Decrypting DPAPI data

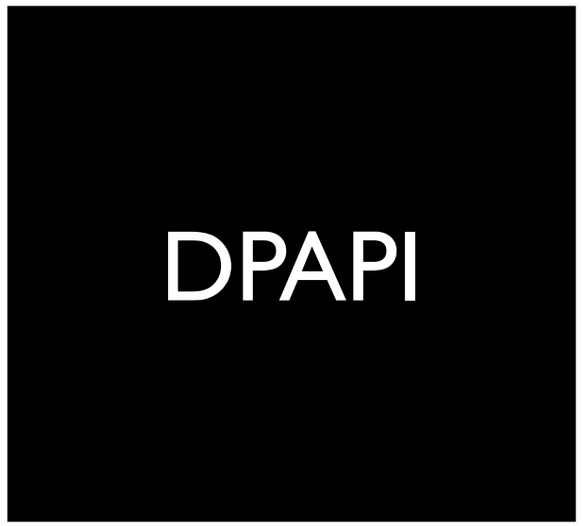
Jean-Michel Picod, Elie Bursztein
EADS, Stanford University

Data Protection API

- Introduced in Windows 2000
- Aim to be an easy way for application to store safely data on disk
- Tie encryption key to user password

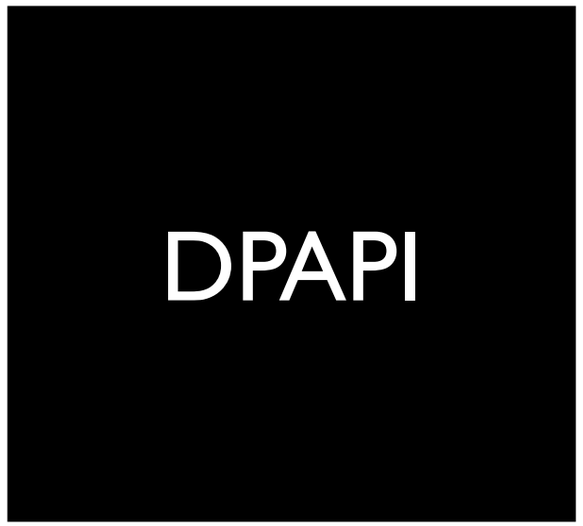
Developer point of view

Application



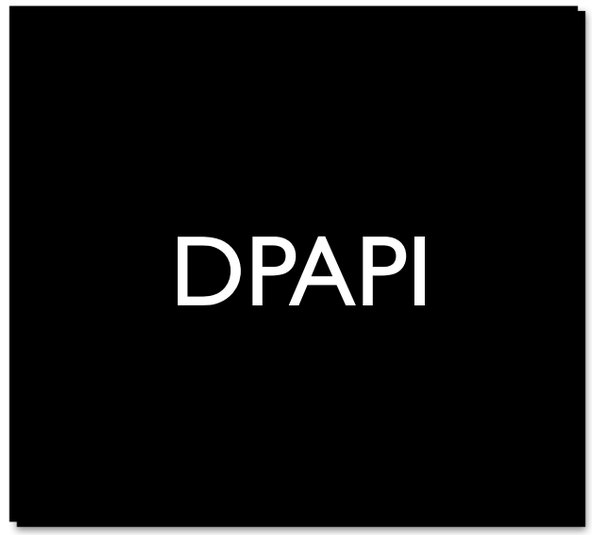
Developer point of view

Application



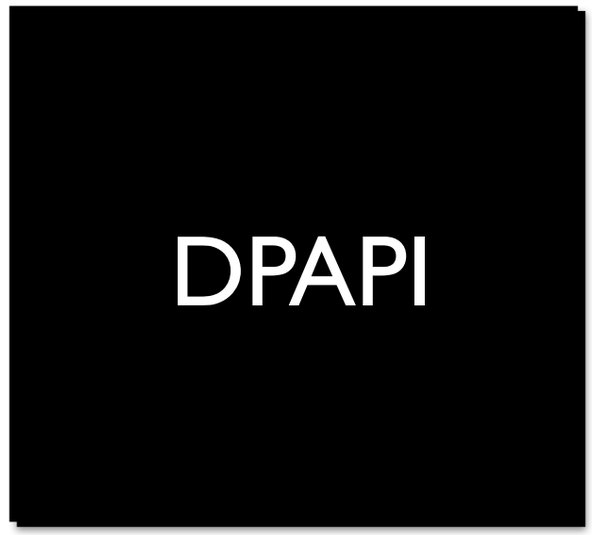
Developer point of view

Application



Developer point of view

Application




Wi
Fi

DPAPI is a simple API*

*http://msdn.microsoft.com/en-us/library/ms995355.aspx

Why digging deeper ?

- Offline forensic
- EFS on Linux
- Security ?

Previous work

- Multiples attempts to analyze DPAPI
  - Some incomplete (Wine)
  - Some close source (Nir Sofer - NirSoft)

Take away

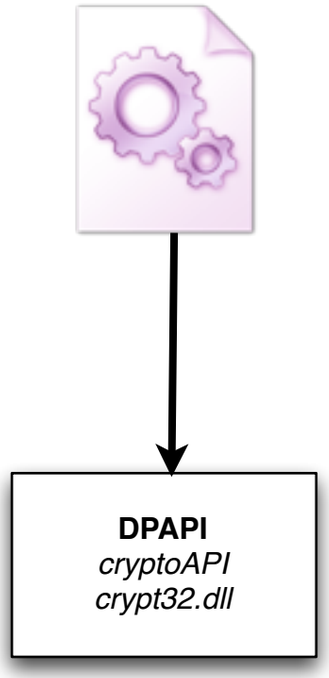
- Decrypt offline sensitive data
- Recovers user previous password
- Do a key escrow attack

Outline

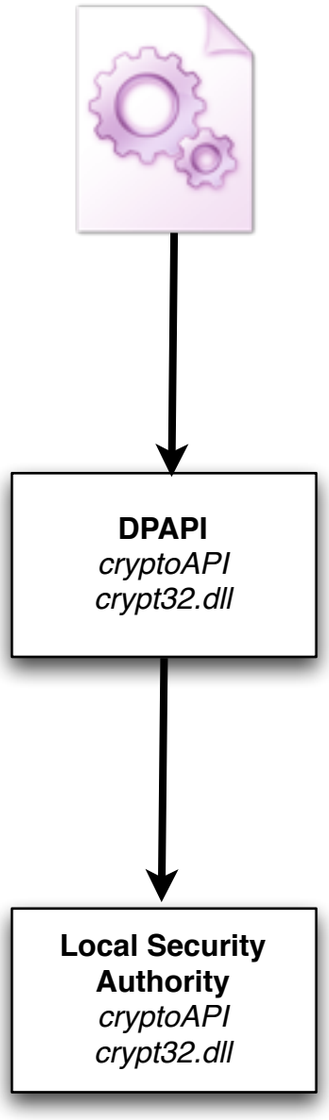
- DPAPI overview
- Decryption process
- Security design implications
- DPAPIck demo

