPI и смена пароля

## 6.1 Смена пароля пользователя и DPAPI

Смену пароля в Windows условно можно разделить на несколько критических этапов. Вначале происходит проверка валидности старого пароля, проверка нового на соответствие политики безопасности и физическое изменение пароля в базах SAM или NTDS.DIT. На втором этапе, если все прошло удачно, происходит обновление внутреннего кэша паролей. Затем отрабатывает процедура смены пароля в DPAPI. И, наконец, обновляется биометрический пароль текущего пользователя (Windows 7 и выше).

Система DPAPI достаточно умна, чтобы обрабатывать различные сценарии изменения пароля пользователя. Рассмотрим, что происходит при этом в DPAPI. Вот типичные сценарии:

- Обычное изменение пароля пользователя с указанием старого пароля (автономные ПК под руководством Windows XP - Windows 7)
- Сброс или перезаписывание пароля, например, администратором, без указания старого (автономные ПК под руководством Windows XP - Windows 7)
- Сброс пароля пользователя домена (Windows 2000-2008)
- Сброс пароля пользователя автономного ПК в Windows 2000

## 6.2 Обычная смена пароля

Политика безопасности Windows подразумевает регулярную смену пароля пользователя, при которой DPAPI должна обеспечивать пользователю тот же уровень доступа к персональным, зашифрованным данным, как и до смены пароля.

Это достигается путем трехэтапной синхронизации всех Мастер Ключей пользователя, на первом этапе которой все Мастер Ключи подвергаются расшифровке старым паролем пользователя, на

втором все ключи перешифровываются новым паролем с сохранением на диск и, наконец, осуществляется их резервное копирование, если установлена соответствующая опция.

Перешифровка всех Мастер Ключей может занять очень много времени, поэтому она выполняется в отдельном потоке внутри LSA. В Windows 7 был также добавлен интересный механизм синхронизации Мастер Ключей: DPAPI хэширует все Мастер Ключи в профиле пользователя и записывает полученный в результате этого процесса хэш в файл **SYNCHIST**. В следующий раз, чтобы определить, были ли произведены к.л. изменения, DPAPI снова хэширует всю пачку Мастер Ключей и сравнивает полученный хэш с тем, что хранится в SYNCHIST, предотвращая таким образом ненужную и времязатратную операцию повторной перешифровки.

Предвосхищает операцию синхронизации Мастер Ключей обновление файла CREDHIST. Старый хэш пароля пользователя шифруется новым и помещается в начало цепочки истории хэшей в CREDHIST. Если история паролей пустая, то создается первое звено в цепи хэшей.

В финале, обновляется пароль дискеты сброса, она была создана ранее. Для этого система ищет в хранилище сертификатов публичный RSA ключ, шифрует им новый пароль и записывает его вместо старого в реестр пользователя.

## 6.3 Сброс пароля на автономных ПК (Windows XP - Windows 7)

Это тот случай, когда, например Администратор, принудительно сбрасывает пароль пользователя или пароль был стерт с помощью одной из программ для сброса паролей.

Несмотря на то, что пользователь сможет войти в систему, все данные, зашифрованные при помощи DPAPI, будут недоступны (например, пароли сети, ключи EFS, сертификаты почты и пр.), ведь старый пароль пользователя неизвестен и DPAPI не удастся расшифровать Мастер Ключ.

Восстановить полноценный доступ к системе (после принудительного сброса пароля) можно только при помощи ранее созданной дискеты сброса пароля.

Если после принудительного сброса пароля, опять установить старый пароль, DPAPI вернется к своему исходному состоянию и все DPAPI объекты будут доступны снова. Это происходит благодаря тому, что DPAPI предусмотрительно хранит все копии Мастер Ключей, даже если они не могут быть расшифрованы.

## 6.4 Принудительный сброс пароля пользователя домена

Если DPAPI используется в среде Active Directory (Windows 2000-2008), Мастер Ключи хранят две свои копии. Первая копия защищена текущим паролем пользователя, а вторая зашифрована публичным RSA ключом, принадлежащим контроллеру домена.

