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 and the account SID

Developer point of view

Application

DPAPI

DPAPI is a simple API*

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

Why digging deeper ?

- Offline forensic
- EFS on Linux
- Security / cool things ?

Previous work

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

Take away

- Decrypt offline sensitive data
- Recover user previous passwords (Yes all of them)
- Do a key escrow attack

Outline

Outline

- DPAPI overview

Outline

- DPAPI overview
- Decryption process

Outline

- DPAPI overview
- Decryption process
- Security design implications

Outline

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

Crypto 911 HMAC

- HMAC (Message authentication code)
  - Usually used to detect data tampering
  - Used here to derive encrypt key and IV

ipad = 0x36 xor key

opad = 0x5c xor key

HMAC= (opad . SHA1(ipad.data))

Crypto 911: PBKDF2

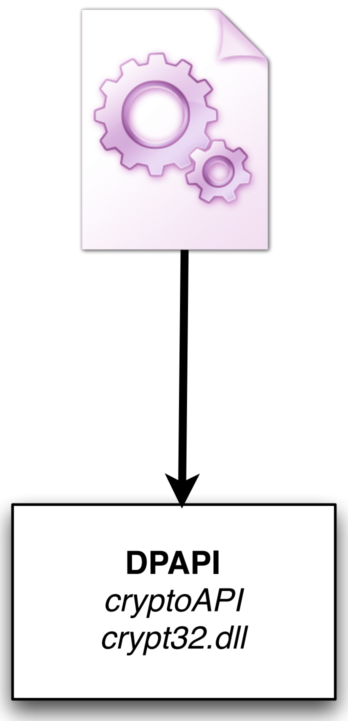
- PBKDF2 = Password based key derivation function
- Basically it is a hash function (SHA1 for us) applied n times to slow down the computation.
- Used to defend against brute-force
- Salt is used against rainbow tables attacks.

Crypto 911 : 3DES

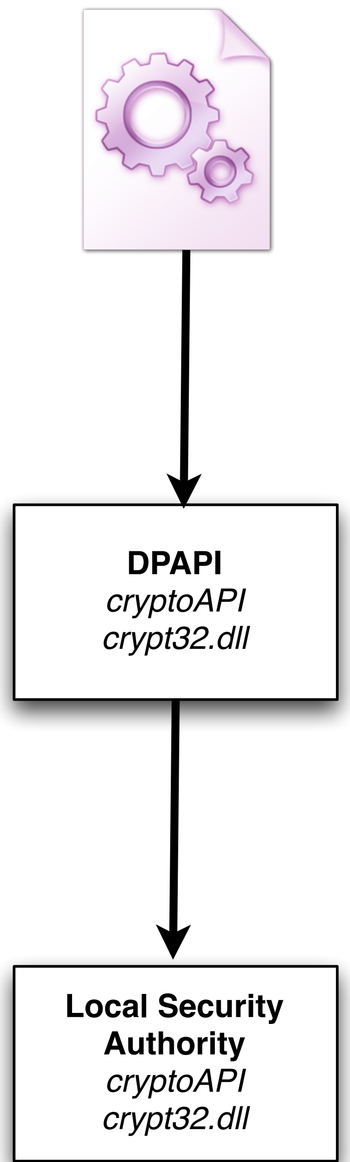
- 3DES : Triple DES encryption
  - Encrypt, Decrypt, Encrypt
  - Exist in two flavor : 2 keys or 3 keys (64 bits each)
  - Windows use the strong version with 3 keys