How the system interacts with DPAPI

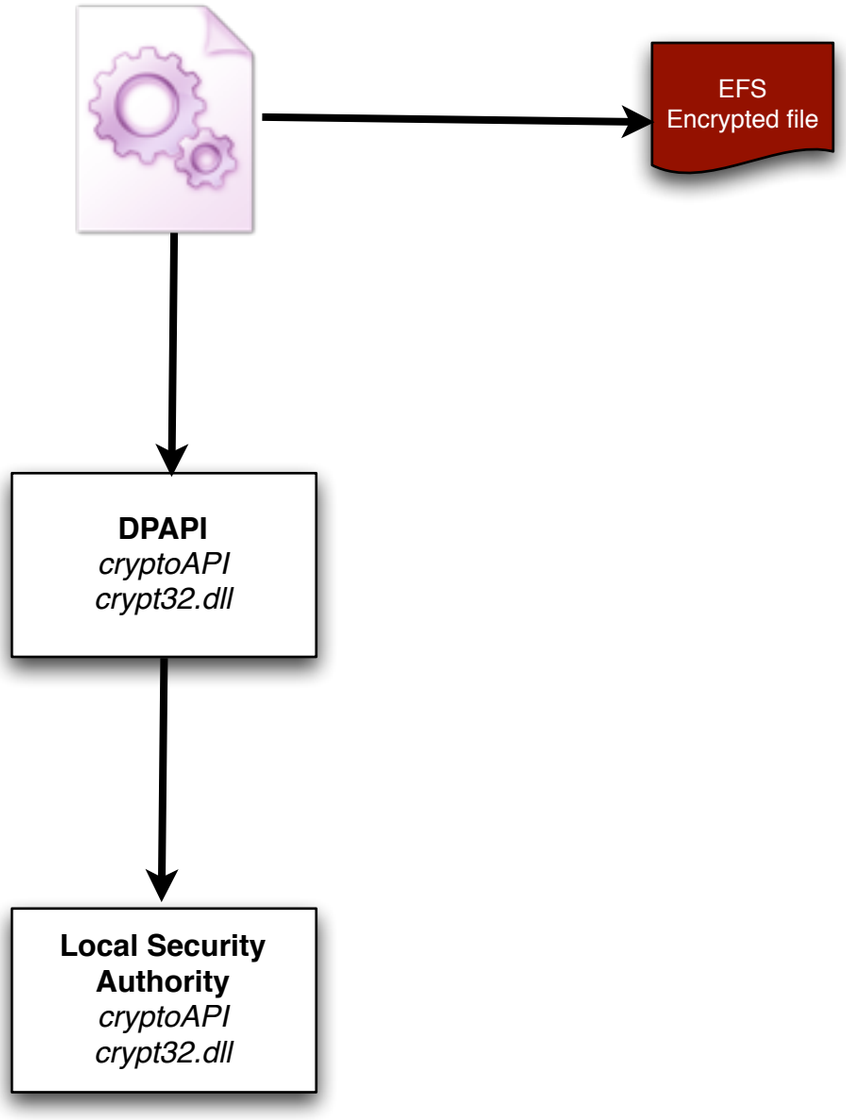
How the system interacts with DPAPI



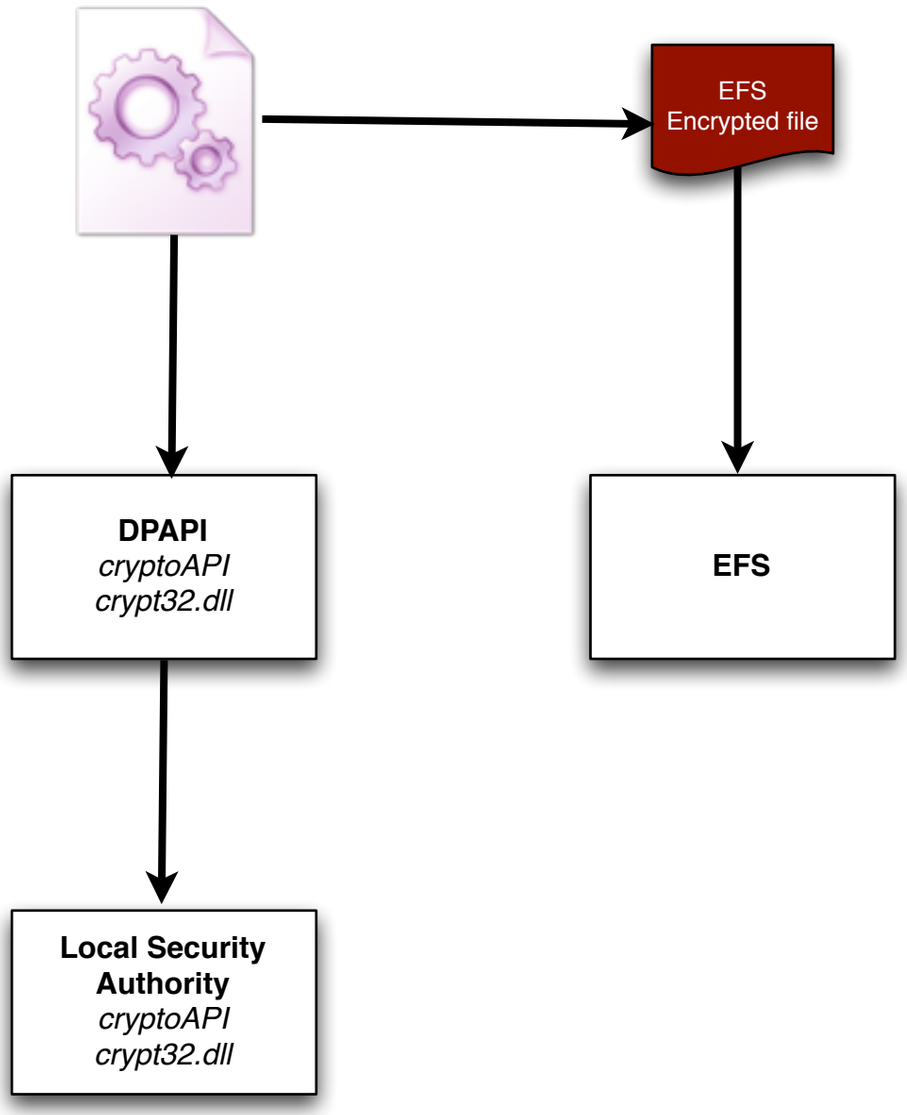
How the system interacts with DPAPI



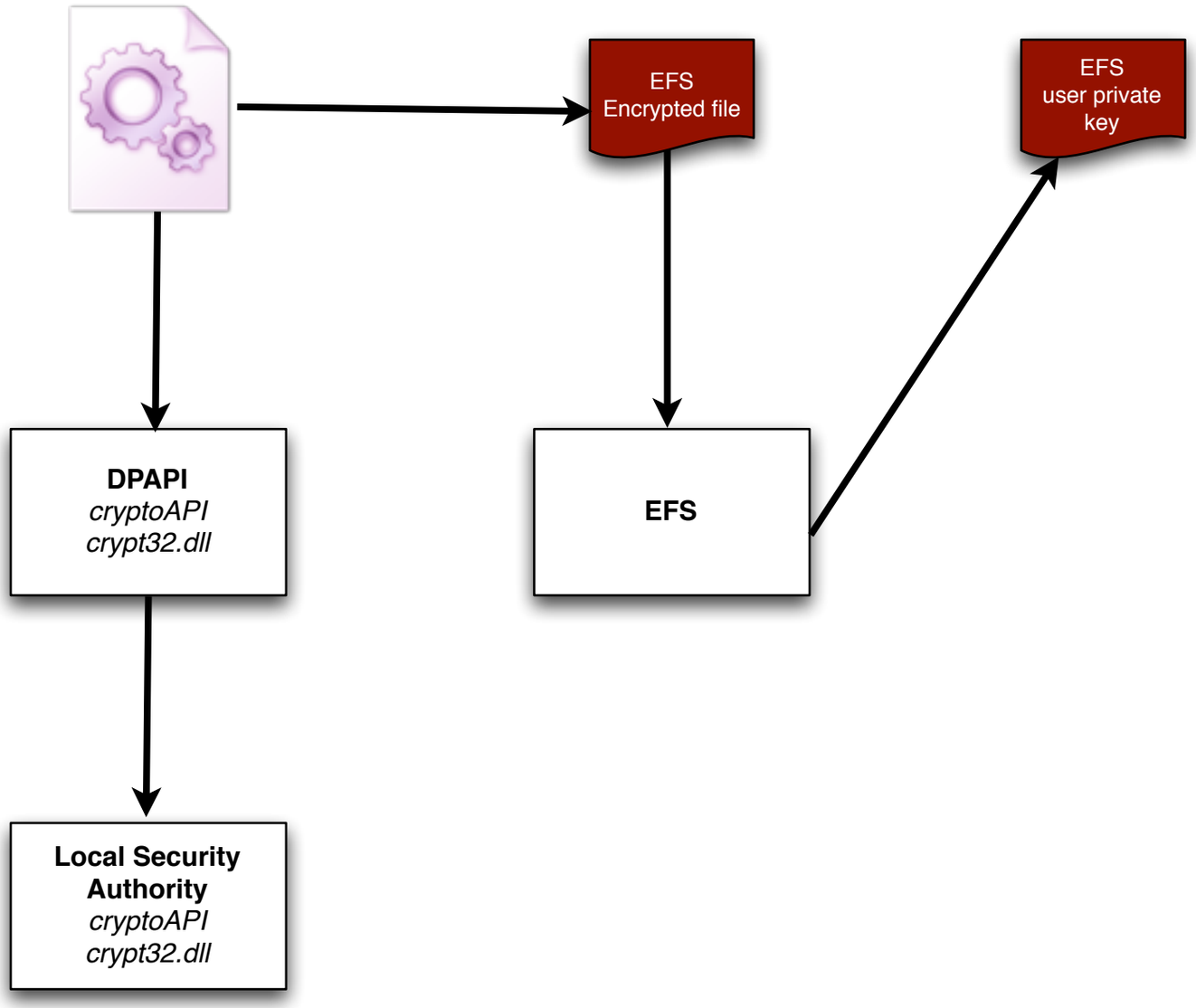
How the system interacts with DPAPI



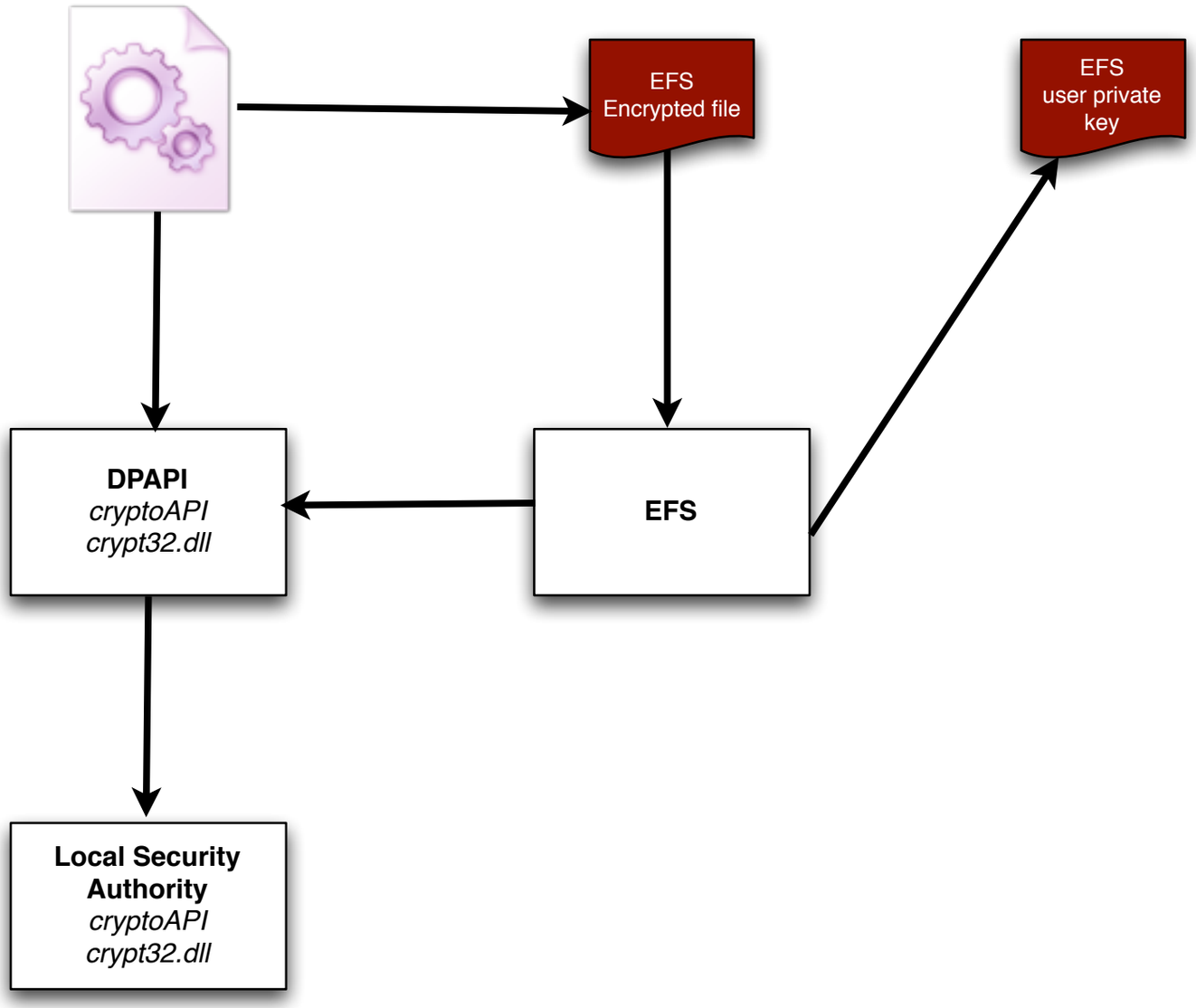
How the system interacts with DPAPI



How the system interacts with DPAPI



How the system interacts with DPAPI



DPAPI CryptUnprotecDData Function

```
BOOL WINAPI CryptUnprotectData (
  *pDataIn,
  *ppszDataDescr,
  *pOptionalEntropy,
  pvReserved,
  *pPromptStruct,
  dwFlags,
  *pDataOut
```

DPAPI CryptUnprotecDataFunction

```
BOOL WINAPI CryptUnprotectData (
  *pDataIn,
  *ppszDataDescr,
  *pOptionalEntropy,
  pvReserved,
  *pPromptStruct,
  dwFlags,
  *pDataOut
```

Encrypted data aka data blob

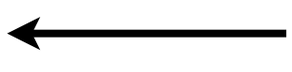
DPAPI CryptUnprotecDataFunction

```
BOOL WINAPI CryptUnprotectData (
  *pDataIn,
  *ppszDataDescr,
  *pOptionalEntropy,
  pvReserved,
  *pPromptStruct,
  dwFlags,
  *pDataOut
```

Optional description

DPAPI CryptUnprotecDData Function

```
BOOL WINAPI CryptUnprotectData (
  *pDataIn,
  *ppszDataDescr,
  *pOptionalEntropy,
  pvReserved,
  *pPromptStruct,
  dwFlags,
  *pDataOut
```



Optional entropy (salt)

DPAPI CryptUnprotecDData Function

```
BOOL WINAPI CryptUnprotectData (
  *pDataIn,
  *ppszDataDescr,
  *pOptionalEntropy,
  pvReserved,
  *pPromptStruct,
  dwFlags,
  *pDataOut
```

Optional password

DPAPI CryptUnprotecDataFunction

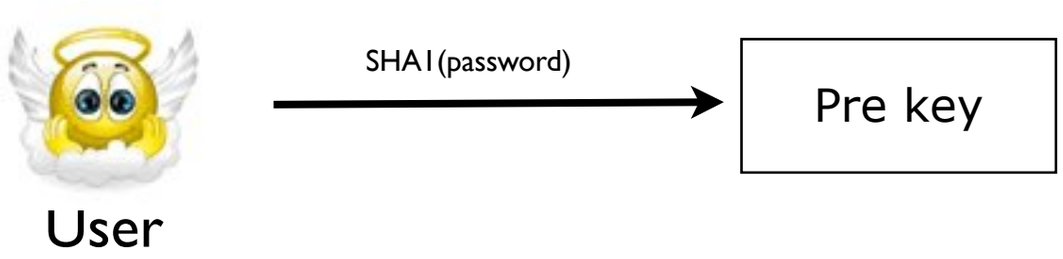
```
BOOL WINAPI CryptUnprotectData (
  *pDataIn,
  *ppszDataDescr,
  *pOptionalEntropy,
  pvReserved,
  *pPromptStruct,
  dwFlags,
  *pDataOut
```