При принудительном сбросе пароля пользователя, DPAPI не сможет расшифровать первую копию Мастер Ключа. Тем не менее, доступ ко всем объектам DPAPI будет восстановлен при помощи второй резервной копии. При этом рабочая станция отправляет зашифрованную резервную копию Мастер Ключа контроллеру домена. Контроллер домена расшифровывает копию с помощью своего приватного RSA ключа и отправляет расшифрованный Мастер Ключ обратно. Рабочая

станция заново шифрует Мастер Ключ при помощи нового пароля пользователя, обновляя файл с обеими копиями и синхронизирует остальные файлы.

## 6.5 Сброс пароля на автономном ПК под управлением Windows 2000

На автономных компьютерах с установленной Win2K, дискета сброса пароля не используется. В файле с Мастер Ключом находятся две копии. Первая зашифрована при помощи пароля пользователя, а вторая с помощью LSA секрета DPAPI_SYSTEM, хранящегося в реестре и доступного только системной учетной записи.

После того, как пароль пользователя был принудительно сброшен, DPAPI не сможет расшифровать первую копию Мастер Ключа, поэтому автоматически будет расшифрована вторая копия. Затем расшифрованная копия защищается новым паролем пользователя и сохраняется обратно в файл. Все остальные Мастер Ключи тоже подвергаются синхронизации.

С точки зрения конечного пользователя, процесс доступа к конфиденциальной информации, защищенной DPAPI, полностью непрерывный. Для обеспечения должной защиты конфиденциальных данных DPAPI на автономных ПК под управлением Windows 2000, рекомендуется изменить настройки SYSKEY таким образом, чтобы при загрузке системы было необходимо было ввести пароль загрузки SYSKEY или предоставить загрузочную дискету SYSKEY.

# 7 Восстановление паролей DPAPI

## 7.1 Восстановление данных из объектов DPAPI

Набор утилит в программе Windows Password Recovery для работы с DPAPI впервые предоставляет широкие возможности в восстановлении паролей и данных, зашифрованных при помощи DPAPI, в режиме офлайн, т.е. без загрузки профиля пользователя. Ниже описаны типичные сценарии работы с программой при восстановлении данных.

## 7.2 Поиск DPAPI объектов на диске

DPAPI объекты в Windows могут храниться где угодно: в двоичных и текстовых файлах, в базах данных, реестре, Active Directory и т.д. Перед тем, как приступить к расшифровке объекта DPAPI, его необходимо «вытащить» и привести в надлежащий вид. Для этого нам понадобится утилита поиска объектов DPAPI на диске.

Запускаем утилиту и выбираем исходный каталог, в котором будет происводиться поиск (рисунок 8). Допустим, пусть исходным каталогом будет C:\ProgramData\Microsoft\Wlansvc. В нем находятся пароли соединений беспроводной сети Windows 7.

Пароли WiFi, зашифрованные DPAPI, хранятся в текстовом виде в xml файлах, поэтому устанавливаем соответствующие опции для поиска текстовых блобов, по умолчанию эта опция включена. Выбрав целевой каталог, куда будут помещены найденные файлы, смело жмем клавишу Start.

Обнаруженные объекты DPAPI будут сохранены в целевой каталог под соответствующими именами. Например, если программа обнаружила DPAPI объект в файле xxx.xml, он будет приведен в двоичный вид и сохранен в целевом каталоге под именем xxx.xml.001. Если в файле yyy.xml обнаружится два объекта, они будут записаны под именами yyy.xml.002 и yyy.xml.003.