How the system interacts with DPAPI

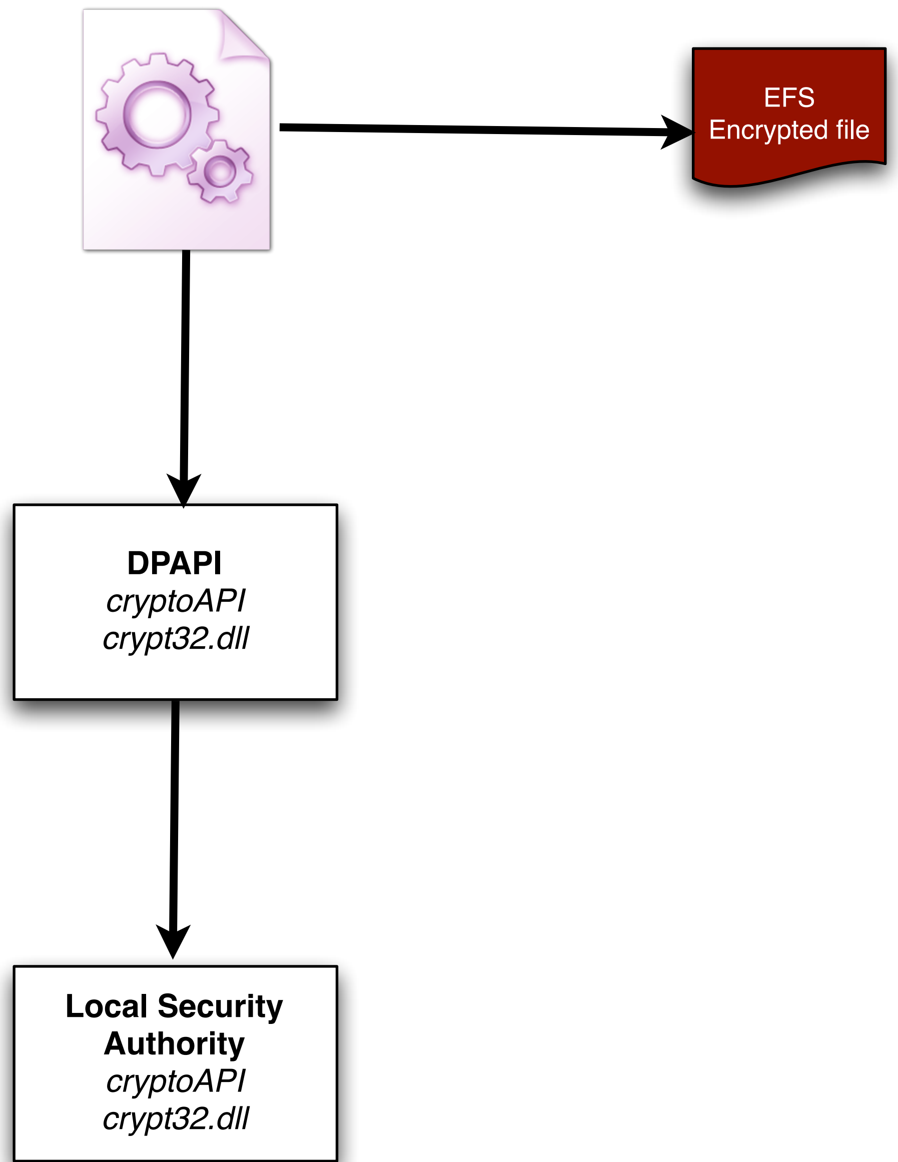
How the system interacts with DPAPI



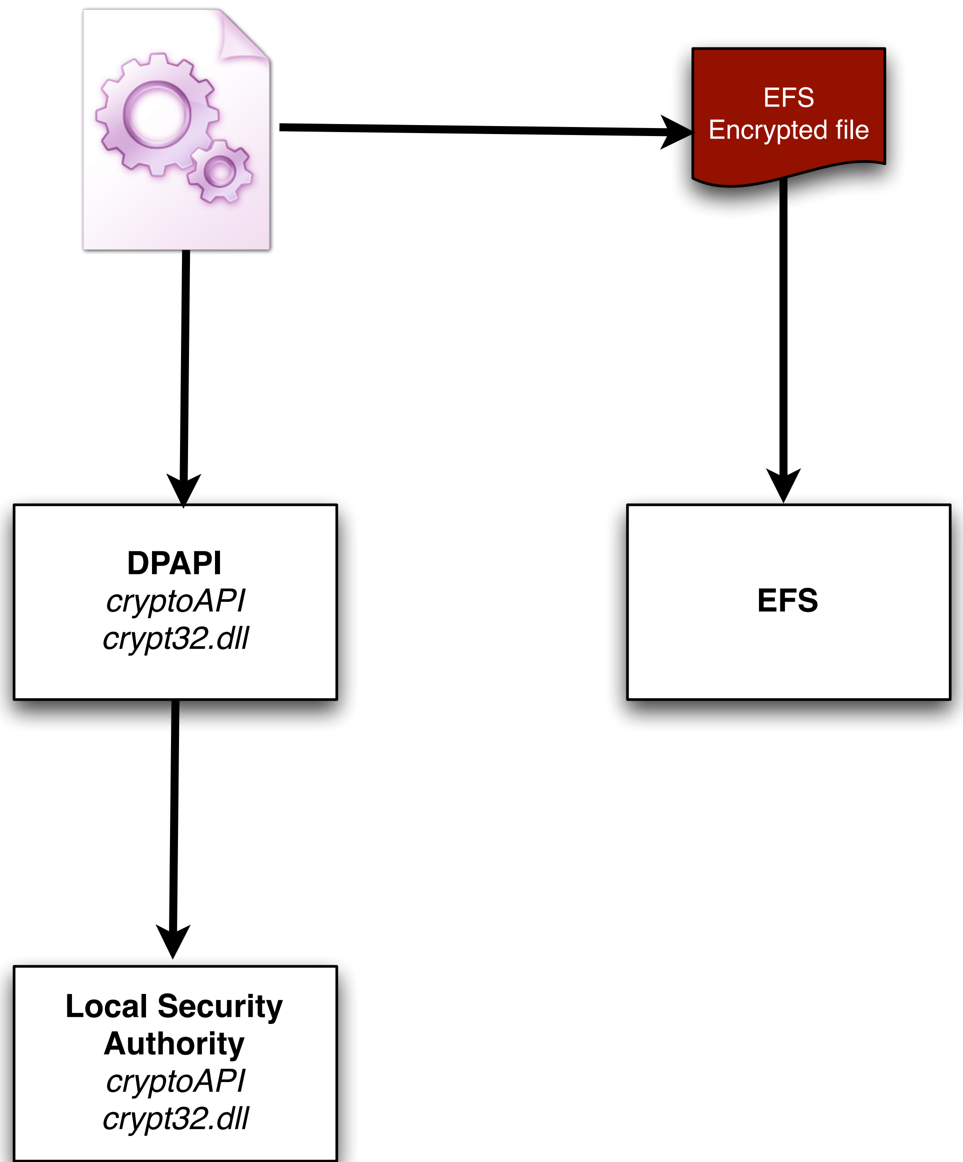
How the system interacts with DPAPI



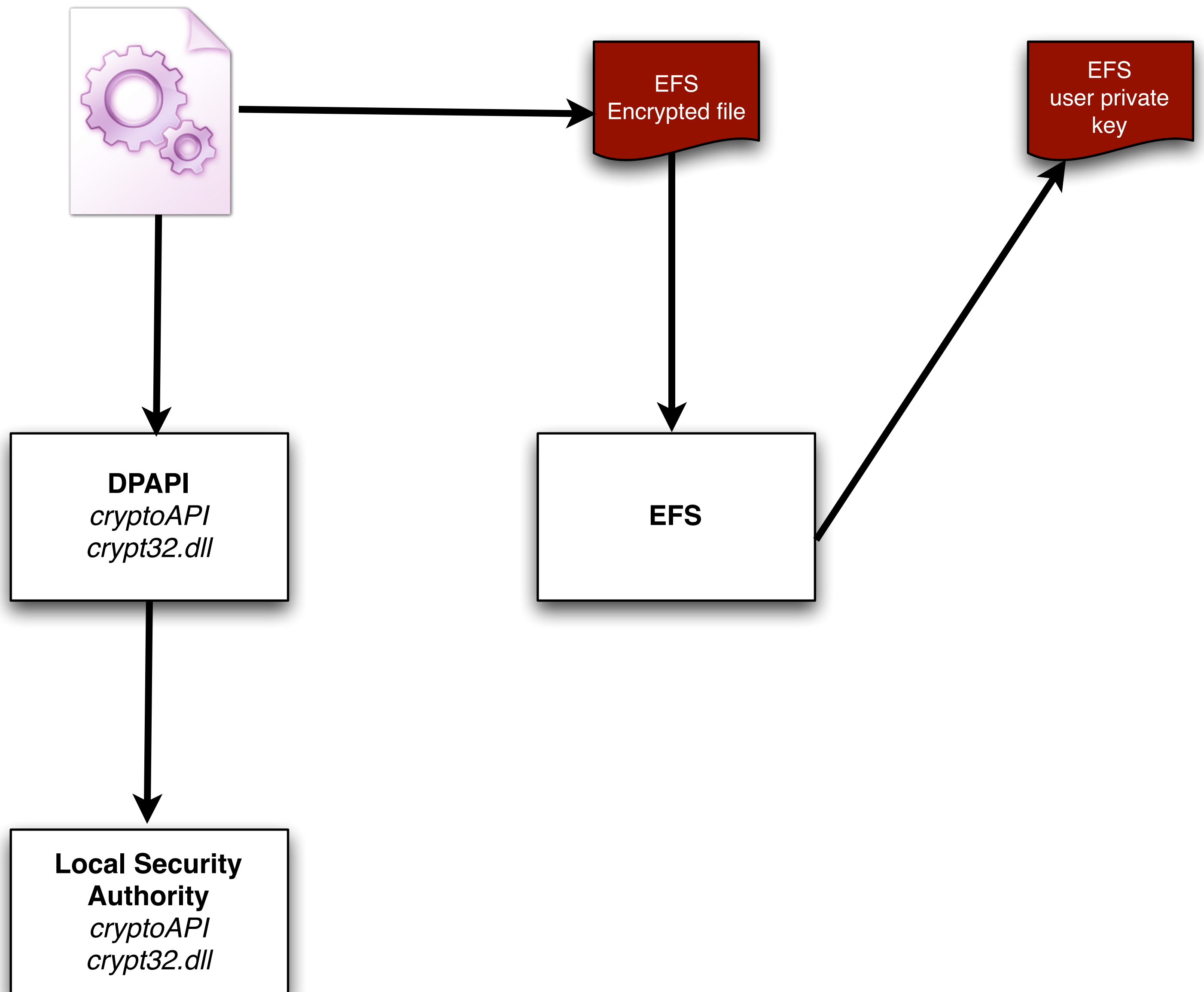
How the system interacts with DPAPI



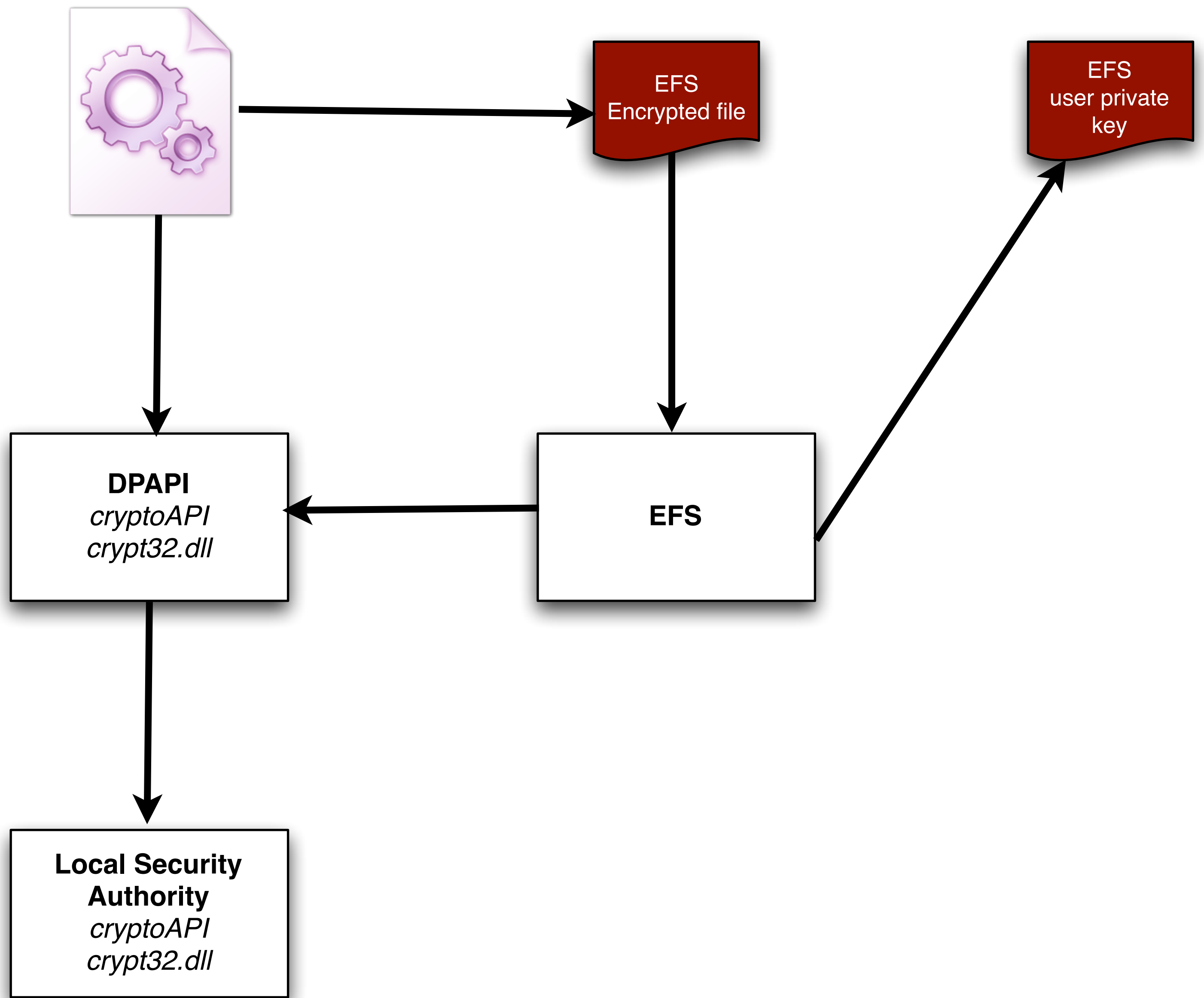
How the system interacts with DPAPI



How the system interacts with DPAPI



How the system interacts with DPAPI



DPAPI CryptUnprotecData Function

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

DPAPI CryptUnprotecData Function

BOOL WINAPI CryptUnprotectData (

*pDataIn, ← Encrypted data aka data blob

*ppszDataDescr,

*pOptionalEntropy,

pvReserved,

*pPromptStruct,

dwFlags,

*pDataOut

DPAPI CryptUnprotecData Function

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

DPAPI CryptUnprotecData Function

BOOL WINAPI CryptUnprotectData (

*pDataIn,

*ppszDataDescr,

*pOptionalEntropy, ← Optional entropy (salt)