Decrypted data

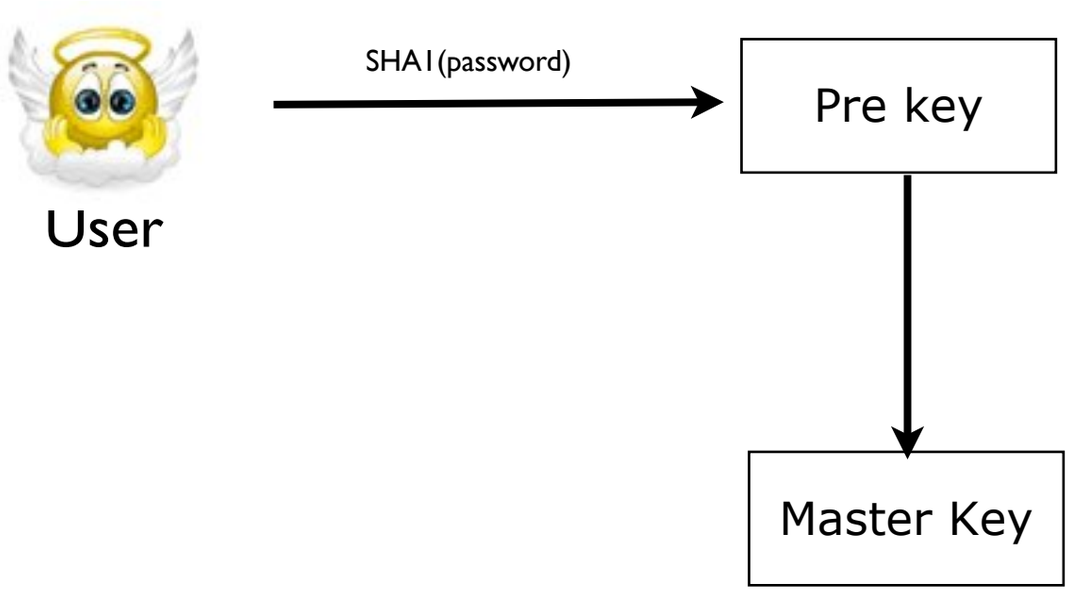
Derivation scheme

User

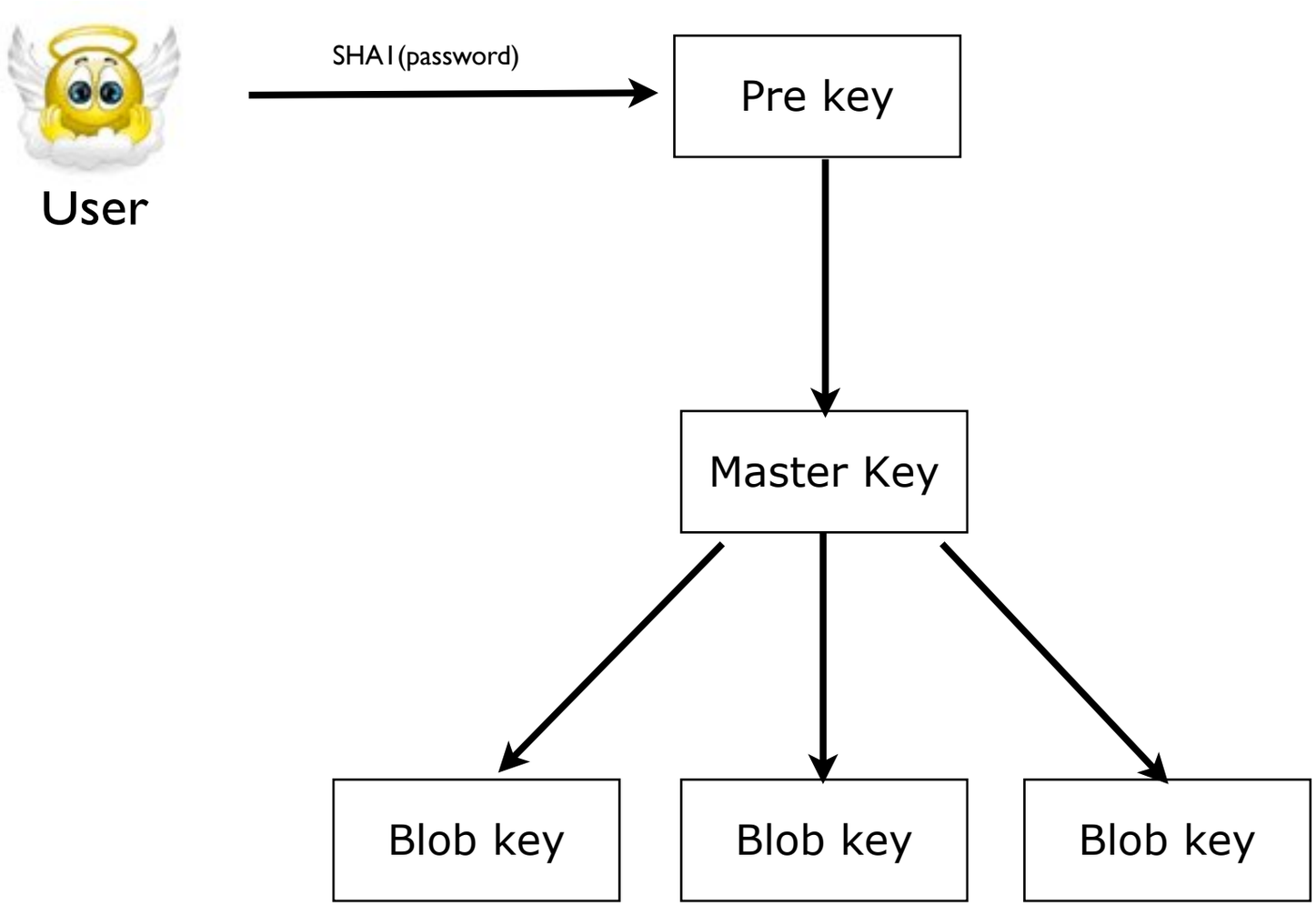
Derivation scheme



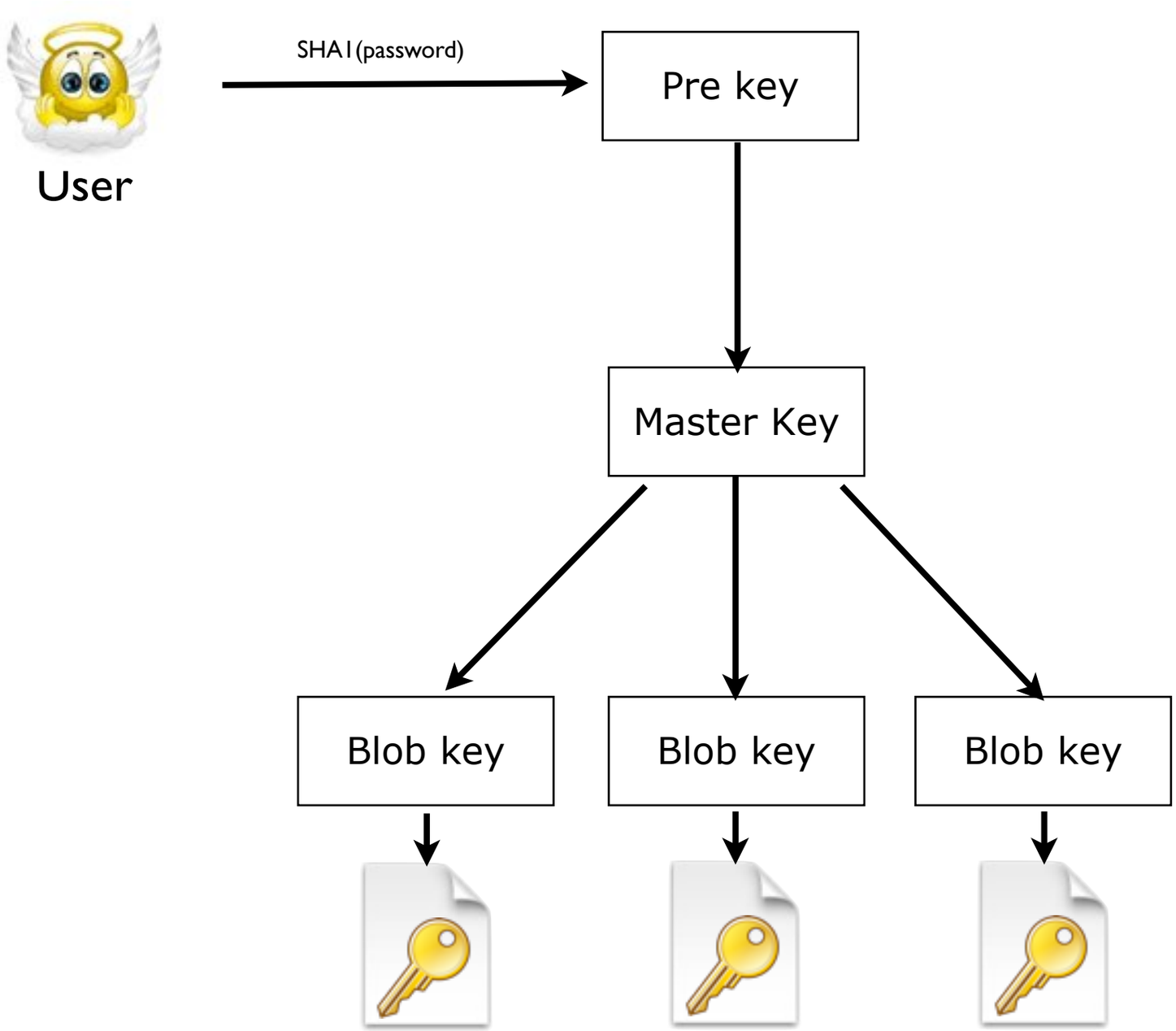
Derivation scheme



Derivation scheme



Derivation scheme



Blob structure

- Returned to the application (opaque structure)
- Store user encrypted data
- Contains decryption parameters

key subtleties

- SHA1 password are in UTF-16LE
- SID for HMAC are also in UTF-16LE (don't forget the \0 !)
- Windows 2000 do not use SHA1/3DES. We think it uses SHA1/RC4 (Anyone want to try ?).

data blob structure key fields

```
DWORD       cbProviders;
GUID        *arrProviders;
DWORD       cbKeys;
GUID        *arrKeys;
WCHAR       *ppszDataDescr;

DWORD       idCipherAlgo;
BYTE        *pbIV;
DWORD       idHashAlgo;
BYTE        *pbSalt;
 BYTE       *pbCipher;
BYTE        *pbHMAC;
```

data blob structure key fields

```
DWORD      cbProviders;
GUID       *arrProviders;
DWORD      cbKeys;
GUID       *arrKeys;
WCHAR      *ppszDataDescr;

DWORD      idCipherAlgo;
BYTE       *pbIV;
DWORD      idHashAlgo;
BYTE       *pbSalt;
 BYTE      *pbCipher;
BYTE       *pbHMAC;
```

Nb of crypto providers

data blob structure key fields

```
DWORD       cbProviders;
GUID        *arrProviders;
DWORD       cbKeys;
GUID        *arrKeys;
WCHAR       *ppszDataDescr;

DWORD       idCipherAlgo;
BYTE        *pbIV;
DWORD       idHashAlgo;
BYTE        *pbSalt;
 BYTE       *pbCipher;
BYTE        *pbHMAC;
```

Crypto providers GUID

data blob structure key fields

```
DWORD       cbProviders;
GUID        *arrProviders;
DWORD       cbKeys;
GUID        *arrKeys;
WCHAR       *ppszDataDescr;

DWORD       idCipherAlgo;
BYTE        *pbIV;
DWORD       idHashAlgo;
BYTE        *pbSalt;
 BYTE       *pbCipher;
BYTE        *pbHMAC;
```

Nb of masters keys

data blob structure key fields

```
DWORD       cbProviders;
GUID        *arrProviders;
DWORD       cbKeys;
GUID        *arrKeys;
WCHAR       *ppszDataDescr;

DWORD       idCipherAlgo;
BYTE        *pbIV;
DWORD       idHashAlgo;
BYTE        *pbSalt;
 BYTE       *pbCipher;
BYTE        *pbHMAC;
```

Masters keys GUID

data blob structure key fields

```
DWORD      cbProviders;
GUID       *arrProviders;
DWORD      cbKeys;
GUID       *arrKeys;
WCHAR      *ppszDataDescr;
DWORD      idCipherAlgo;
BYTE       *pbIV;
DWORD      idHashAlgo;
BYTE       *pbSalt;
BYTE       *pbCipher;
BYTE       *pbHMAC;
```

Optional description

data blob structure key fields

```
DWORD       cbProviders;
GUID        *arrProviders;
DWORD       cbKeys;
GUID        *arrKeys;
WCHAR       *ppszDataDescr;
DWORD       idCipherAlgo;
BYTE        *pbIV;
DWORD       idHashAlgo;
BYTE        *pbSalt;
BYTE        *pbCipher;
BYTE        *pbHMAC;
```

Encryption algorithm ID

data blob structure key fields

```
DWORD       cbProviders;
GUID        *arrProviders;
DWORD       cbKeys;
GUID        *arrKeys;
WCHAR       *ppszDataDescr;

DWORD       idCipherAlgo;
BYTE        *pbIV;
DWORD       idHashAlgo;
BYTE        *pbSalt;
 BYTE       *pbCipher;
BYTE        *pbHMAC;
```

Initialization vector

data blob structure key fields

```
DWORD       cbProviders;
GUID        *arrProviders;
DWORD       cbKeys;
GUID        *arrKeys;
WCHAR       *ppszDataDescr;

DWORD       idCipherAlgo;
BYTE        *pbIV;
DWORD       idHashAlgo;
BYTE        *pbSalt;
 BYTE       *pbCipher;
BYTE        *pbHMAC;
```

Hash algorithm ID

data blob structure key fields

```
DWORD       cbProviders;
GUID        *arrProviders;
DWORD       cbKeys;
GUID        *arrKeys;
WCHAR       *ppszDataDescr;

DWORD       idCipherAlgo;
BYTE        *pbIV;
DWORD       idHashAlgo;
BYTE        *pbSalt;
BYTE        *pbCipher;
BYTE        *pbHMAC;
```

← Salt generated by DPAPI

data blob structure key fields

```
DWORD       cbProviders;
GUID        *arrProviders;
DWORD       cbKeys;
GUID        *arrKeys;
WCHAR       *ppszDataDescr;

DWORD       idCipherAlgo;
BYTE        *pbIV;
DWORD       idHashAlgo;
BYTE        *pbSalt;
BYTE        *pbCipher;
BYTE        *pbHMAC;
```

Encrypted data

data blob structure key fields

```
DWORD       cbProviders;
GUID        *arrProviders;
DWORD       cbKeys;
GUID        *arrKeys;
WCHAR       *ppszDataDescr;

DWORD       idCipherAlgo;
BYTE        *pbIV;
DWORD       idHashAlgo;
BYTE        *pbSalt;
 BYTE       *pbCipher;
BYTE        *pbHMAC;
```

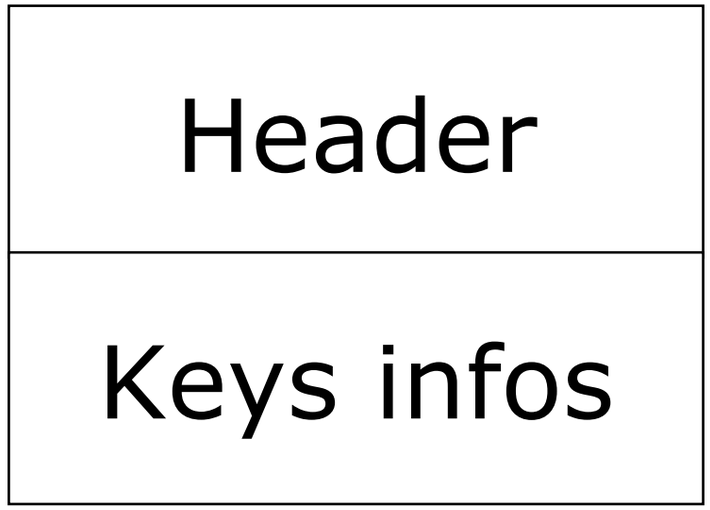
Blob HMAC

Master key structure

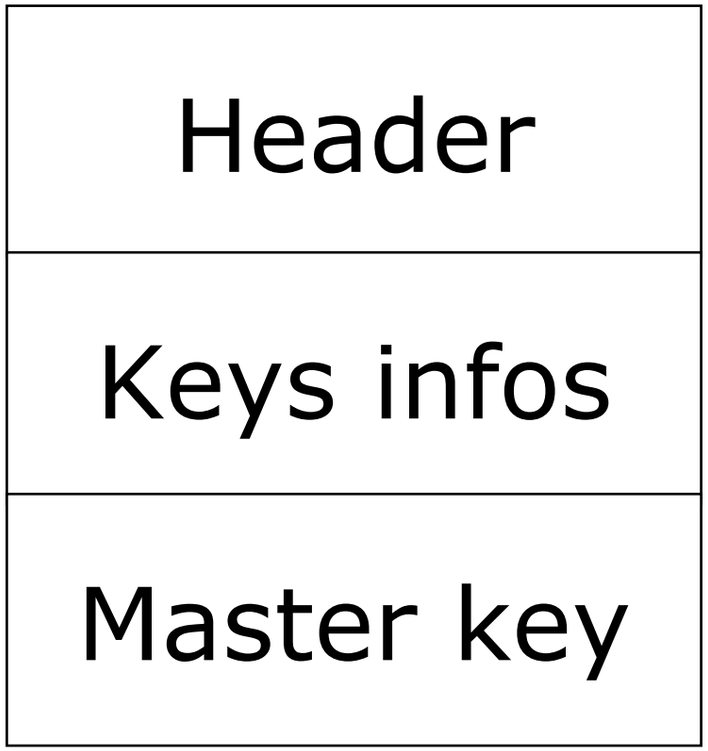
- Store the key used to decrypt blob
- Encrypted with the user password
- Renewed every 3 months