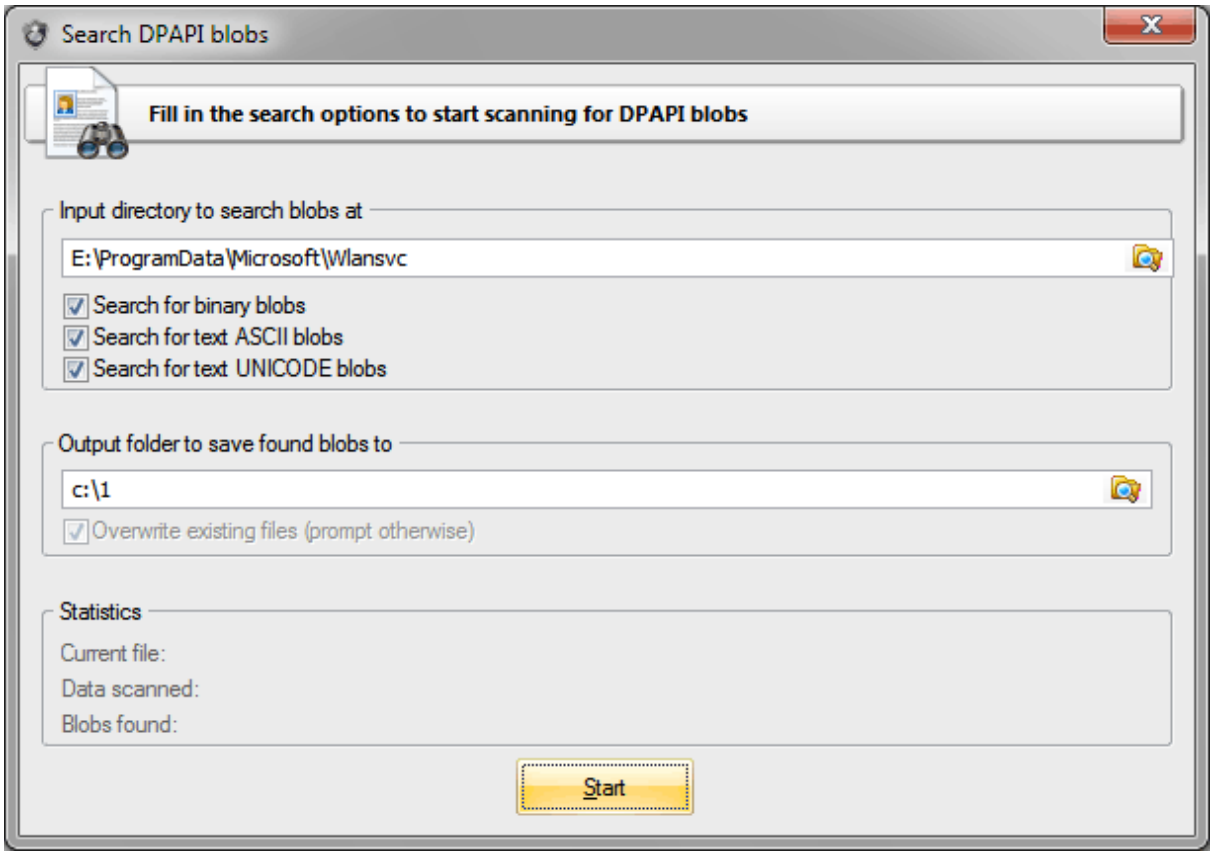


**Рис 8. Поиск паролей WiFi.**

## 7.3 Поиск DPAPI объектов в реестре и Active Directory

Поиск DPAPI объектов на диске не представляет особого труда. Но как быть, если необходимо просканировать реестр текущего пользователя или базу данных Active Directory? Ведь доступ к этим файлам запрещен всем, даже администраторам.

Затруднительно, но к счастью, не невозможно. В этом нам пригодится инструмент для резервного копирования реестра и Active Directory, работа с которым не представляет особого труда даже начинающим пользователям.

Запустив его, выбираем куда и что именно нам нужно зарезервировать: реестр или Active Directory. Учтите, обработка базы данных Active Directory может занять много времени. Теперь можно переходить к утилите поиска DPAPI объектов с указанием пути к каталогу, в котором были сохранены реестр или база ntds.dit.

## 7.4 Восстановление паролей соединений беспроводной сети

Расшифровке паролей беспроводной сети предшествует поиск соответствующих объектов DPAPI в каталоге C:\ProgramData\Microsoft\Wlansvc, описанный выше. Поэтому опустим здесь его повторное описание.

Запускаем утилиту расшифровки DPAPI и указываем путь к извлеченному объекту DPAPI. Пароли беспроводных соединений защищены при помощи флага CRYPTPROTECT_LOCAL_MACHINE, поэтому для их восстановления потребуется системный Мастер Ключ из каталога **C:\Windows\System32\Microsoft\Protect\S-1-5-18\User**.

Находим в нем нужный нам файл с Мастер Ключом. По умолчанию, диалог открытия файлов показывает только интересующий нас файл. Путь к CREDHIST в нашем случае не понадобится, оставляем это поле пустым.

После перехода к третьему шагу мастера программы отмечаем, что пароль пользователя не нужен. Вместо него нам понадобятся два файла реестра: **SYSTEM** и **SECURITY**. Что ж, указываем и их. Учтите, текущие файлы реестра необходимо предварительно разблокировать (т.е. выполнить их резервное копирование). В поле с SID пользователя программа автоматически подставляет SID системы: S-1-5-18. Оставляем это поле и завершаем расшифровку, получив оригинальный текстовый пароль беспроводной сети Windows 7.