pvReserved,

*pPromptStruct,

dwFlags,

*pDataOut

DPAPI CryptUnprotecData Function

BOOL WINAPI CryptUnprotectData (

*pDataIn,

*ppszDataDescr,

*pOptionalEntropy,

pvReserved,

*pPromptStruct, ← Optional password

dwFlags,

*pDataOut

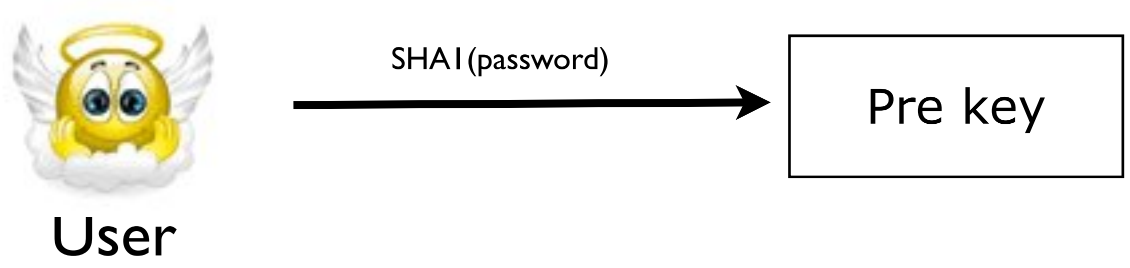
DPAPI CryptUnprotecData Function

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

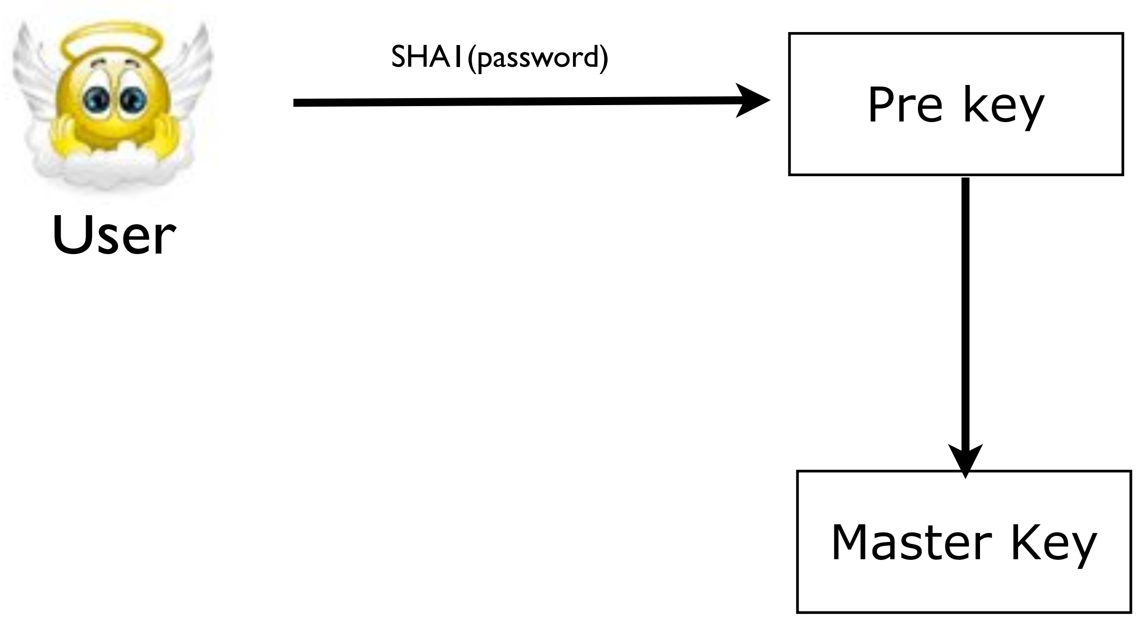
Derivation scheme

User

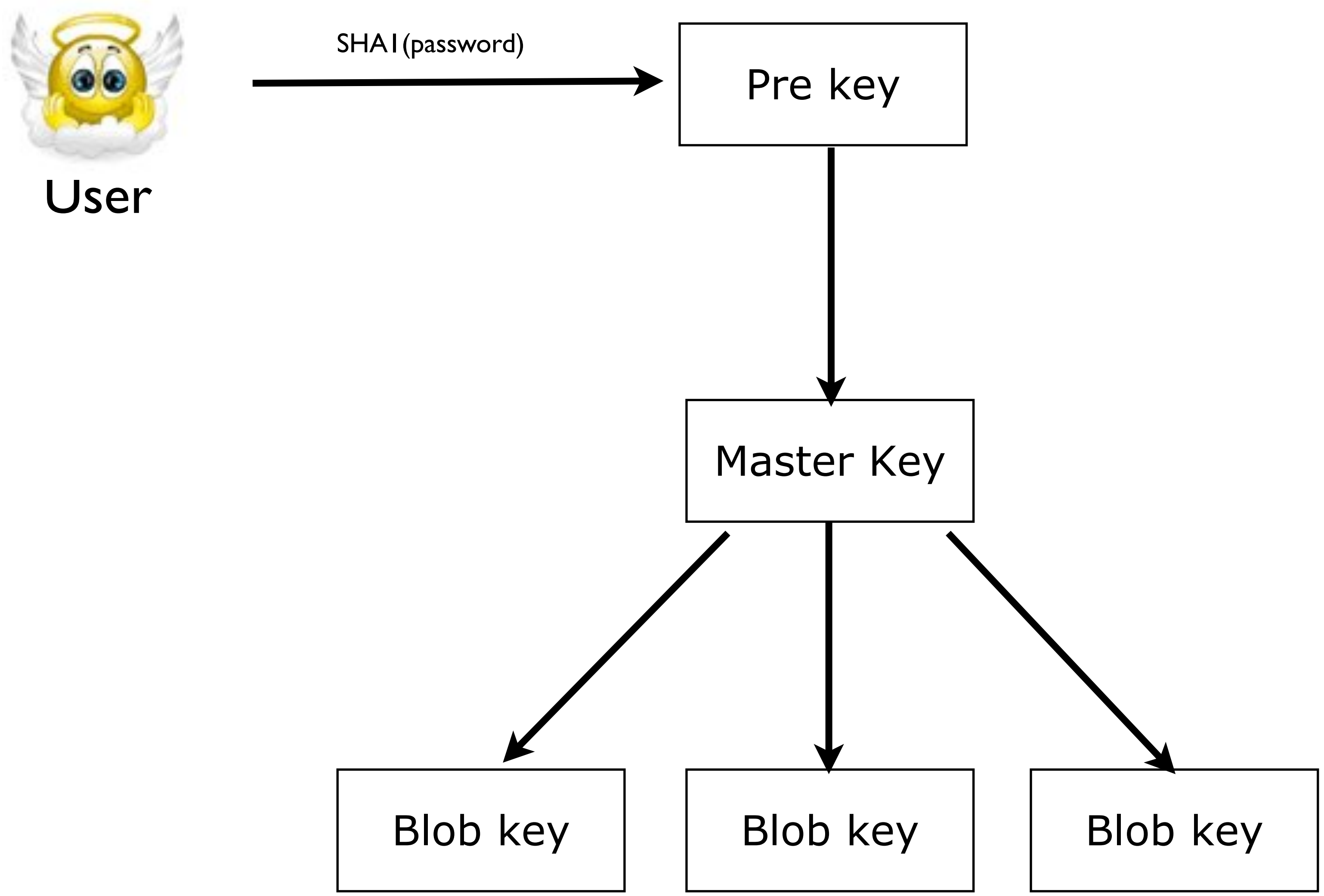
Derivation scheme



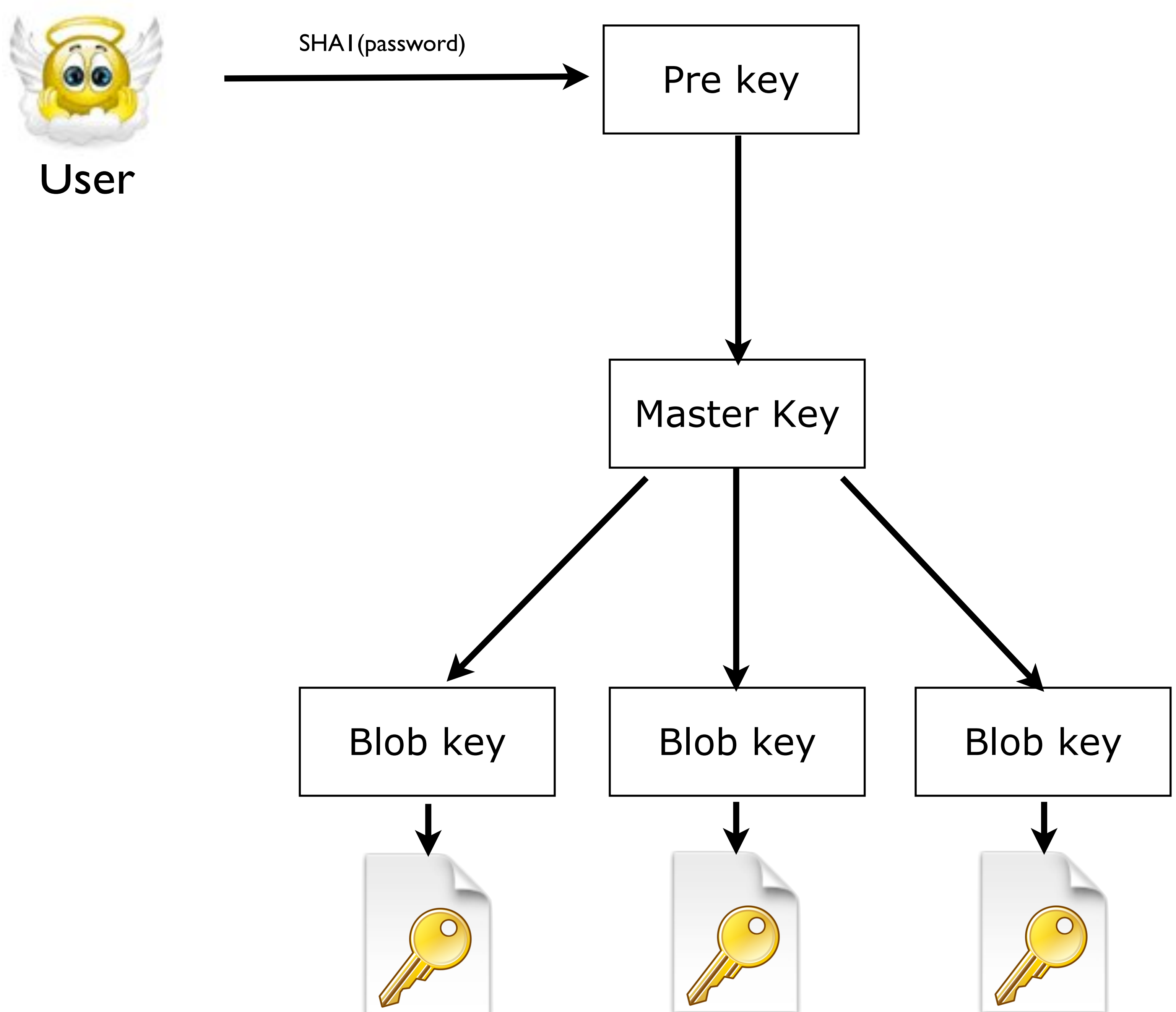
Derivation scheme



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     *pbSalt;
DWORD    idHashAlgo;
BYTE     *pbUnknown;
BYTE     *pbCipher;
BYTE     *pbHMAC;
```

data blob structure key fields

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

Nb of crypto providers

data blob structure key fields