The master key file

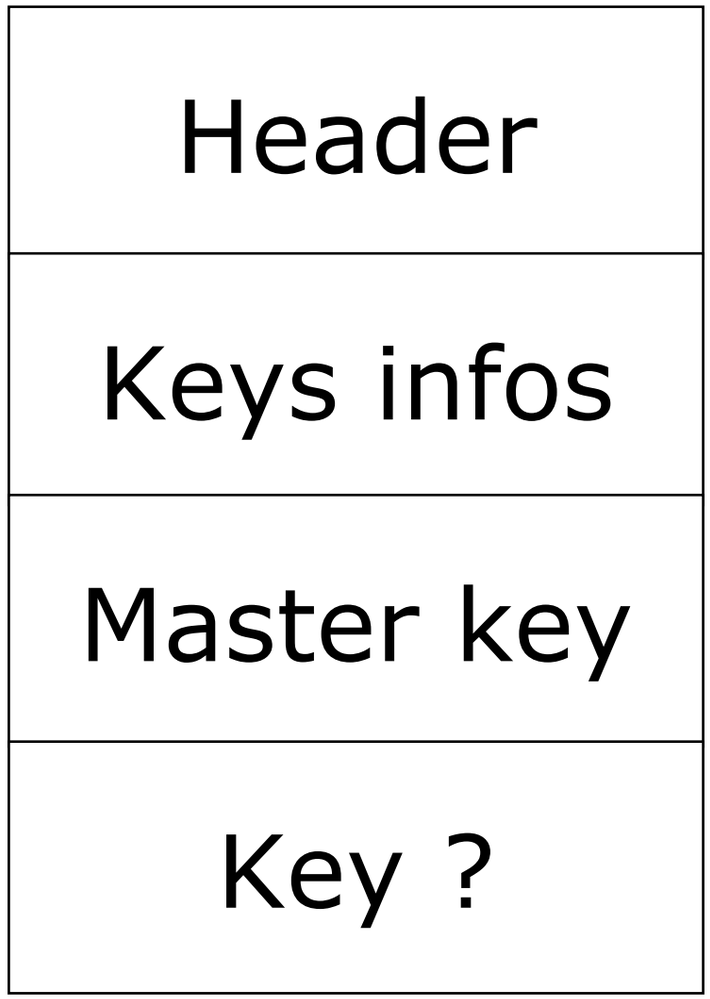
The master key file



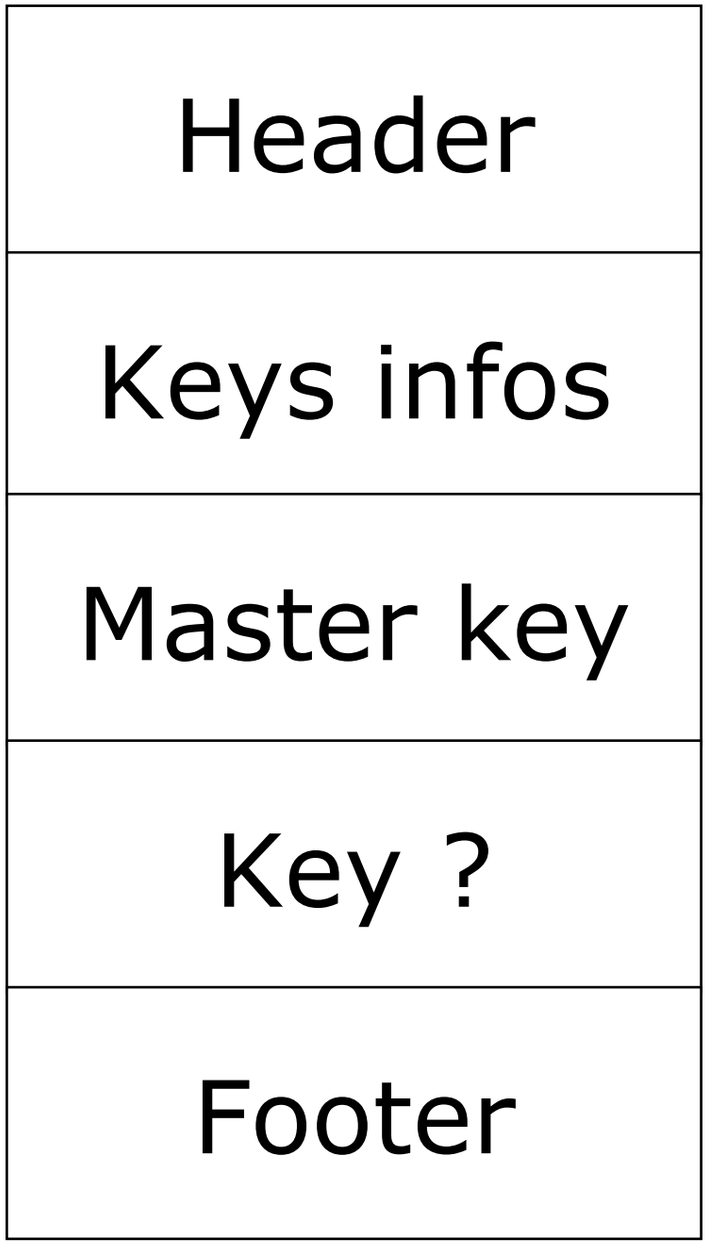
The master key file



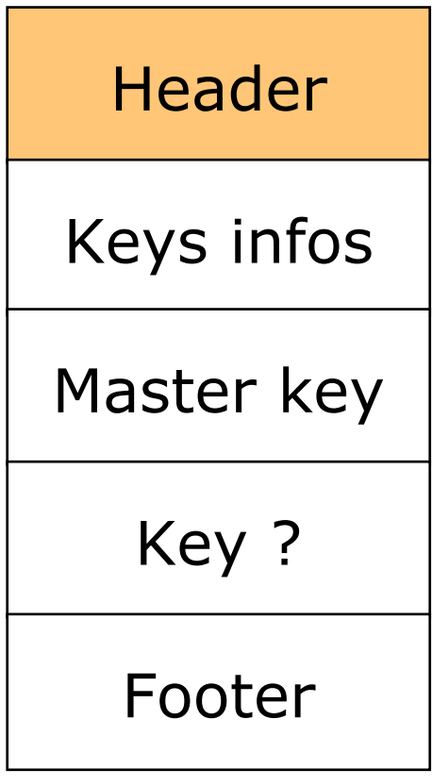
The master key file



The master key file



Header structure



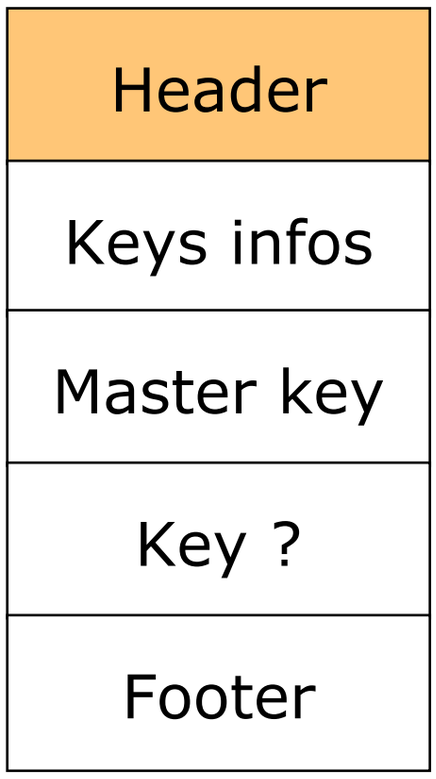
dwVersion;

nullPad1;

szKeyGUID[36];

nullPad2;

Header structure



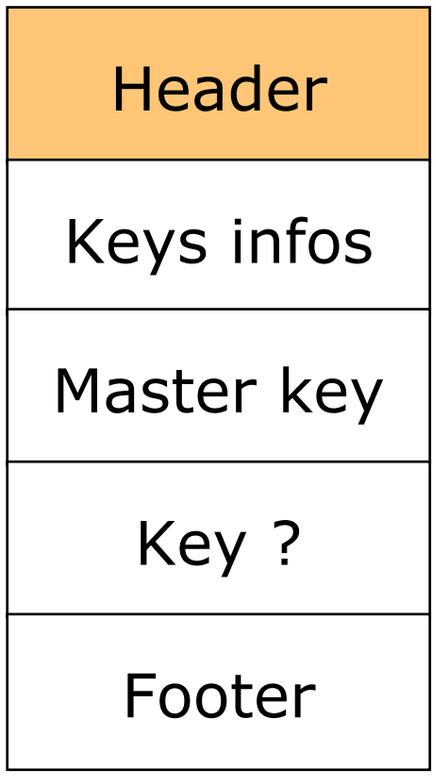
dwVersion; ← File version

nullPad1;

szKeyGUID[36];

nullPad2;

Header structure



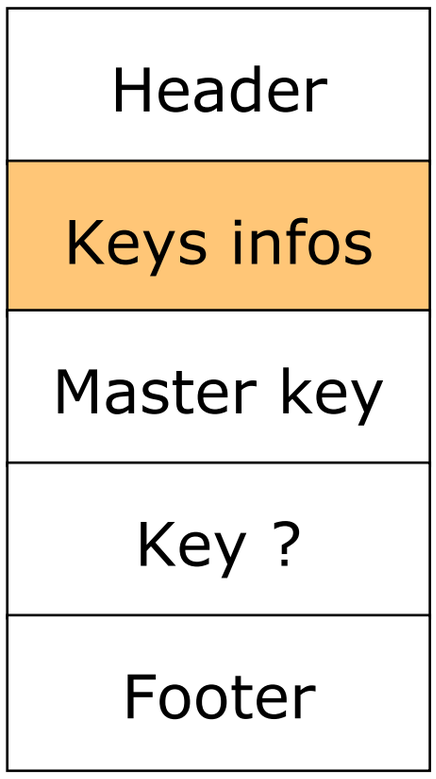
dwVersion;

nullPad1;

szKeyGUID[36]; ← Master key GUID

nullPad2;

Key infos structure



dwUnknown;

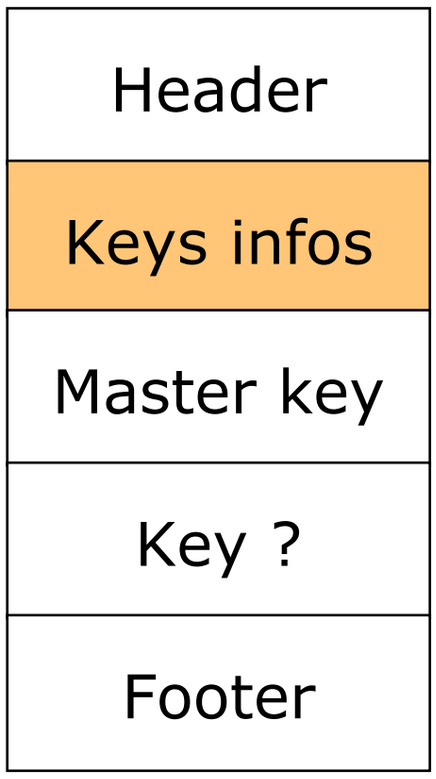
cbMasterKey;

cbMysteryKey;

dwHMACLen;

nullPad3;

Key infos structure



dwUnknown;

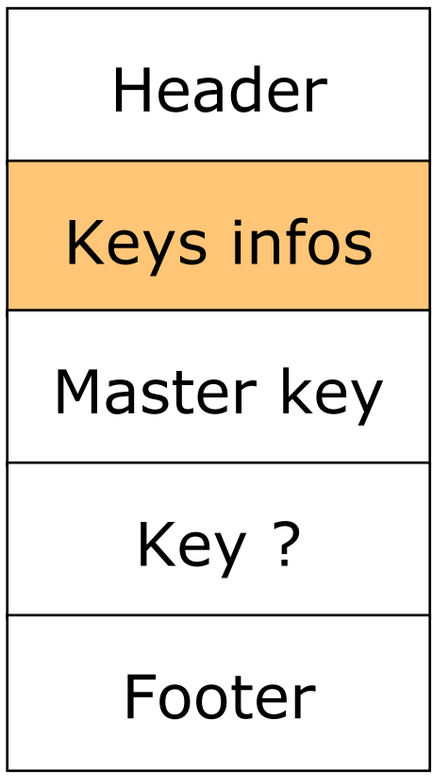
cbMasterKey; ← Master Key struct length

cbMysteryKey;

dwHMACLen;

nullPad3;

Key infos structure



dwUnknown;

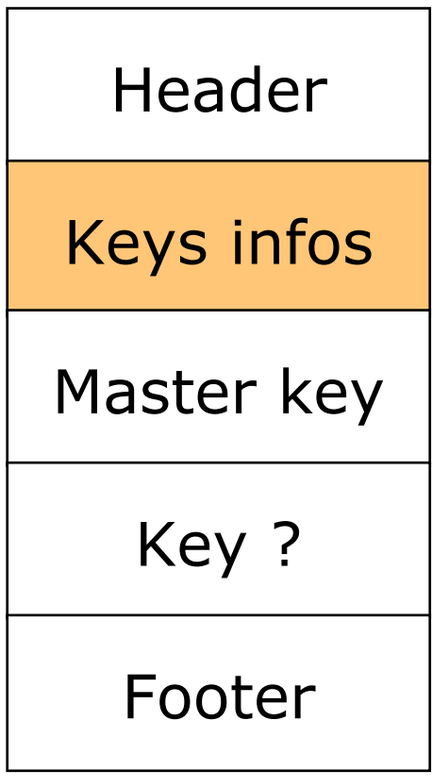
cbMasterKey;

cbMysteryKey; ← Key ? struct length

dwHMACLen;

nullPad3;

Key infos structure



dwUnknown;

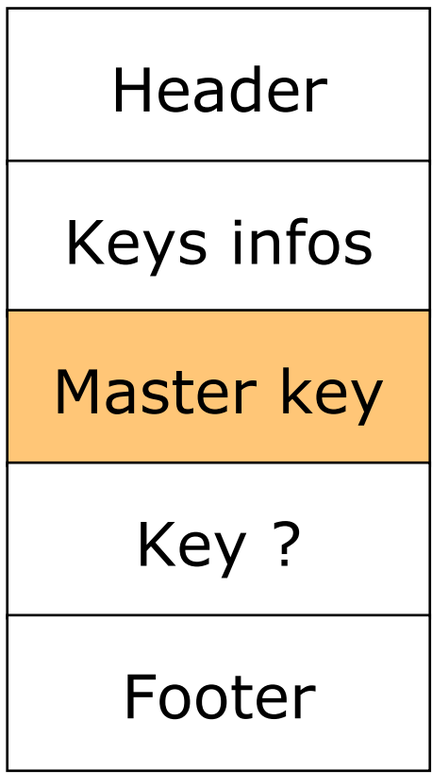
cbMasterKey;

cbMysteryKey;

dwHMACLen; ← HMAC length

nullPad3;

Master key structure



dwMagic;

pbSalt[16];

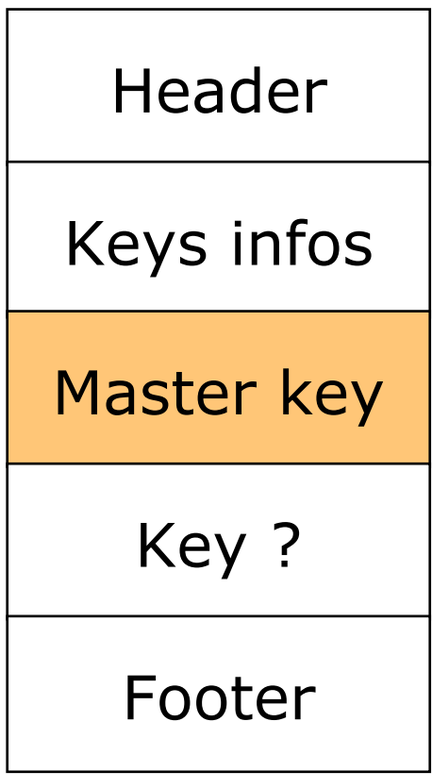
cbIteration;

idMACAlgo;

idCipherAlgo;

pbCipheredKey[];

Master key structure



dwMagic;

pbSalt[16]; ← Key salt

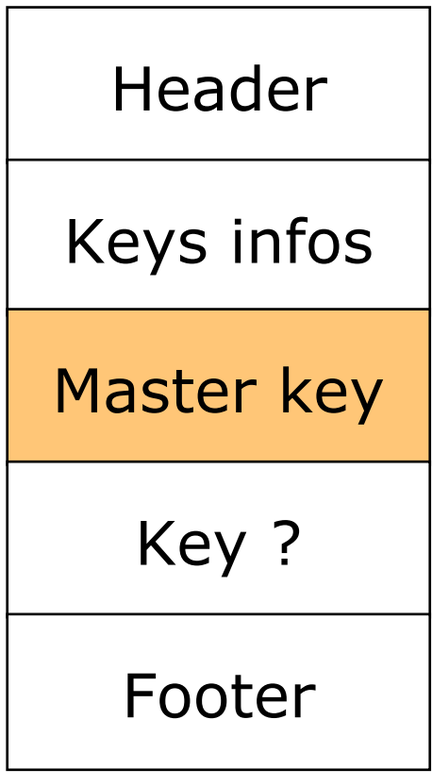
cbIteration;

idMACAlgo;

idCipherAlgo;

pbCipheredKey[];

Master key structure



dwMagic;

pbSalt[16];

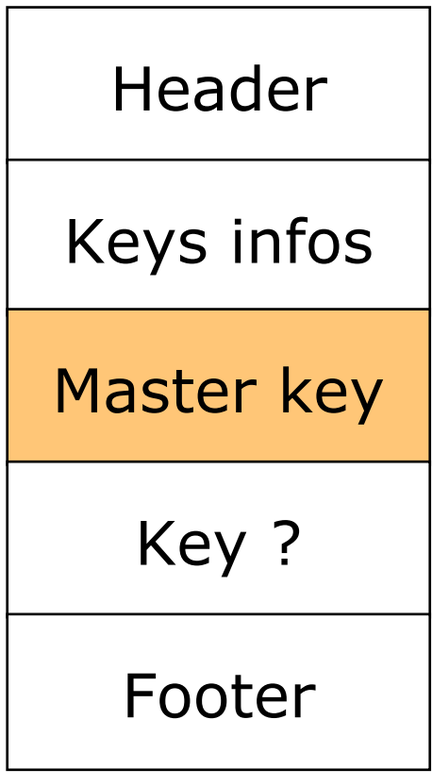
cbIteration; ← PBKDF2 nb rounds

idMACAlgo;

idCipherAlgo;

pbCipheredKey[];