Просто? Тогда идем дальше.

## 7.5 Восстановление пароля Facebook, сохраненного в Internet Explorer

Теперь задачка посложнее. Попробуем восстановить пароль учетной записи Facebook в Internet Explorer. Для этого придется основательно потанцевать с бубном в руках, а посему заранее запасаемся терпением.

Пароли автозаполнения IE хранятся в ветке реестра **HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\IntelliForms\Storage2**. Запускаем Regedit и открываем эту ветку. Там вы наверняка увидите несколько двоичных записей. Нам нужна только одна из них с именем **F6FFE33B9EF4D7CB8F5A2F41F3222D21E131ED787A**.

Если такой записи у вас нет, попробуйте создать тестовый пароль на Facebook: откройте страницу http://www.facebook.com/ (обязательно с www), введите любой адрес электронной почты, тестовый пароль и нажмите кнопку войти (рисунок 9). После того, как Internet Explorer любезно предложит сохранить новый пароль, согласитесь. Естественно, сайт Facebook выдаст ошибку, но это уже неважно – пароль был сохранен.

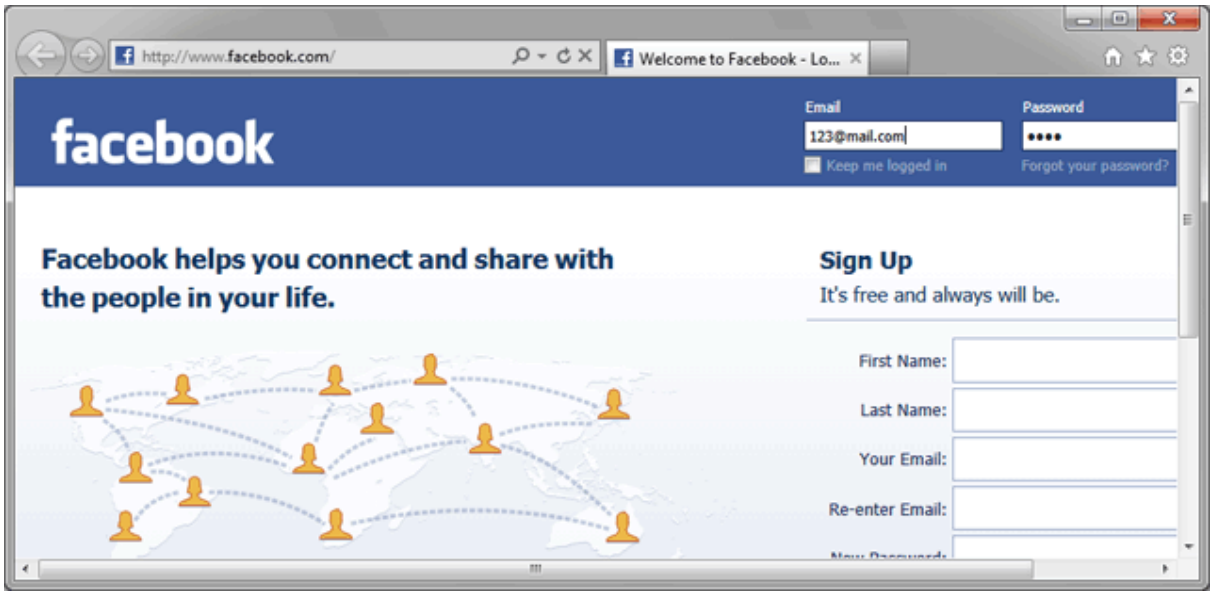


**Рис 8. Создание тестового пароля Facebook**

Переключитесь обратно на regedit и нажмите F5, чтобы обновить данные. В ключе реестра с паролями Internet Explorer должна появиться новая запись с именем F6FFE33B9EF4D7CB8F5A2F41F3222D21E131ED787A. Такое интересное название есть не что иное, как SHA хэш названия сайта.

Открыв ее, внутри обнаруживается бъект DPAPI, в котором зашифрован логин и пароль к этому сайту. Давайте экспортируем DPAPI блоб в текстовый файл. Для этого откроем cmd и выполним следующую команду:

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

Кавычки обязательны! Наш пароль будет сохранен в ASCII виде в каталог c:\test, Можете задать любой другой, не суть важно. Теперь, все, что нам осталось, это вытащить из ASCII файла fb.txt интересуемый нас двоичный объект DPAPI.