```
DWORD    cbProviders;
GUID     *arrProviders;      <—— Crypto providers GUID
DWORD    cbKeys;
GUID     *arrKeys;
WCHAR    *ppszDataDescr;
DWORD    idCipherAlgo;
BYTE     *pbSalt;
DWORD    idHashAlgo;
BYTE     *pbUnknown;
BYTE     *pbCipher;
BYTE     *pbHMAC;
```

data blob structure key fields

```
DWORD    cbProviders;
GUID     *arrProviders;
DWORD    cbKeys;          <-- Nb of masters keys
GUID     *arrKeys;
WCHAR    *ppszDataDescr;
DWORD    idCipherAlgo;
BYTE     *pbSalt;
DWORD    idHashAlgo;
BYTE     *pbUnknown;
BYTE     *pbCipher;
BYTE     *pbHMAC;
```

data blob structure key fields

```
DWORD    cbProviders;
GUID     *arrProviders;
DWORD    cbKeys;
GUID     *arrKeys;          ←  Masters keys GUID
WCHAR    *ppszDataDescr;
DWORD    idCipherAlgo;
BYTE     *pbSalt;
DWORD    idHashAlgo;
BYTE     *pbUnknown;
BYTE     *pbCipher;
BYTE     *pbHMAC;
```

data blob structure key fields

```
DWORD    cbProviders;
GUID     *arrProviders;
DWORD    cbKeys;
GUID     *arrKeys;
WCHAR    *ppszDataDescr;   ←   Optional description
DWORD    idCipherAlgo;
BYTE     *pbSalt;
DWORD    idHashAlgo;
BYTE     *pbUnknown;
BYTE     *pbCipher;
BYTE     *pbHMAC;
```

data blob structure key fields

```
DWORD    cbProviders;
GUID     *arrProviders;
DWORD    cbKeys;
GUID     *arrKeys;
WCHAR    *ppszDataDescr;
DWORD    idCipherAlgo;      ← Encryption algorithm ID
BYTE     *pbSalt;
DWORD    idHashAlgo;
BYTE     *pbUnknown;
BYTE     *pbCipher;
BYTE     *pbHMAC;
```

data blob structure key fields

```
DWORD    cbProviders;
GUID     *arrProviders;
DWORD    cbKeys;
GUID     *arrKeys;
WCHAR    *ppszDataDescr;
DWORD    idCipherAlgo;
BYTE     *pbSalt;          ← Salt generated by DPAPI
DWORD    idHashAlgo;
BYTE     *pbUnknown;
BYTE     *pbCipher;
BYTE     *pbHMAC;
```

data blob structure key fields

```
DWORD   cbProviders;
GUID    *arrProviders;
DWORD   cbKeys;
GUID    *arrKeys;
WCHAR   *ppszDataDescr;
DWORD   idCipherAlgo;
BYTE    *pbSalt;
DWORD   idHashAlgo;      <---- Hash algorithm ID
BYTE    *pbUnknown;
BYTE    *pbCipher;
BYTE    *pbHMAC;
```

data blob structure key fields

```
DWORD    cbProviders;
GUID     *arrProviders;
DWORD    cbKeys;
GUID     *arrKeys;
WCHAR    *ppszDataDescr;
DWORD    idCipherAlgo;
BYTE     *pbSalt;
DWORD    idHashAlgo;
BYTE     *pbUnknown;      ← Unknown data
BYTE     *pbCipher;
BYTE     *pbHMAC;
```

data blob structure key fields

```
DWORD    cbProviders;
GUID     *arrProviders;
DWORD    cbKeys;
GUID     *arrKeys;
WCHAR    *ppszDataDescr;
DWORD    idCipherAlgo;
BYTE     *pbSalt;
DWORD    idHashAlgo;
BYTE     *pbUnknown;
BYTE     *pbCipher;       ← Encrypted data
BYTE     *pbHMAC;
```

data blob structure key fields