Master key structure



dwMagic;

pbSalt[16];

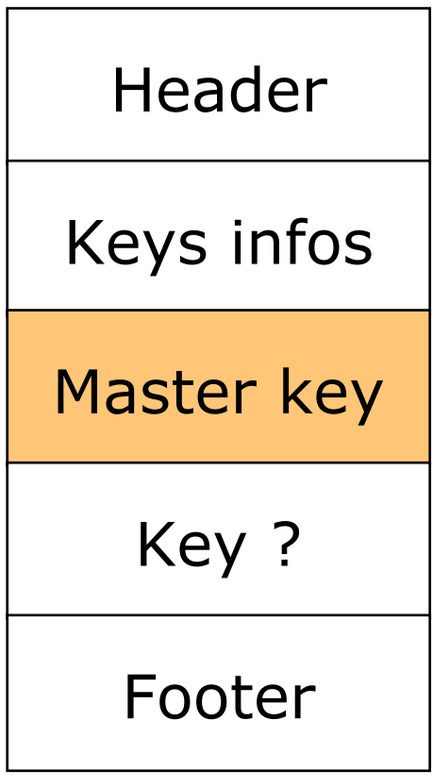
cbIteration;

idMACAlgo; ← HMAC algorithm ID

idCipherAlgo;

pbCipheredKey[];

Master key structure



dwMagic;

pbSalt[16];

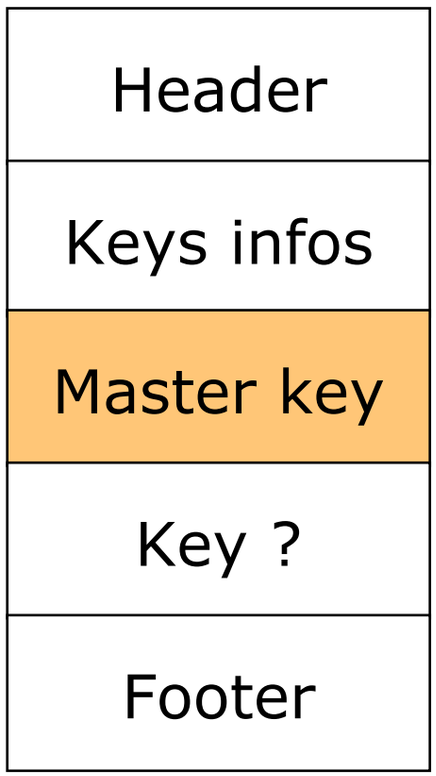
cbIteration;

idMACAlgo;

idCipherAlgo; ← Encryption Algo id

pbCipheredKey[];

Master key structure



dwMagic;

pbSalt[16];

cbIteration;

idMACAlgo;

idCipherAlgo;

pbCipheredKey[]; ← Encrypted key

Decrypting the Master key

DPAPIDecryptKey(sha1, encKey) {

tmp-key = HMAC(sha1, SID)

pre-key = PBKDF2(decryptKey, Salt, ID_ALGO, nblteration)

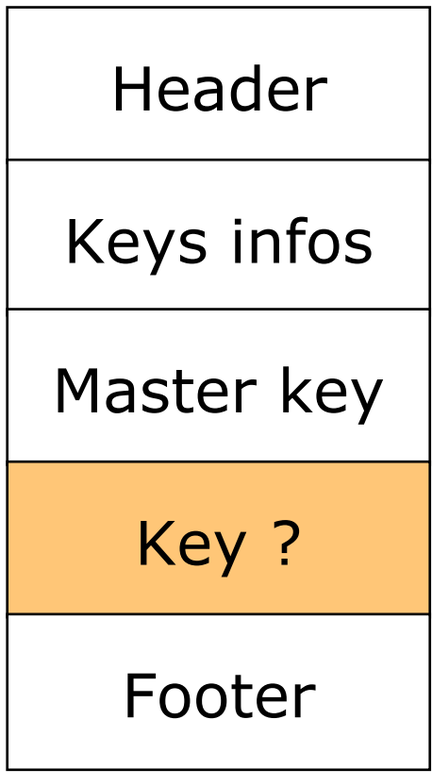
3desKey = pre-key[0 - 23]

3desIV = [24 - 31]

(hmac[0-35], DWORD[36-39], master-key [40-104]) = 3des-cbc(3desKey, iv, encKey)

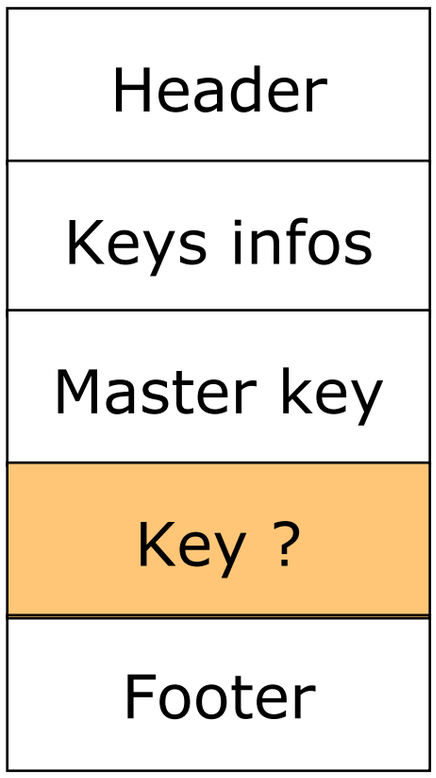
}

key structure



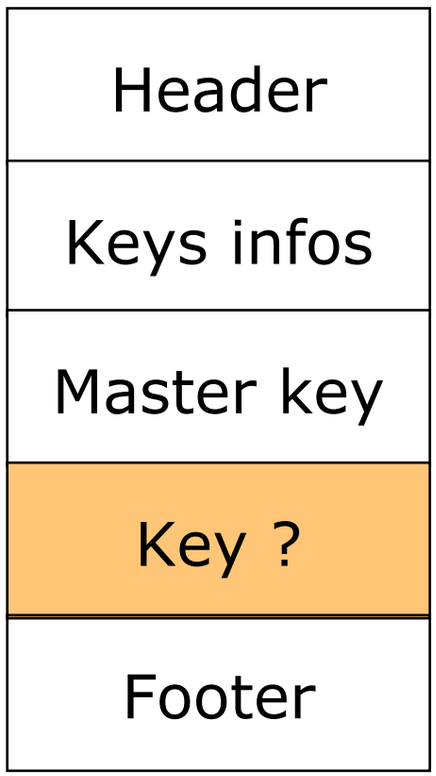
- Seems to have the same structure than the master key
- One round of derivation (XP not Seven)
- 256 bits (half size of the real master-key)

Possible explanation



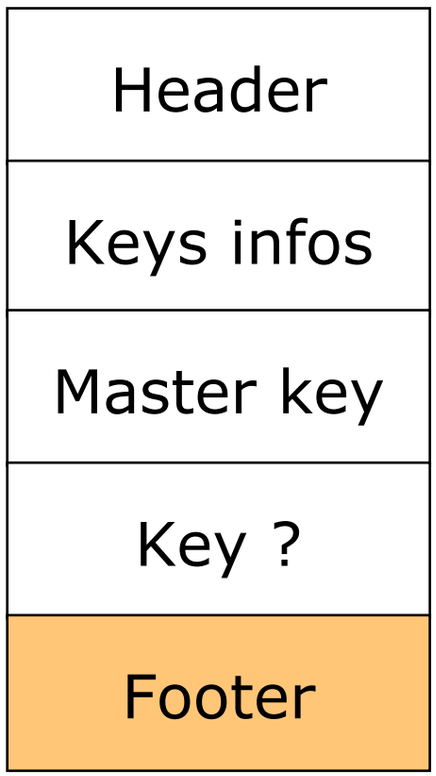
- The documentation state a compatibility mode for windows 2000 exist.
- The registry key to trigger it is unknown
- If we are correct and W2k uses RC4 then the mystery key is possibly a RC4 key (256bits is the correct size).
- PBKDF2 used to compute the IV ??

Possible explanation continued



- We know that RC4 have a weak key scheduling algorithm (remember WEP ?)
- Might be a potential weakness (or not)

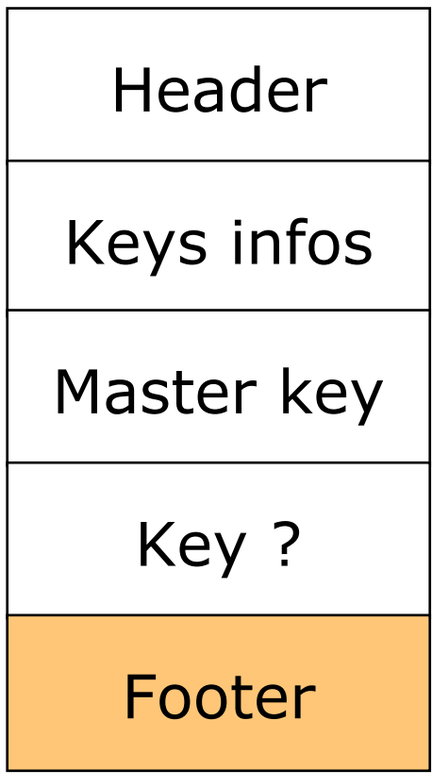
Header structure



dwMagic;

credHist[16];

Header structure



dwMagic;

credHist[16]; ← Password GUID

Differences between windows version

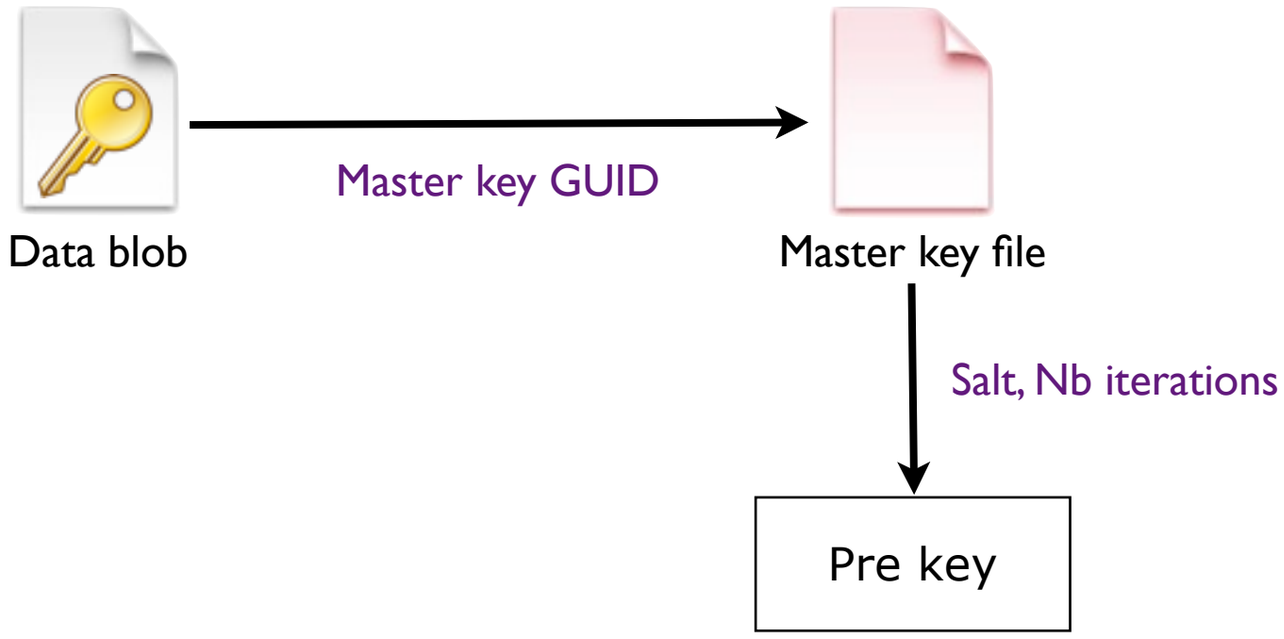
	XP	Vista	Seven
PBKDF2 rounds	4000	24000	Variable (factor ?)
Symmetric algorithm	3DES	3DES	AES
Hash algorithm	SHA1	SHA1	SHA512