Запускаем утилиту поиска, указываем исходный каталог c:\test, устанавливаем опцию поиска в текстовых файлах и получаем на выходе двоичный файл fb.txt.001. Ну наконец можно приступить непосредственно к восстановлению пароля.

Открываем соответствующую утилиту и задаем путь к fb.txt.001. На втором шаге указываем Мастер Файл, которым был зашифрован этот объект. Все Мастер Ключи пользователя хранятся в каталоге **%AppData%\Roaming\Microsoft\Protect\%SID%**. На один уровень выше страдает от своего одиночества файл **CREDHIST**. Он нам тоже может пригодиться.

Самое интересное начинается при переходе к третьему шагу мастера программы. Для расшифровки пароля Facebook, нам необходимо корректно выставить три параметра: **SID** пользователя, его **пароль** на вход и **секрет**, которым был зашифрован DPAPI объект. SID пользователя программа обычно извлекает сама из пути к Мастер Ключу. Если этого не произошло, вам придется добыть этот параметр самостоятельно (например, путем анализа файла CREDHIST). С паролем на вход пользователя тоже все понятно. А как быть с секретом?

В качестве секрета, при шифровании пароля, Internet Explorer использует имя страницы, на которой он был введен. Страница должна быть в нижнем регистре, в формате UNICODE и оканчиваться нулем. Если у вас под рукой есть HEX редактор и вы имеете достаточно навыков для комфортной работы в нем, попробуйте самостоятельно создать этот секрет. Иначе, просто скачайте его отсюда.

Итак, указав в параметре энтропии путь к файлу с секретом, заканчиваем, наконец, нашу эпопею восстановлением пароля Facebook. Расшифровав его, вы увидите, что пароль, логин и другая служебная информация объеденены в единую структуру данных (рисунок 10). Пусть это вас не смущает.