```
DWORD    cbProviders;
GUID     *arrProviders;
DWORD    cbKeys;
GUID     *arrKeys;
WCHAR    *ppszDataDescr;
DWORD    idCipherAlgo;
BYTE     *pbSalt;
DWORD    idHashAlgo;
BYTE     *pbUnknown;
BYTE     *pbCipher;
BYTE     *pbHMAC;     ←—— Blob HMAC
```

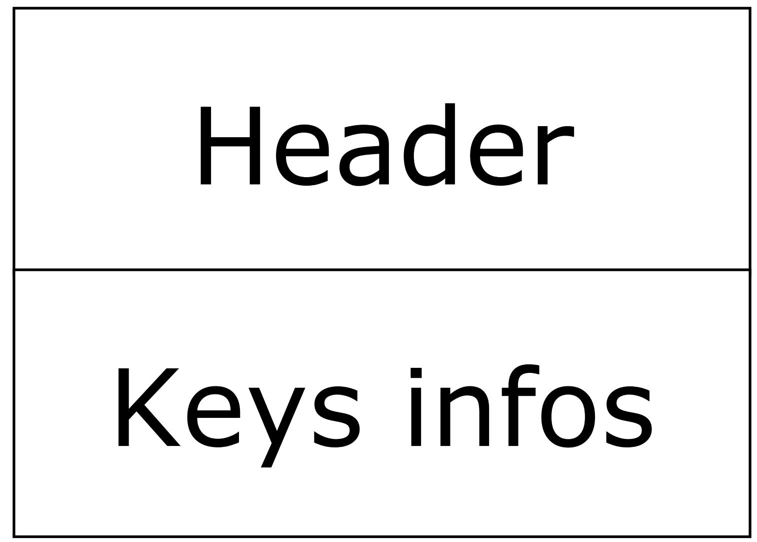
Master key structure

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

The master key file

Header

The master key file



The master key file

Header
Keys infos
Master key

The master key file

Header
Keys infos
Master key
Key ?

The master key file

Header
Keys infos
Master key
Key ?
Footer

Header structure

Header
Keys infos
Master key
Key ?
Footer

dwVersion;

nullPad1;

szKeyGUID[36];

nullPad2;

Header structure

Header
Keys infos
Master key
Key ?
Footer

dwVersion; ← File version

nullPad1;

szKeyGUID[36];

nullPad2;

Header structure

Header
Keys infos
Master key
Key ?
Footer

dwVersion;

nullPad1;

szKeyGUID[36]; ← Master key GUID

nullPad2;

Key infos structure

Header
Keys infos
Master key
Key ?
Footer

dwUnknown;

cbMasterKey;

cbMysteryKey;

dwHMACLen;

nullPad3;

Key infos structure

Header
Keys infos
Master key
Key ?
Footer

dwUnknown;

cbMasterKey; ← Master Key struct length

cbMysteryKey;

dwHMACLen;

nullPad3;

Key infos structure

Header
Keys infos
Master key
Key ?
Footer

dwUnknown;