Decrypting a blob

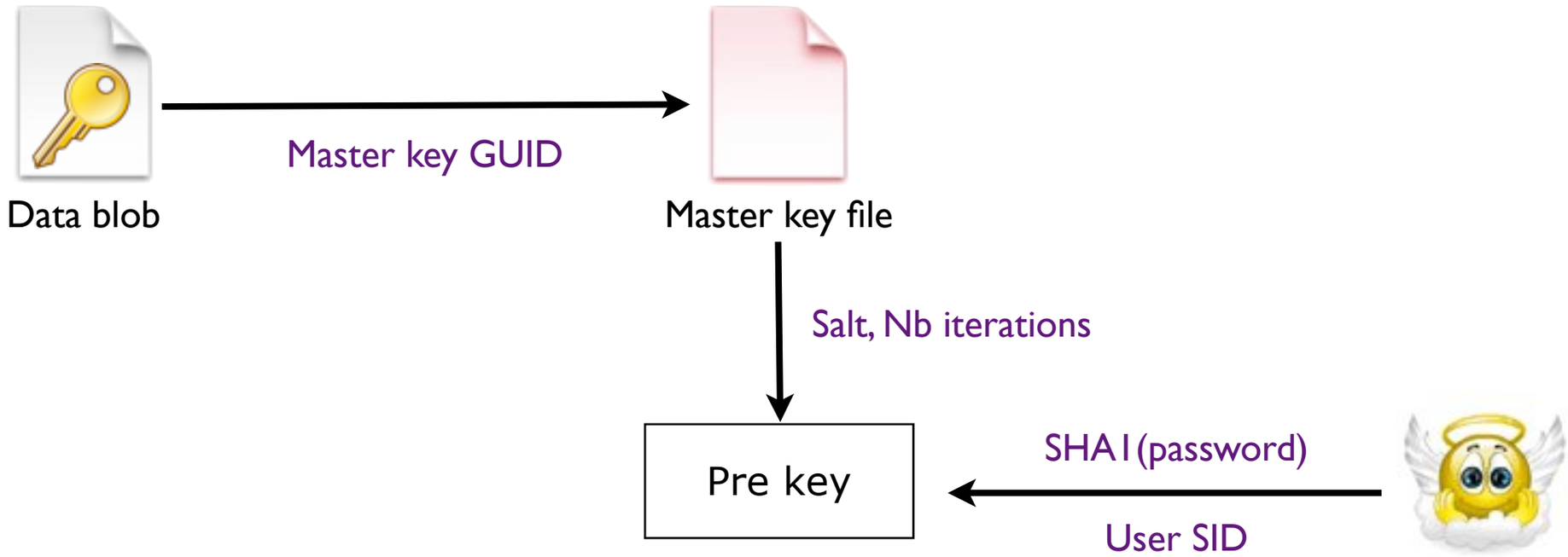
Data blob

Decrypting a blob

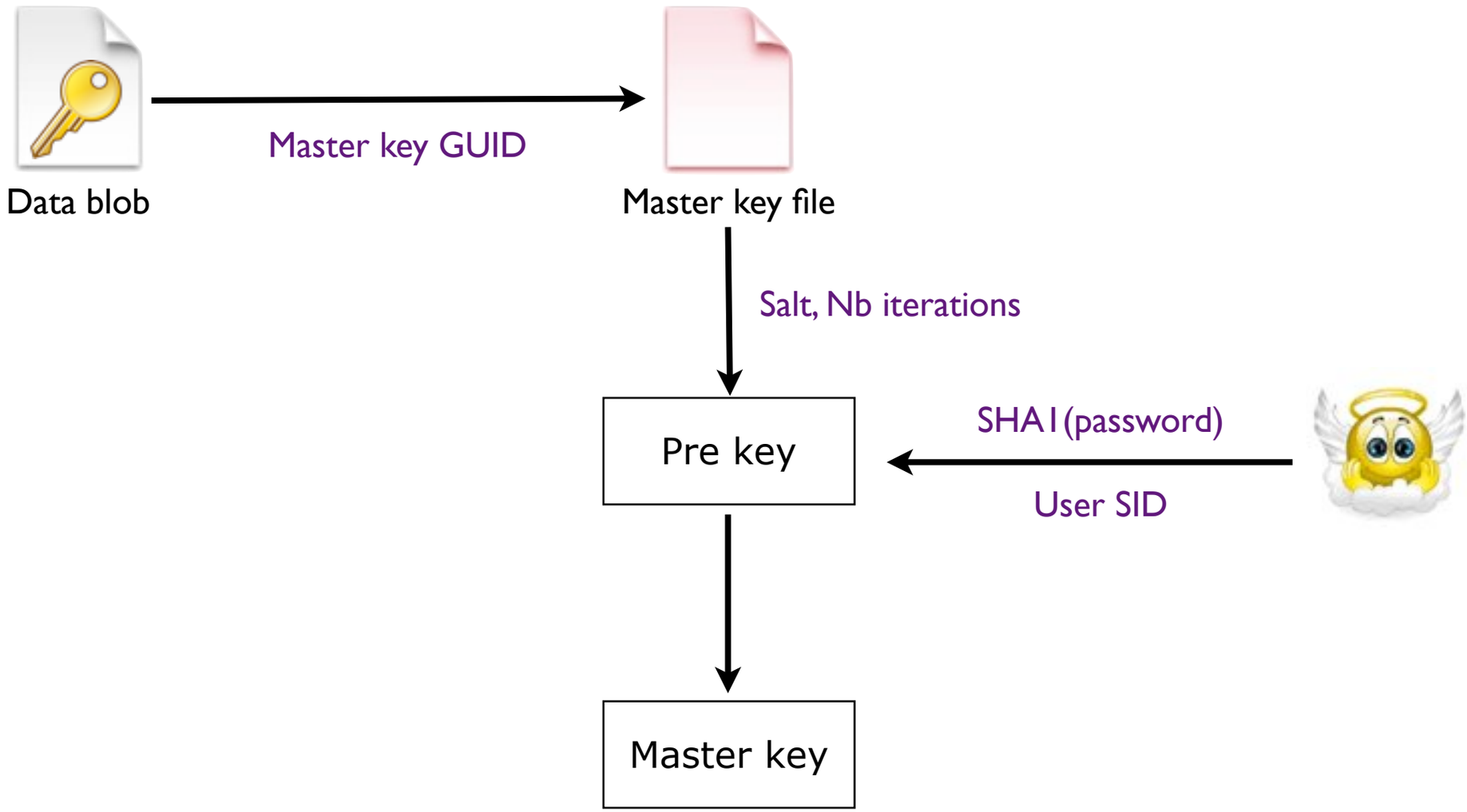
Decrypting a blob



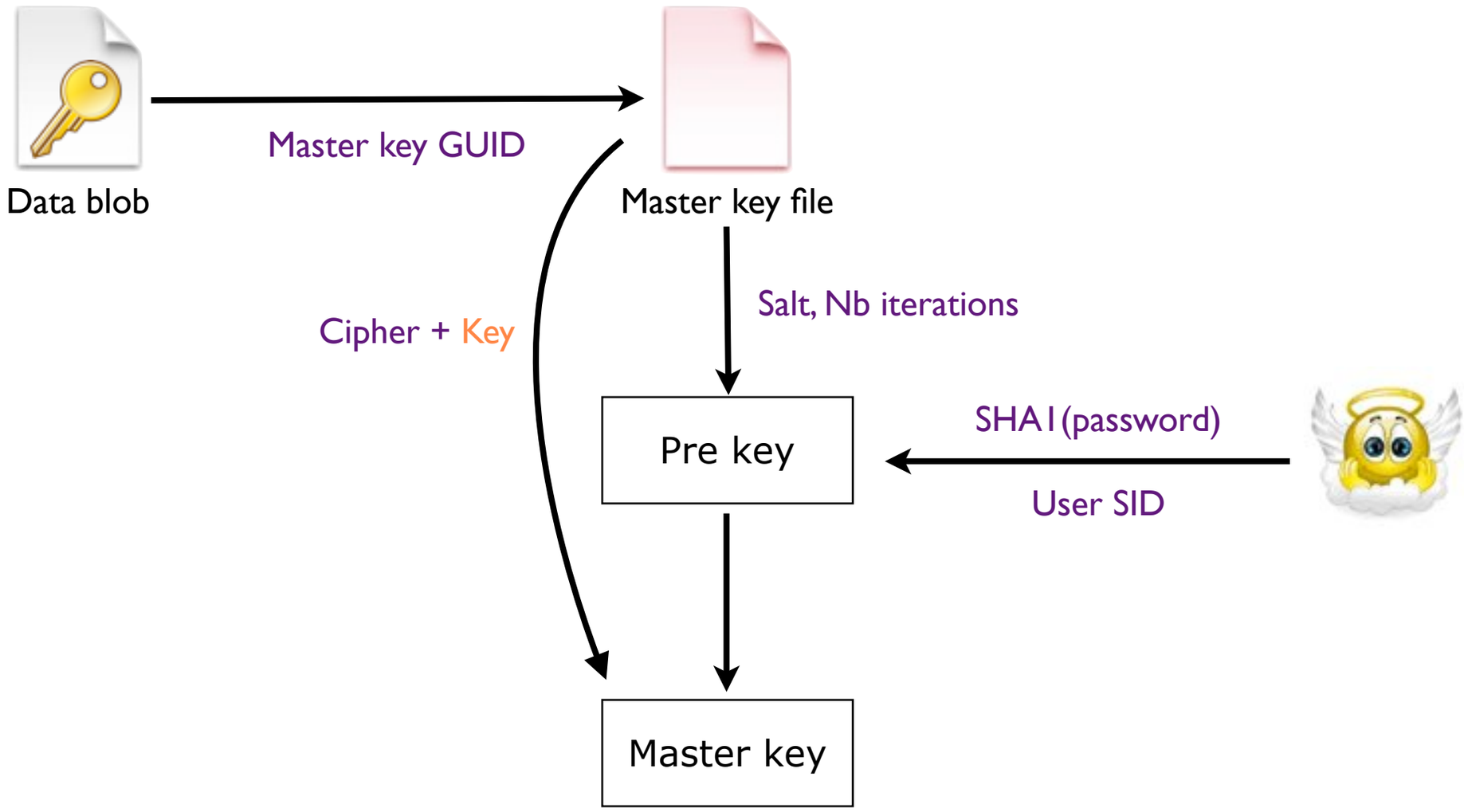
Decrypting a blob



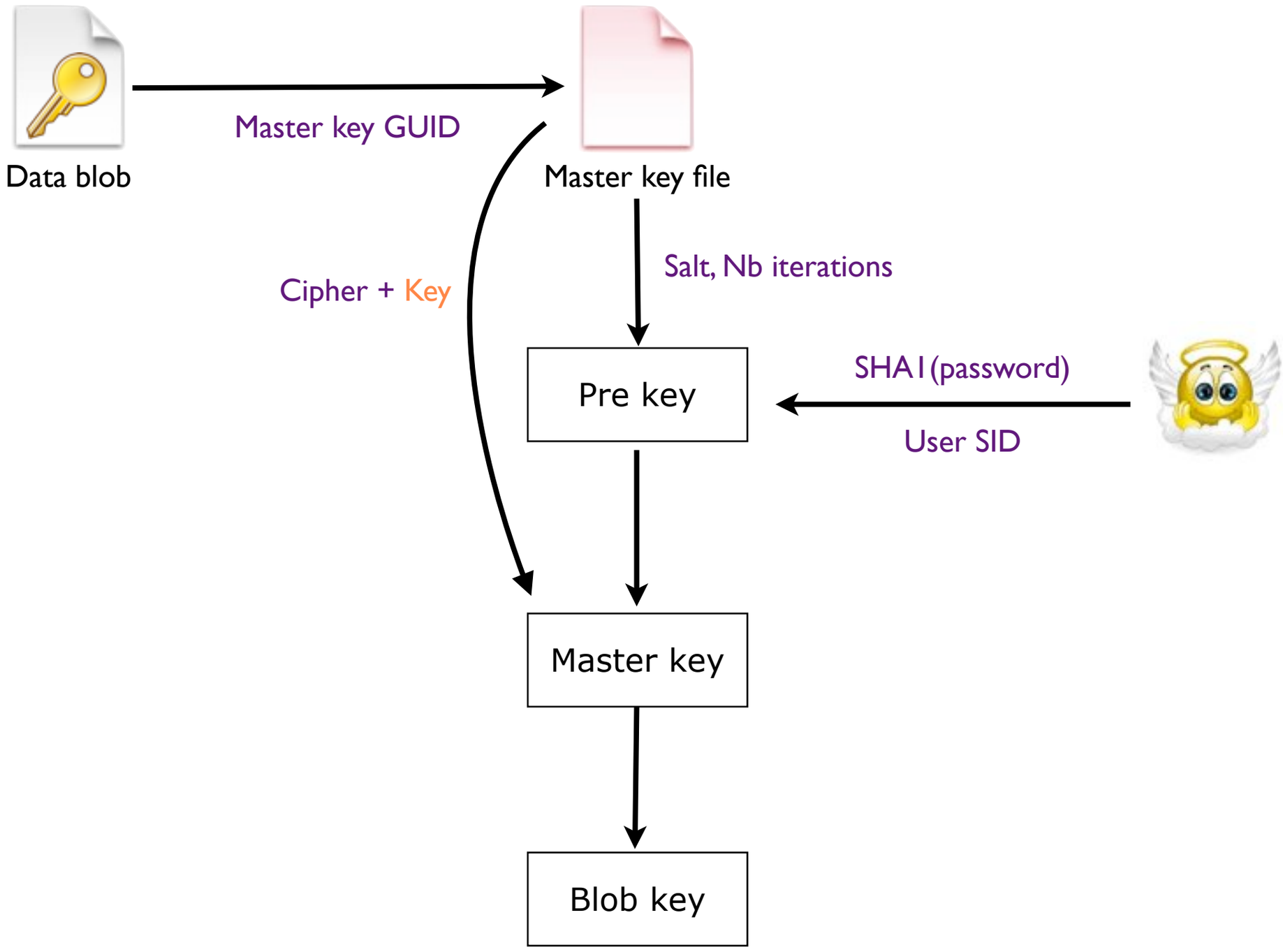
Decrypting a blob



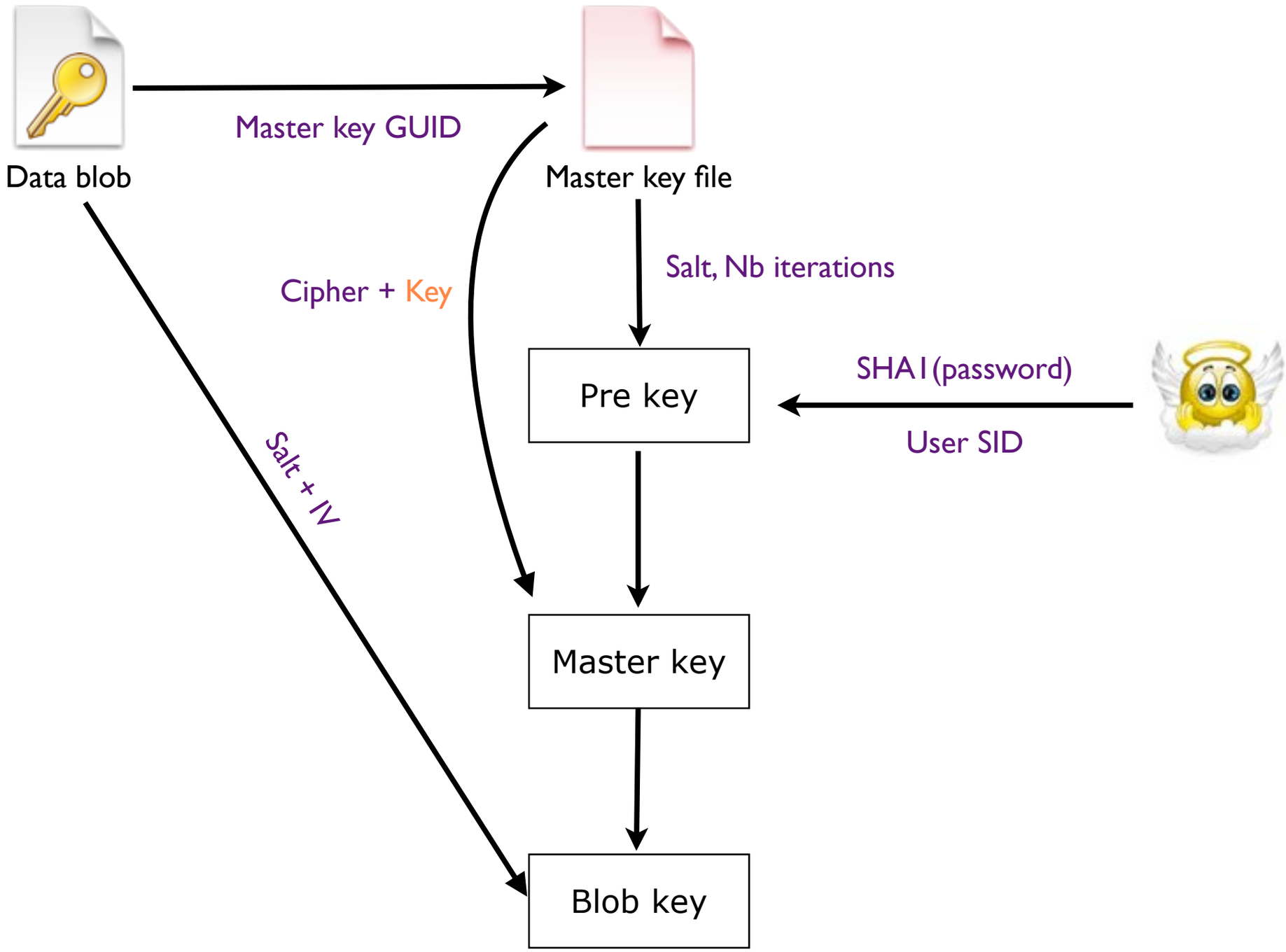
Decrypting a blob



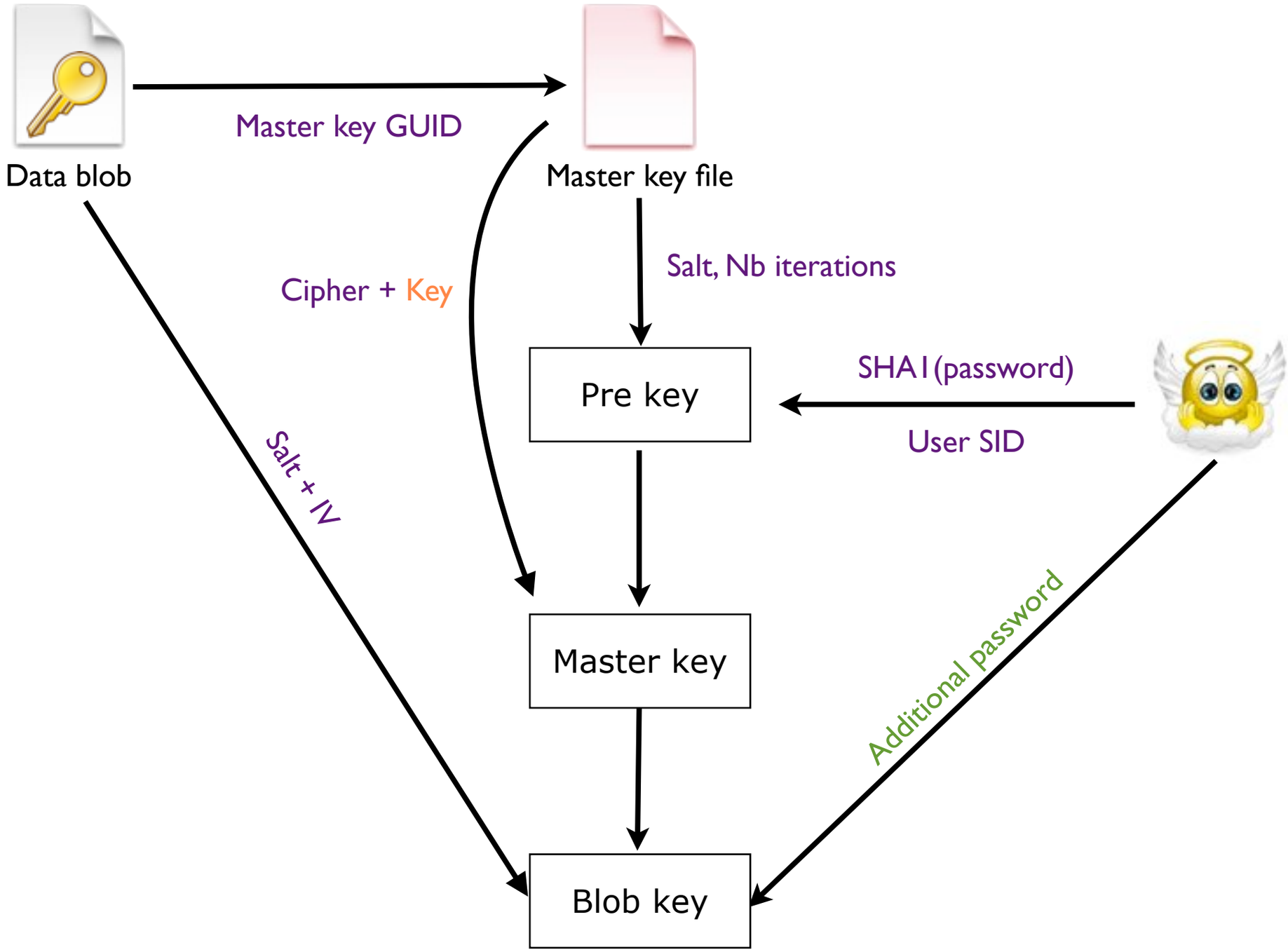
Decrypting a blob



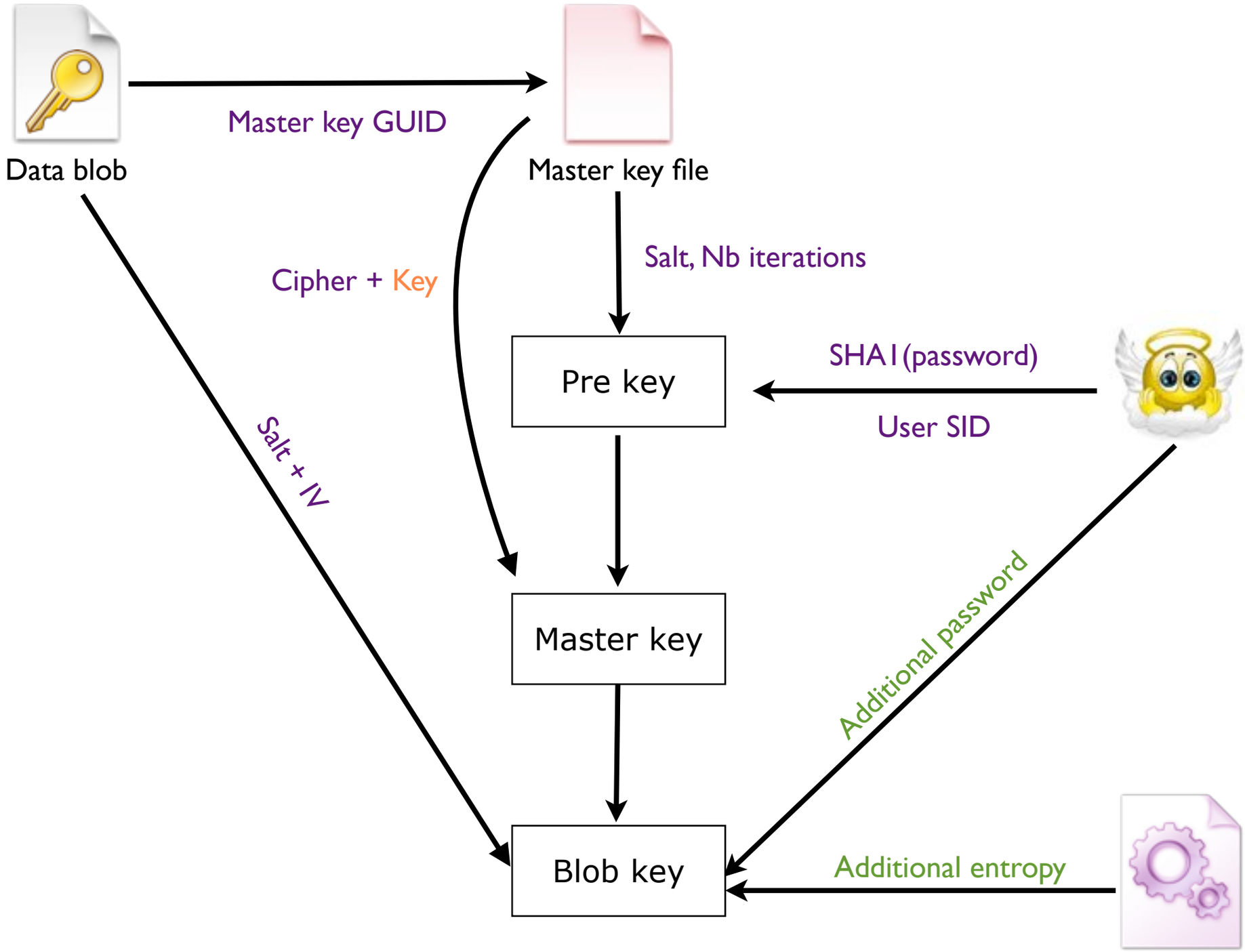
Decrypting a blob



Decrypting a blob



Decrypting a blob



Did I miss something ?

- How the OS knows the current master key ?
- How the OS decides to renew the master key ?
- What happen when the user changes his password ?

Key renewal process

- Renewed every 3 months automatically
  - Passive process: executed when CryptProtect called
- Hardcoded limit (location unknown)
  - Possibly in psbase.dll (MS crypto provider)