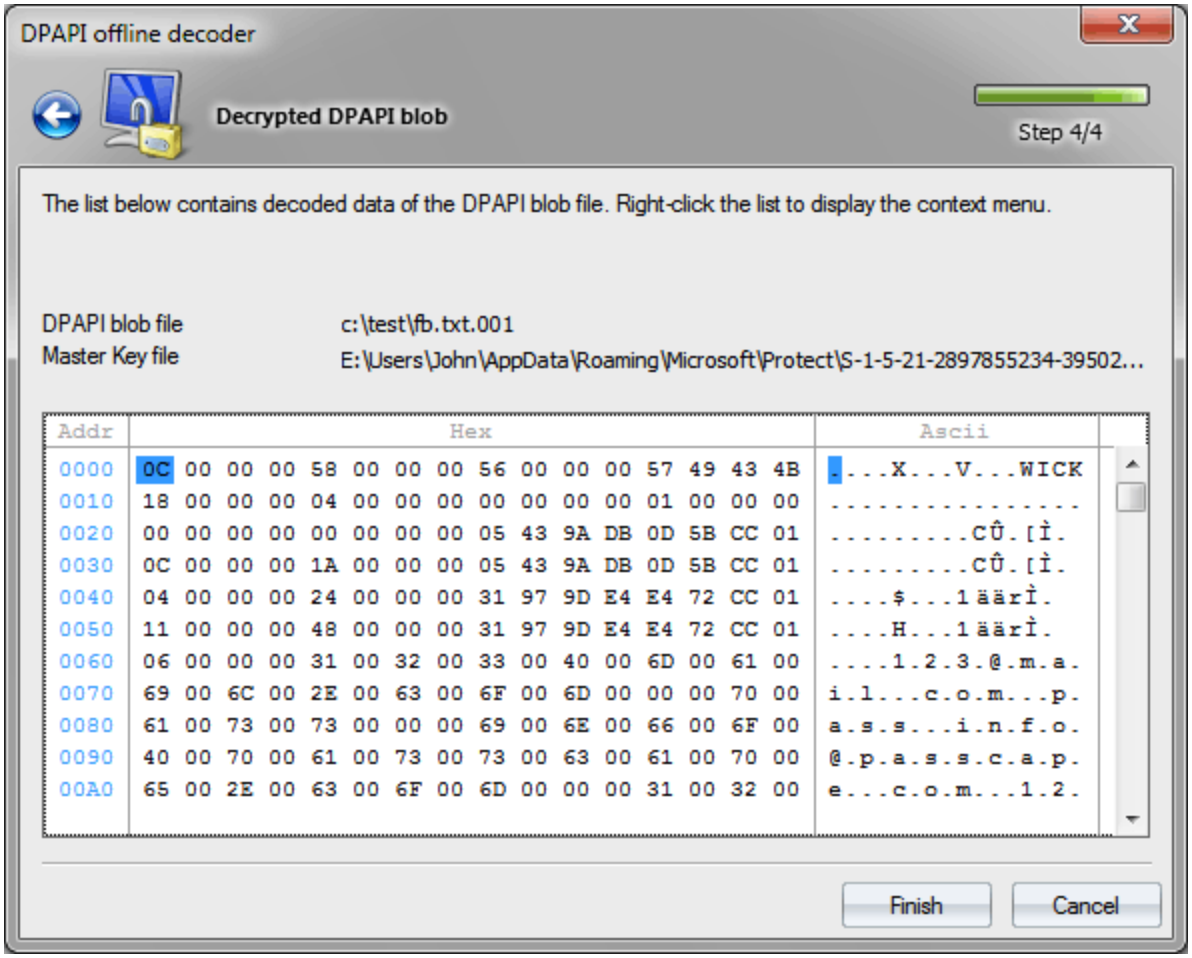


**Рис 10. Расшифрованный пароль Facebook**

Аналогичным образом, проявив чуточку фантазии, можно восстанавливать пароли Internet Explorer из незагружаемой машины.

## 7.6 Восстановление пароля пользователя без загрузки хэшей SAM/NTDS.DIT

Впервые в мире программа Windows Password Recovery предоставляет возможность восстановления пароля на вход без загрузки хэшей пользователя из SAM или NTDS.DIT. Это стало возможным благодаря тому, что шифрование Мастер Ключа происходит при участии пароля пользователя. Следовательно, Мастер Ключ можно использовать для проверки валидности пароля.

Чтобы наглядно продемонстрировать сказанное, запустите утилиту анализа Мастер Ключей и укажите путь к к.л. файлу с Мастер Ключом. После успешной обработки Мастер Ключа, будет показана его внутренняя структура. Щелкните на ней правой кнопкой мыши, выберите пункт проверки пароля с использованием словаря и укажите путь к любому словарю.

Несмотря на свою революционность, восстановление пароля пользователя из Мастер Ключа практической пользы не несет. Например, в Windows 7 проверка паролей идет со скоростью около 10 п/с (рисунок 11). Теоретически, эту скорость можно повысить на пару порядков путем оптимизации алгоритмов шифрования и задействованием GPU при проверке. Но даже после этого, скорость подбора пароля по Мастер Ключу в сравнении с NTLM хэшем будет казаться смехотворной. В плане безопасности, у DPAPI все в порядке.