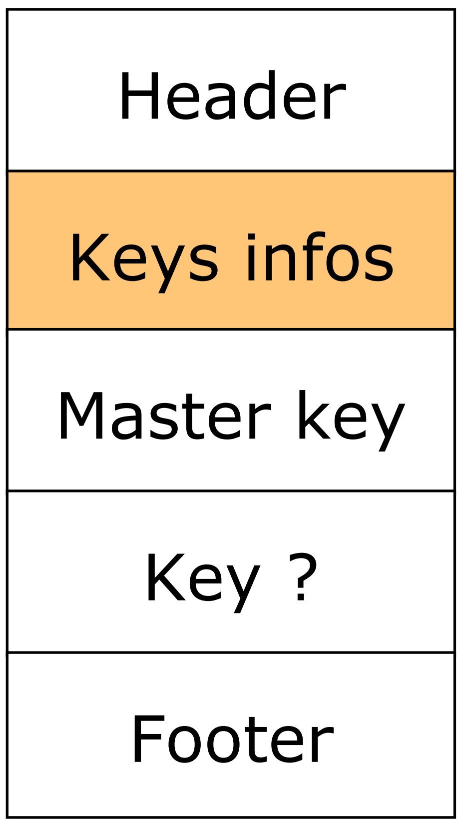
cbMasterKey;

cbMysteryKey; ← Key ? struct length

dwHMACLen;

nullPad3;

Key infos structure



dwUnknown;

cbMasterKey;

cbMysteryKey;

dwHMACLen; ← HMAC length

nullPad3;

Master key structure

Header
Keys infos
Master key
Key ?
Footer

dwMagic;

pbSalt[16];

cbIteration;

idMACAlgo;

idCipherAlgo;

pbCipheredKey[];

Master key structure

Header
Keys infos
Master key
Key ?
Footer

dwMagic;

pbSalt[16]; ← Key salt

cbIteration;

idMACAlgo;

idCipherAlgo;

pbCipheredKey[];

Master key structure

Header
Keys infos
Master key
Key ?
Footer

dwMagic;

pbSalt[16];

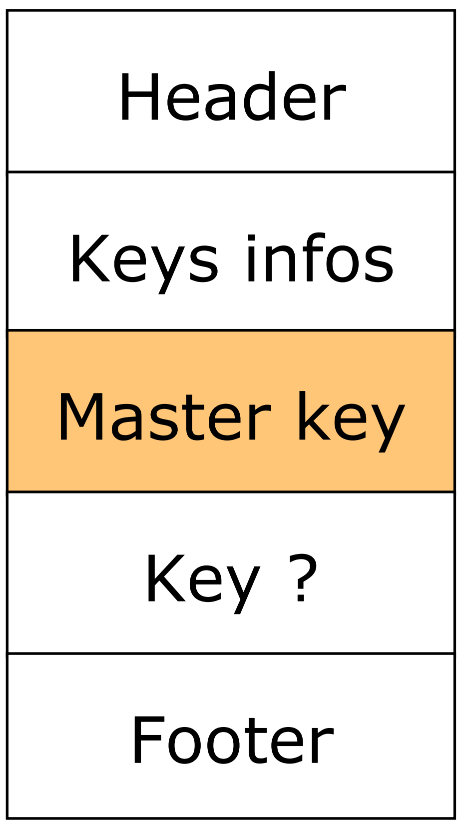
cbIteration; ← PBKDF2 nb rounds

idMACAlgo;

idCipherAlgo;

pbCipheredKey[];

Master key structure



dwMagic;

pbSalt[16];

cbIteration;

idMACAlgo; ← HMAC algorithm ID

idCipherAlgo;

pbCipheredKey[];

Master key structure

Header
Keys infos
Master key
Key ?
Footer

dwMagic;

pbSalt[16];

cbIteration;

idMACAlgo;

idCipherAlgo; ← Encryption Algo id

pbCipheredKey[];

Master key structure

Header
Keys infos
Master key
Key ?
Footer

dwMagic;

pbSalt[16];

cbIteration;

idMACAlgo;

idCipherAlgo;

pbCipheredKey[]; ← Encrypted key

Decrypting the Master key

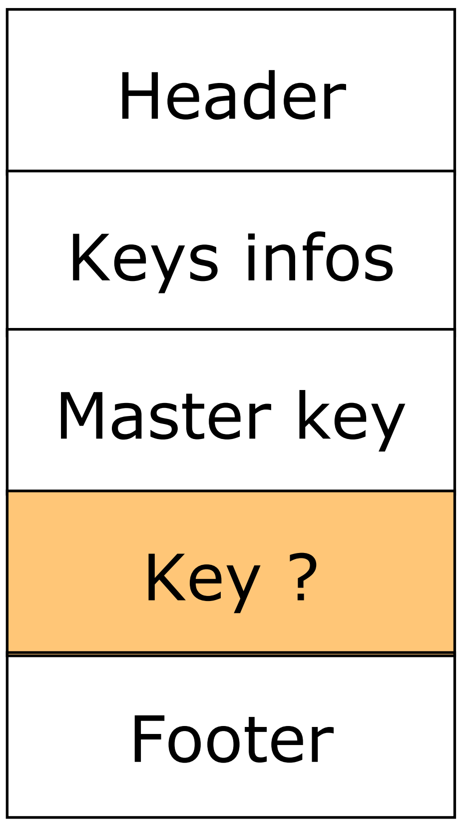
```
DPAPIDecryptKey(sha1, encKey) {
    tmp-key = HMAC(sha1, SID)