  - Can be reduced by using registry override

Master key selection

- All master keys are kept because Windows can't tell if a key is still used
- Keys are stored in %APPDATA%/Microsoft/Protect/[SID]
- Current master key is specified in the Preferred file

The Preferred file

- Simply contains :

  “GUID master key” . “timestamp”

- The key is renewed when

  current time > timestamp

The Preferred file

- Simply contains :

  “GUID master key” . “timestamp”

- The key is renewed when

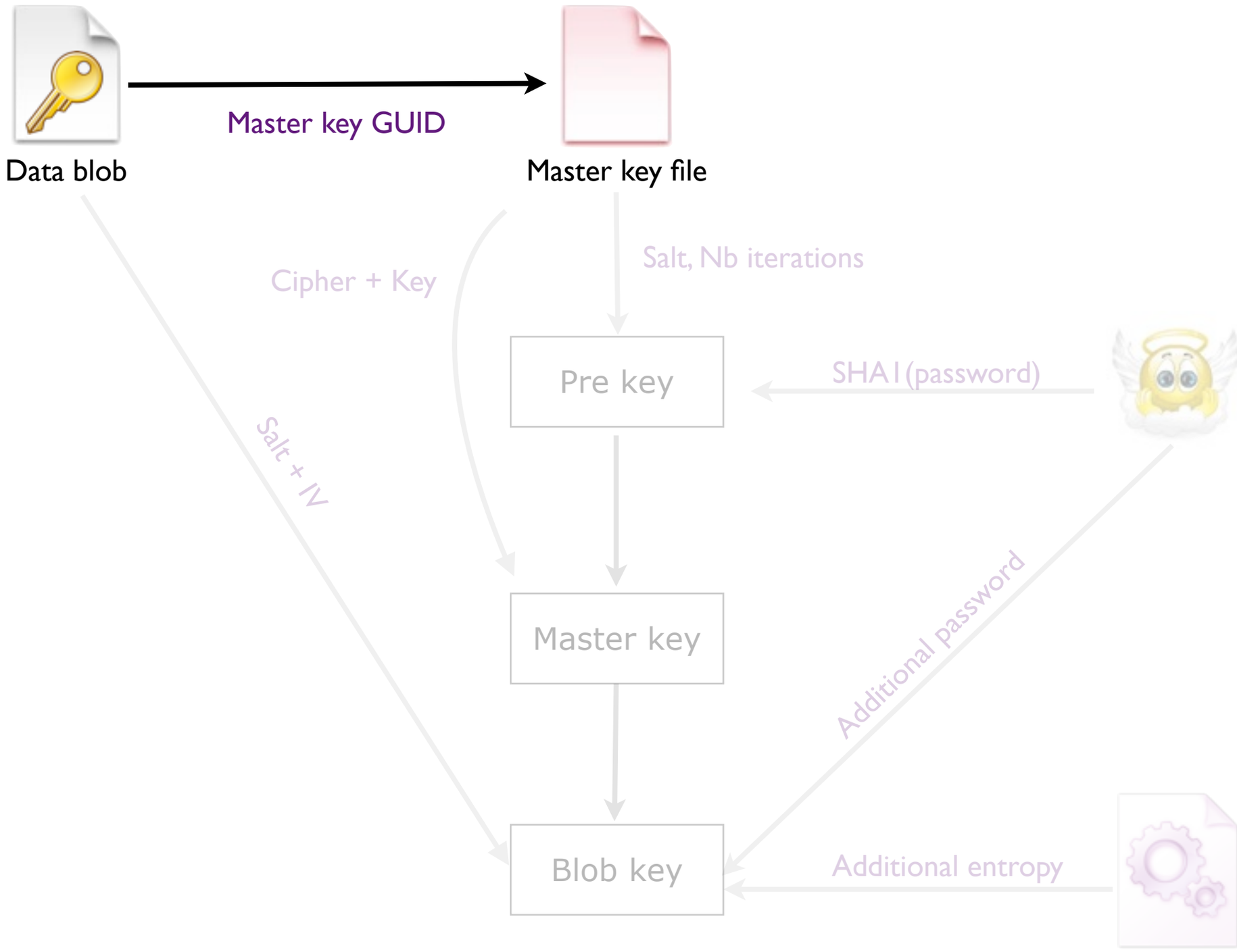
  current time > timestamp

➡ Key escrow attack : Plant a key and update the Preferred file every 3 months (e.g using the task scheduler)

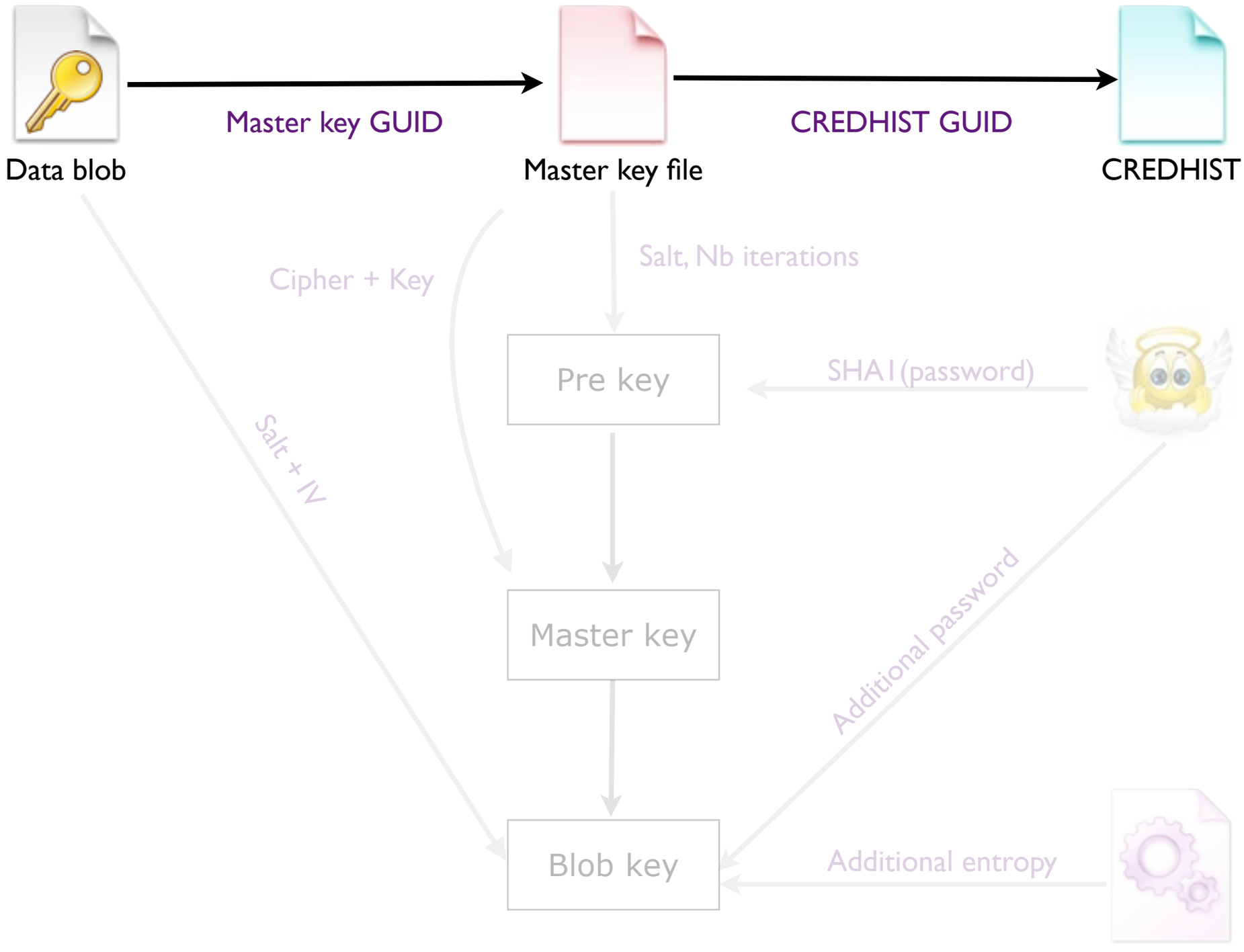
User password renewal

- Master keys are re-encrypted when the password change
- Experimentally not all of them, just the last few ones