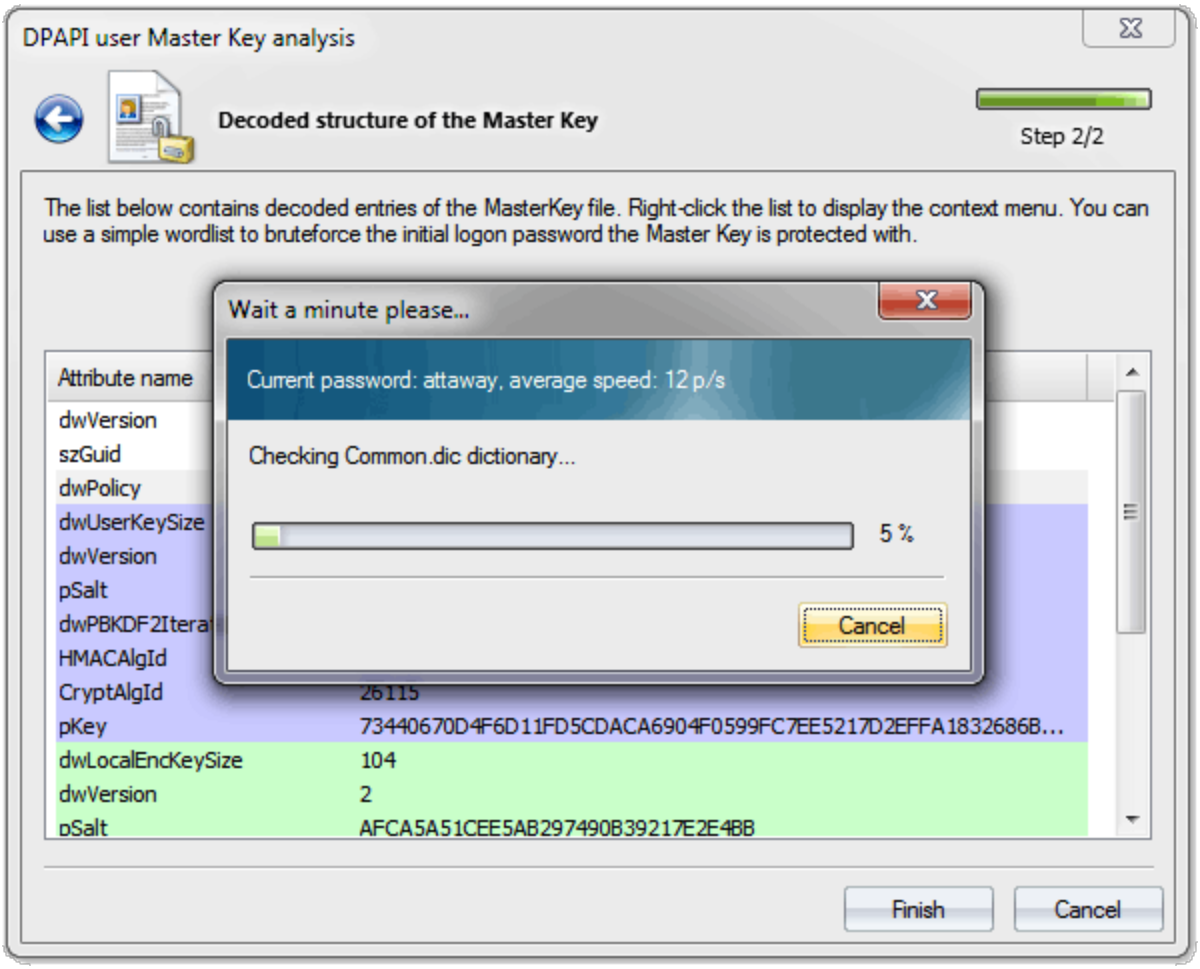

**Рис 11. Восстановление пароля пользователя анализом Мастер Ключа DPAPI в Windows Vista**

## 7.7 Расшифровка хэшей истории паролей DPAPI

Откройте утилиту дампа истории паролей, указав путь к файлу CREDHIST, который находится в каталоге **%AppData%\Roaming\Microsoft\Protect**. В случае, если этот файл хранит предыдущие пароли пользователя, то в поле "**CREDHIST key chains**" должно быть значение, отличное от нуля.

Предыдущие пароли пользователя хранятся в виде **SHA1** и **NTLM** хэшей. В утилите имеется соответствующий переключатель (рисунок 12). Если вы выбрали файл CREDHIST текущего пользователя, пароль на вход вводить не обязательно, программа сама вытащит его из кэша Windows. Для этого просто установите соответствующую опцию. Иначе придется ввести текущий пароль владельца.

Утилита поддерживает частичный дамп. Это означает, что в случае, когда текущий пароль пользователя неизвестен, но есть один из старых паролей, программа сможет расшифровать те хэши, которые использовались ранее, т.е. до того, как был введен этот старый пароль.

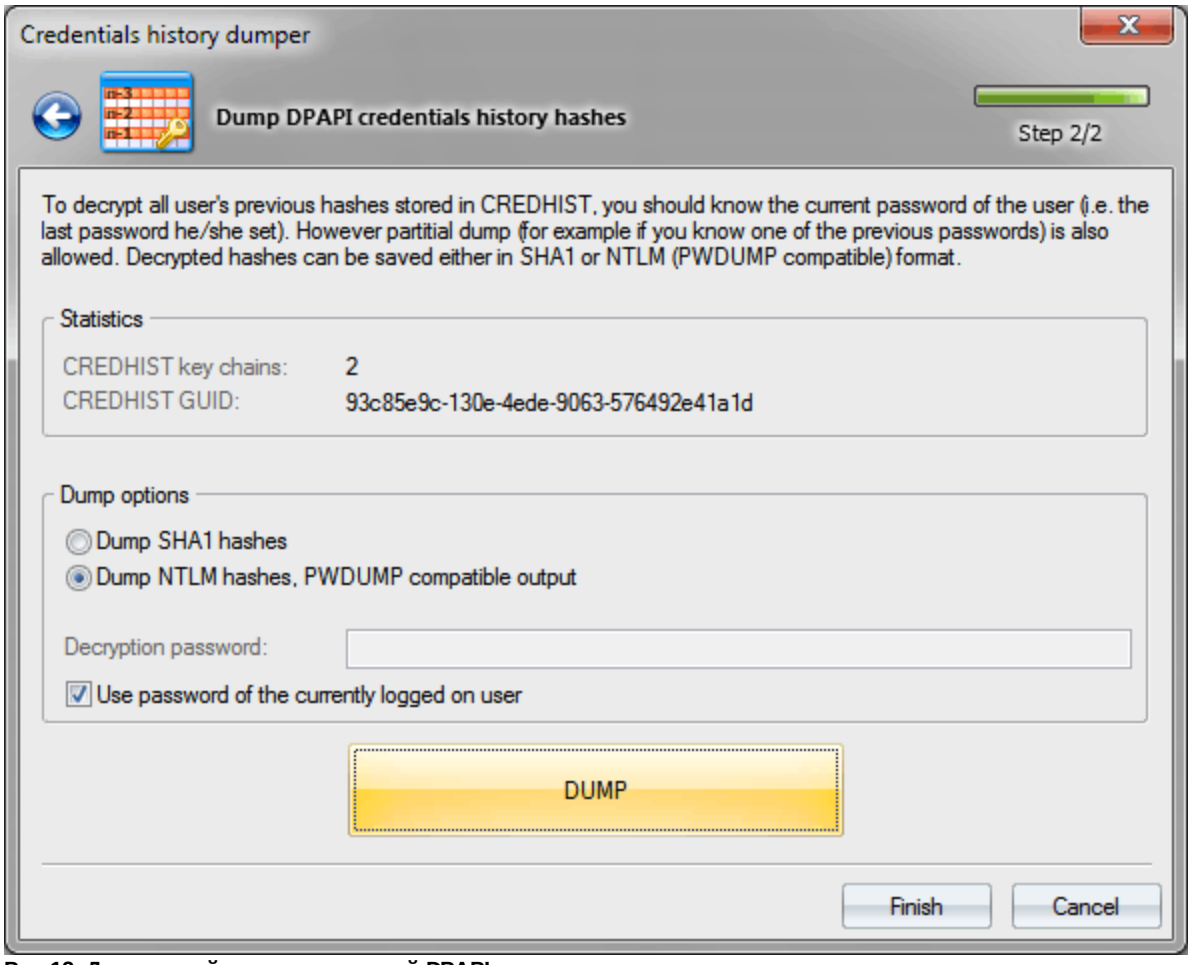


**Рис 12. Дамп хэшей истории паролей DPAPI**

# 8 Заключение

DPAPI заслуживает такого пристального внимания, по крайней мере, потому, что это единственная система на основе пароля, обеспечивающая надлежащую и тщательно продуманную защиту персональных данных пользователя. Ни в одной операционной системе нет более достойной альтернативы DPAPI !

Стоит, пожалуй, оговориться, что в первой реализации DPAPI было допущено несколько серьезных изъянов, благодаря которым, защищенные DPAPI данные пользователя легко могли быть скомпрометированы потенциальным злоумышленником.

Первый блин, как известно, всегда комом. Во всех последующих операционных системах, начиная с Windows XP, эти уязвимости были не только устранены, но и вся система DPAPI подверглась серьезной ревизии. В частности, были применены новые алгоритмы шифрования, благодаря которым скорость подбора пароля к Мастер Ключам уменьшилась приблизительно в 1000 (!) раз. Были исправлены ошибки шифрования Мастер Ключей, которые приводили к тому, что любой пользователь мог получить доступ к любым файлам, зашифрованным EFS. Система локального резервирования Мастер Ключей была заменена дискетой сброса пароля и др.

В целом система шифрования DPAPI стала более продуманной, мощной, отвечающей самым строгим требованиям парольной безопасности.