Decrypting a blob



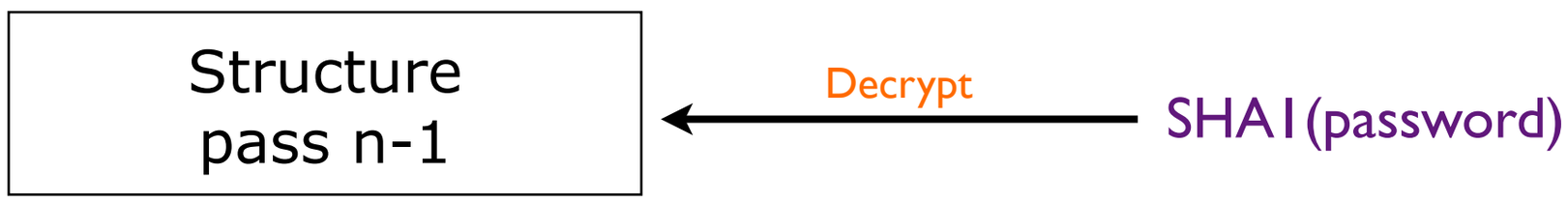
Decrypting a blob



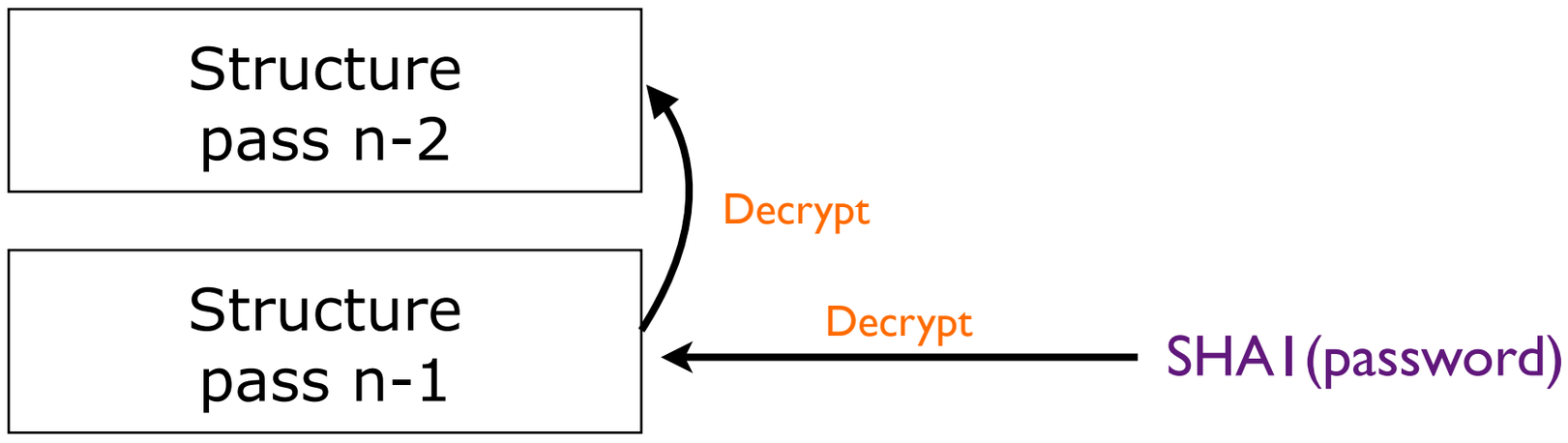
CREDHIST overview

SHA1(password)

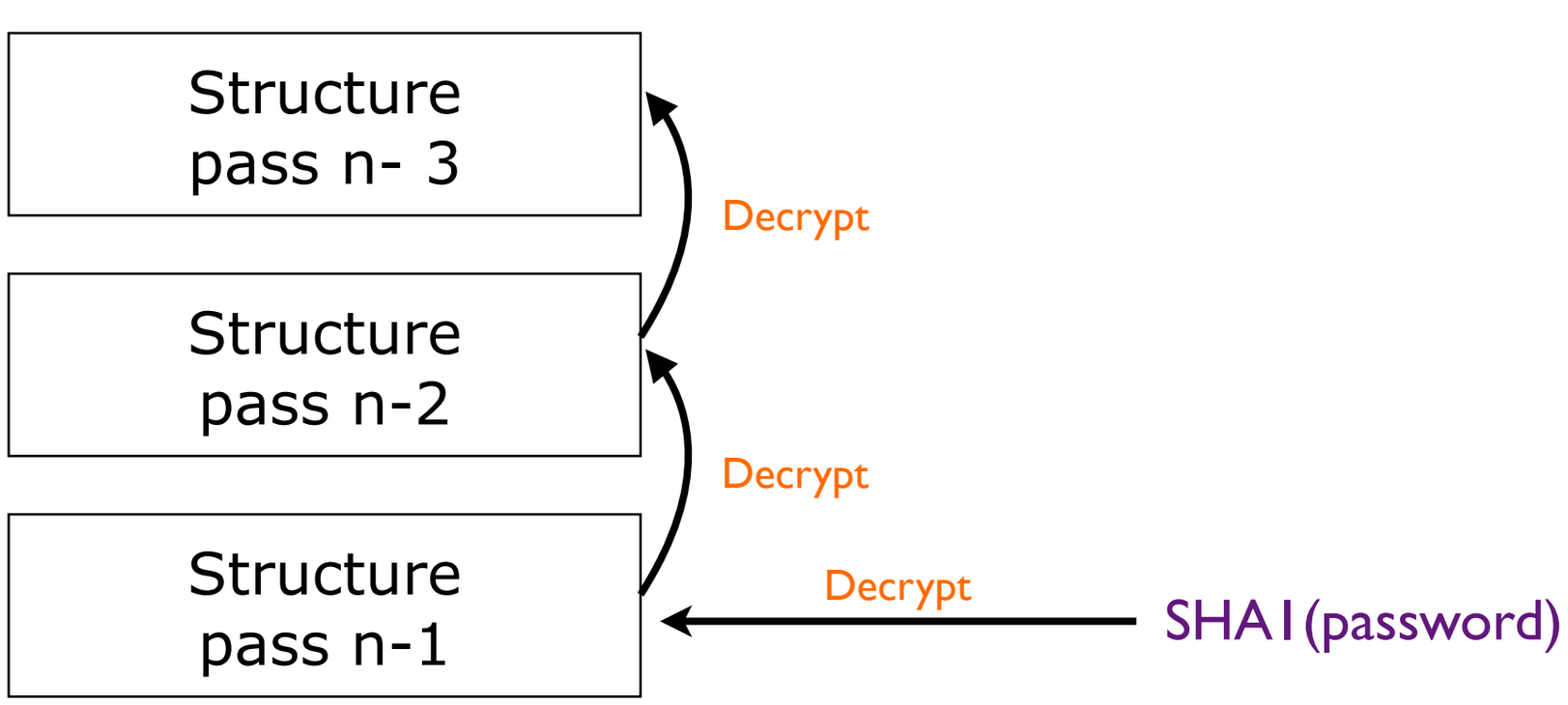
CREDHIST overview



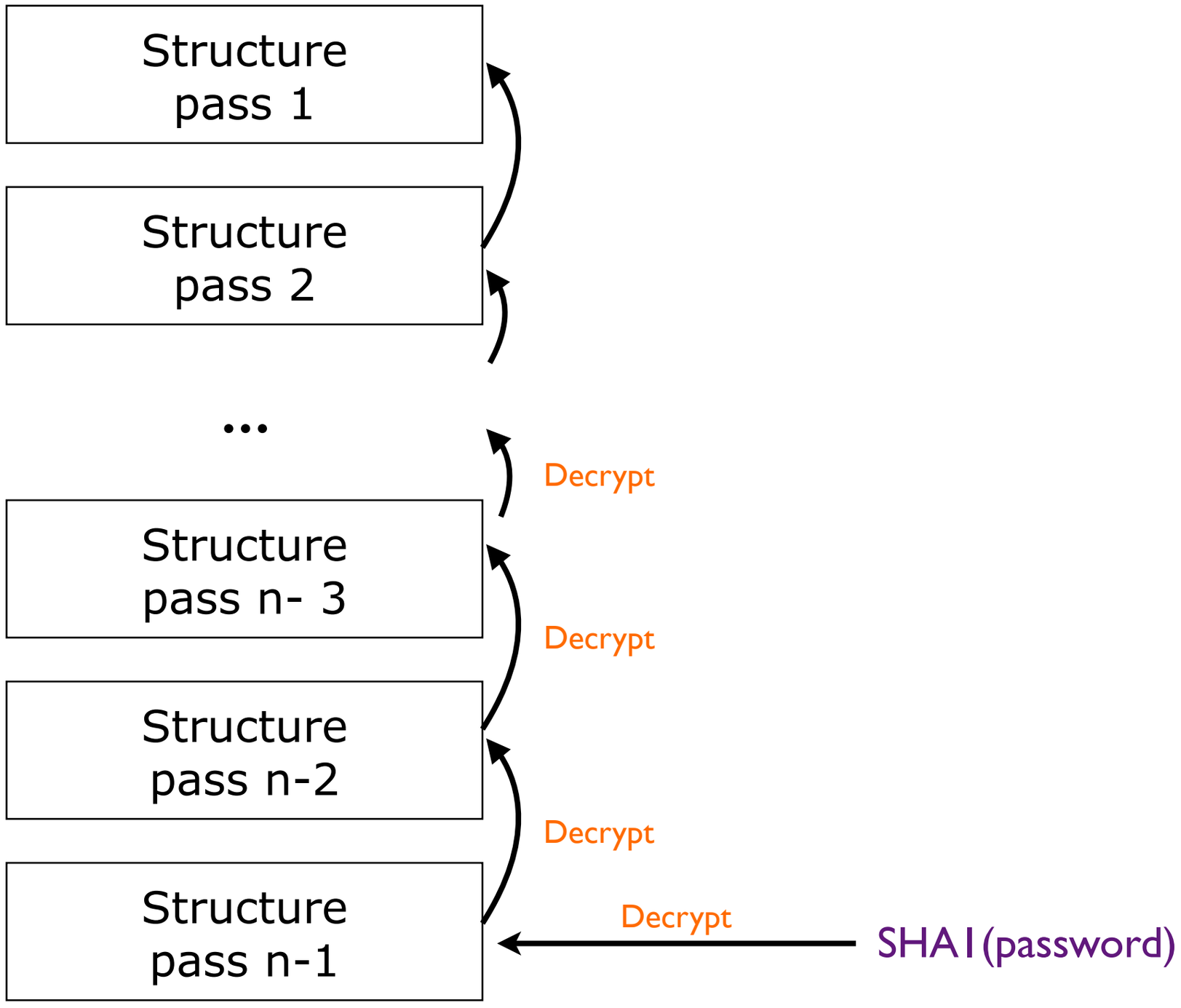
CREDHIST overview



CREDHIST overview



CREDHIST overview



CREDHIST entry structure main fields

idHashAlgo;

dwRounds;

dwCipherAlgo;

bSID[12];

dwComputerSID[3];

dwAccountID;

bData[28];

bPasswordID[16]

CREDHIST entry structure main fields

idHashAlgo; ← Hash algo ID

dwRounds;

dwCipherAlgo;

bSID[12];

dwComputerSID[3];

dwAccountID;

bData[28];

bPasswordID[16]

CREDHIST entry structure main fields

idHashAlgo;

dwRounds; ← Nb rounds

dwCipherAlgo;

bSID[12];

dwComputerSID[3];

dwAccountID;

bData[28];

bPasswordID[16]

CREDHIST entry structure main fields

idHashAlgo;

dwRounds;

dwCipherAlgo; ← Encryption Algorithm ID

bSID[12];

dwComputerSID[3];

dwAccountID;

bData[28];

bPasswordID[16]

CREDHIST entry structure main fields

idHashAlgo;

dwRounds;

dwCipherAlgo;

bSID[12]; ← User USID

dwComputerSID[3];

dwAccountID;

bData[28];

bPasswordID[16]

CREDHIST entry structure main fields

idHashAlgo;

dwRounds;

dwCipherAlgo;

bSID[12];

dwComputerSID[3];

dwAccountID;

bData[28];

bPasswordID[16]

Computer SID

CREDHIST entry structure main fields

idHashAlgo;

dwRounds;

dwCipherAlgo;

bSID[12];

dwComputerSID[3];

dwAccountID;

bData[28];

bPasswordID[16]

Account ID

CREDHIST entry structure main fields

idHashAlgo;

dwRounds;

dwCipherAlgo;

bSID[12];

dwComputerSID[3];

dwAccountID;

bData[28]; ← Encrypted password SHA1

bPasswordID[16]

CREDHIST entry structure main fields

idHashAlgo;

dwRounds;

dwCipherAlgo;

bSID[12];

dwComputerSID[3];

dwAccountID;

bData[28];

bPasswordID[16] ← Password GUID

Decryption algorithm overview

DecryptCredhist{

SID = (USID-ComputerID-AccountID)

tmp-key = HMAC(sha1, SID)

pre-key = PBKDF2(decryptKey, Salt, ID_ALGO, nbIteration)

3desKey = pre-key[0 - 23]

3desIV = [24 - 31]

(SHA1[0-19], HMAC[20-39]) = 3des-cbc (3desKey, iv, encKey)

DPAPIck demo

grab a copy from http://dpapick.com

LSA

- LSASS secret contains a DPAPI_SYSTEM value
- Length == 2 * SHA1
- Usage are unknown
- We think that 1 of them is used as a SYSTEM account "password"
- Need to be confirmed

EFS

- Certificate private key is encrypted with DPAPI
- Key are stored in

- To read EFS file offline, we just need to import the user certificate and its private keys in our key store.
- Work in progress in DPAPIck

What is next

- Can we build a rogue crypto provider ?
- What are the two SHA1 stored in the LSA ?
- Where is stored the renewal hard lime ?
- CryptDeriveKey needed to be reversed to have a fully portable implementation (Everything else is already portable)

Conclusion

- Open the door to offline forensic
- First step toward EFS on alternative systems
- CREDHIST allows to recover previous passwords
- DPAPIck : http://dpapick.com
- Some things remain unknown

Questions ?

Thanks to the nightingale team