    pre-key = PBKDF2(decryptKey,  Salt, ID_ALGO,
    nbIteration)
    3desKey = pre-key[0 - 23]
    3desIV = [24 - 31]
    (hmac[0-35], DWORD[36-39], master-key
    [40-104]) = 3des-cbc(3desKey, iv, encKey)
}
```

key structure

Header
Keys infos
Master key
Key ?
Footer

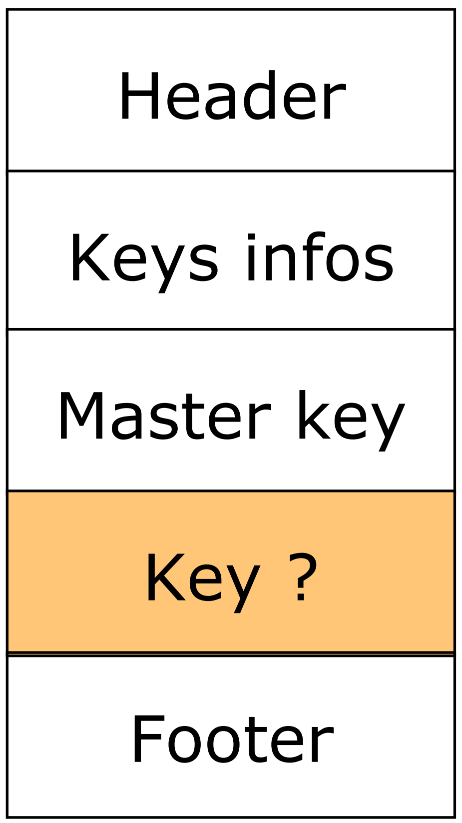
- Seems to have the same structure than the master key
- One round of derivation (XP not Seven)
- 256 bits (half size of the real master-key)

Possible explanation



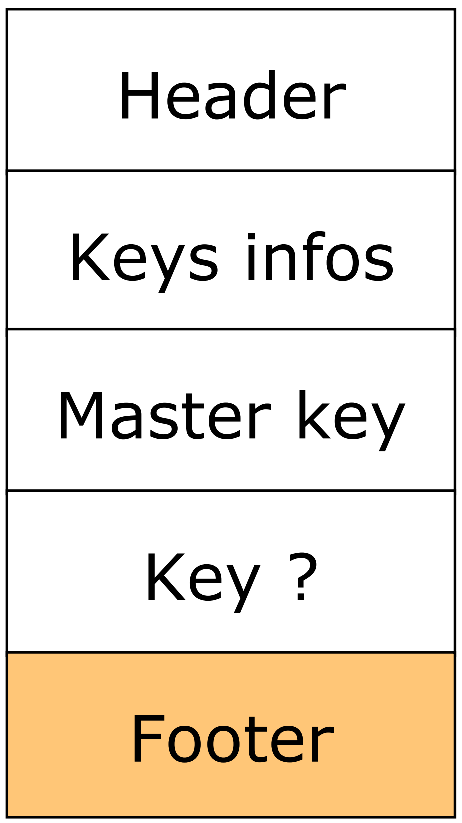
- The documentation state a compatibility mode for windows 2000 exist.
- The registry key to trigger it is unknown
- If we are correct and W2k uses RC4 then the mystery key is possibly a RC4 key (256bits is the correct size).
- PBKDF2 used to compute the IV ??

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

Header
Keys infos
Master key
Key ?
Footer

dwMagic;

credHist[16]; ← Password GUID

Differences between windows version

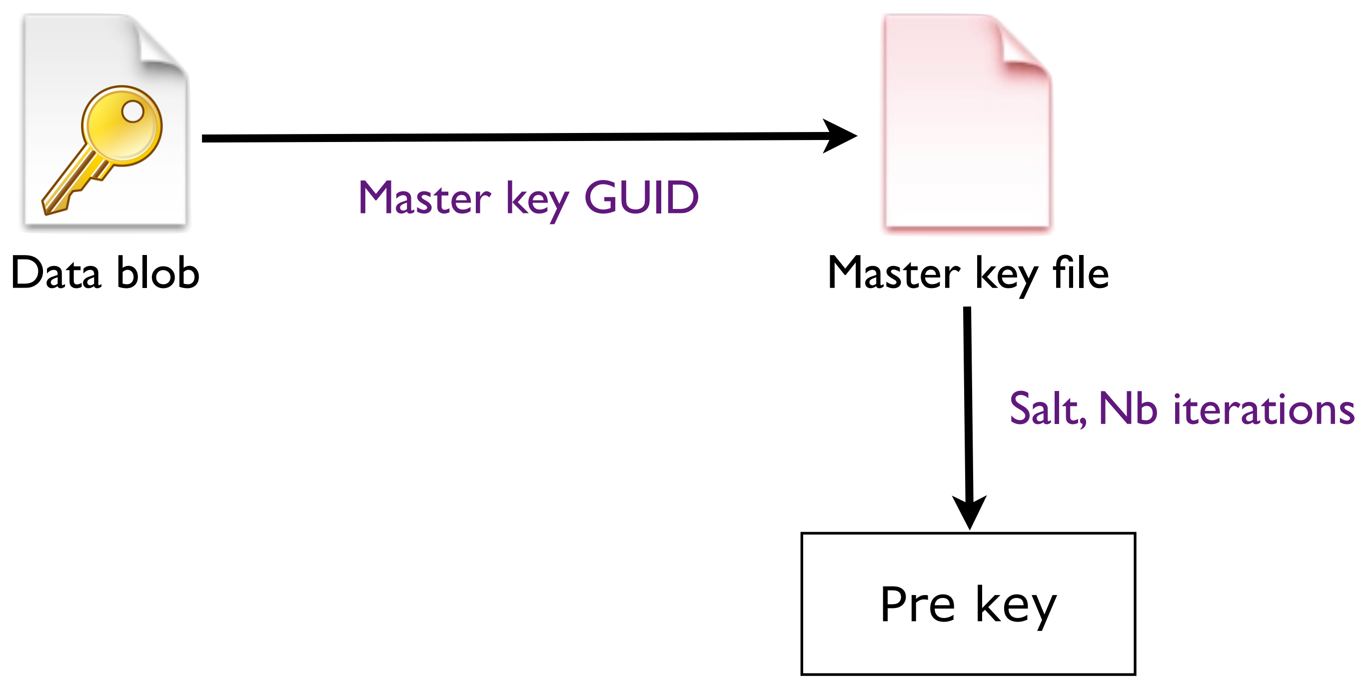
	XP	Vista	Seven
PBKDF2 rounds	4000	24000	Variable (factor ?)
Symmetric algorithm	3DES	3DES	AES
Hash algorithm	SHA1	SHA1	SHA512

Decrypting a blob

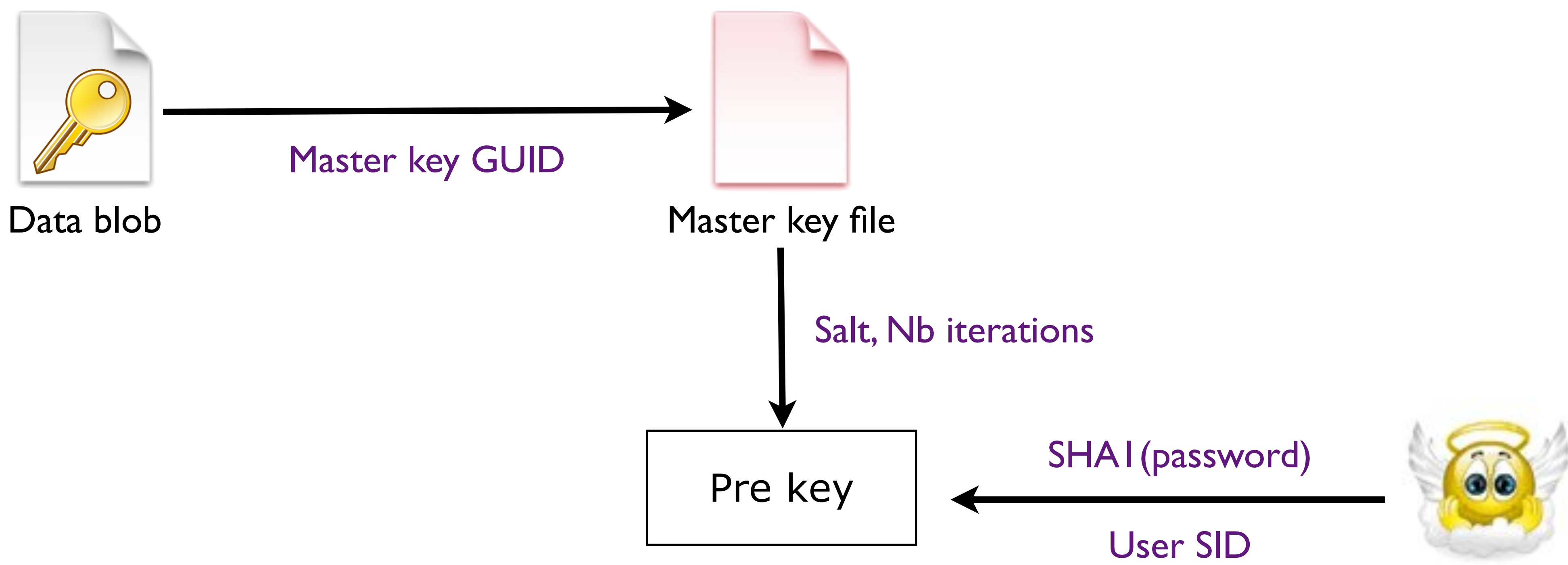
Data blob

Decrypting a blob

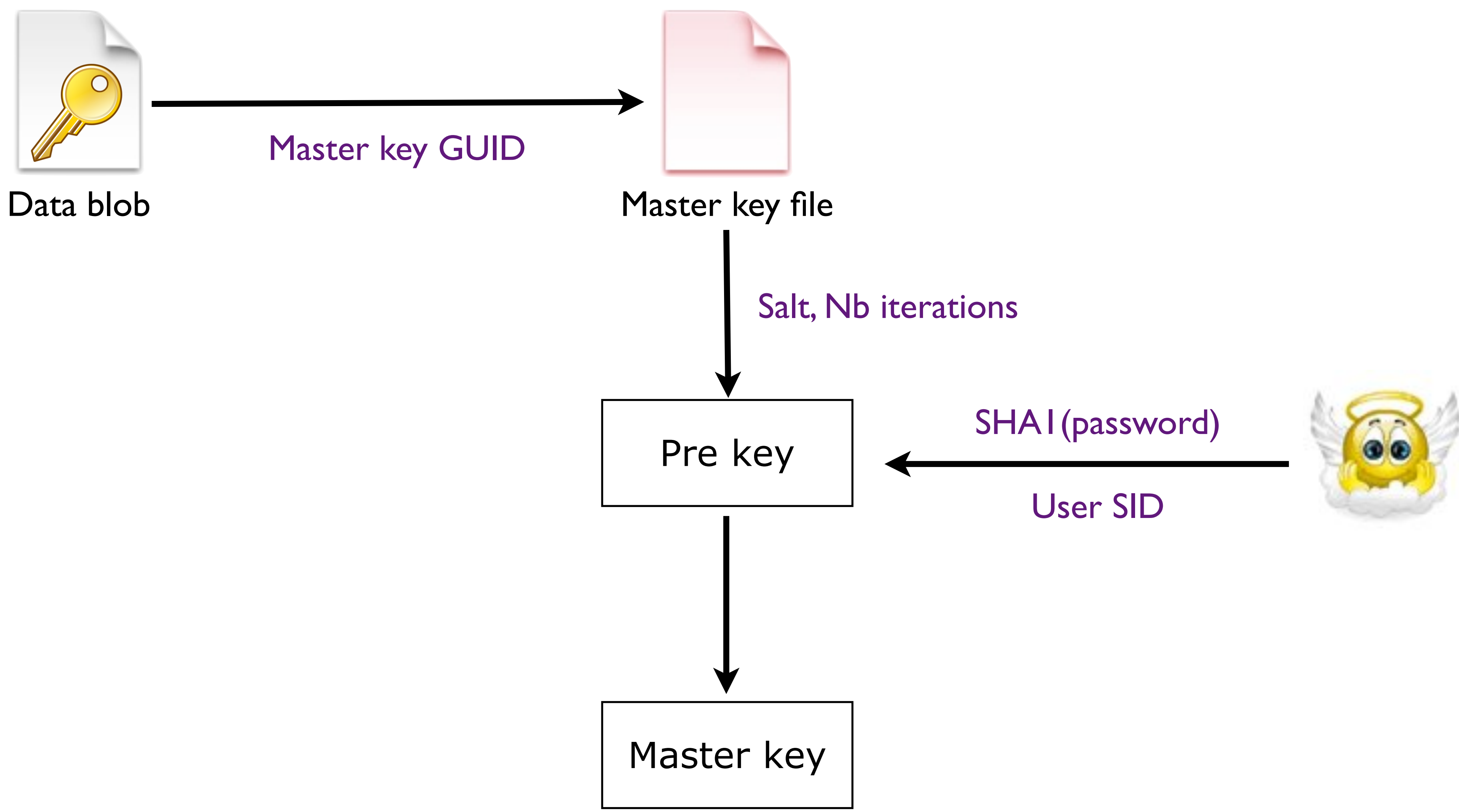
Decrypting a blob



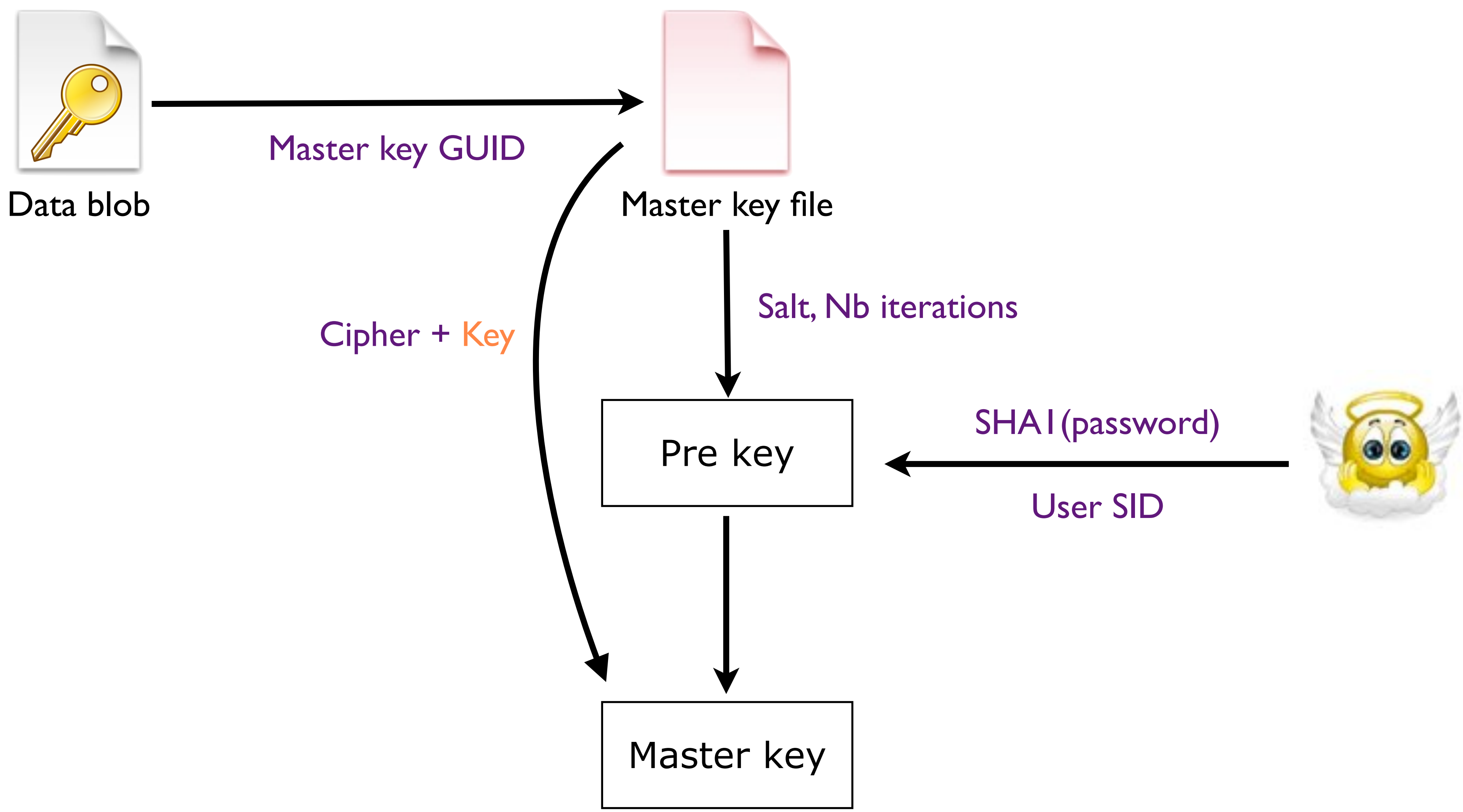
Decrypting a blob



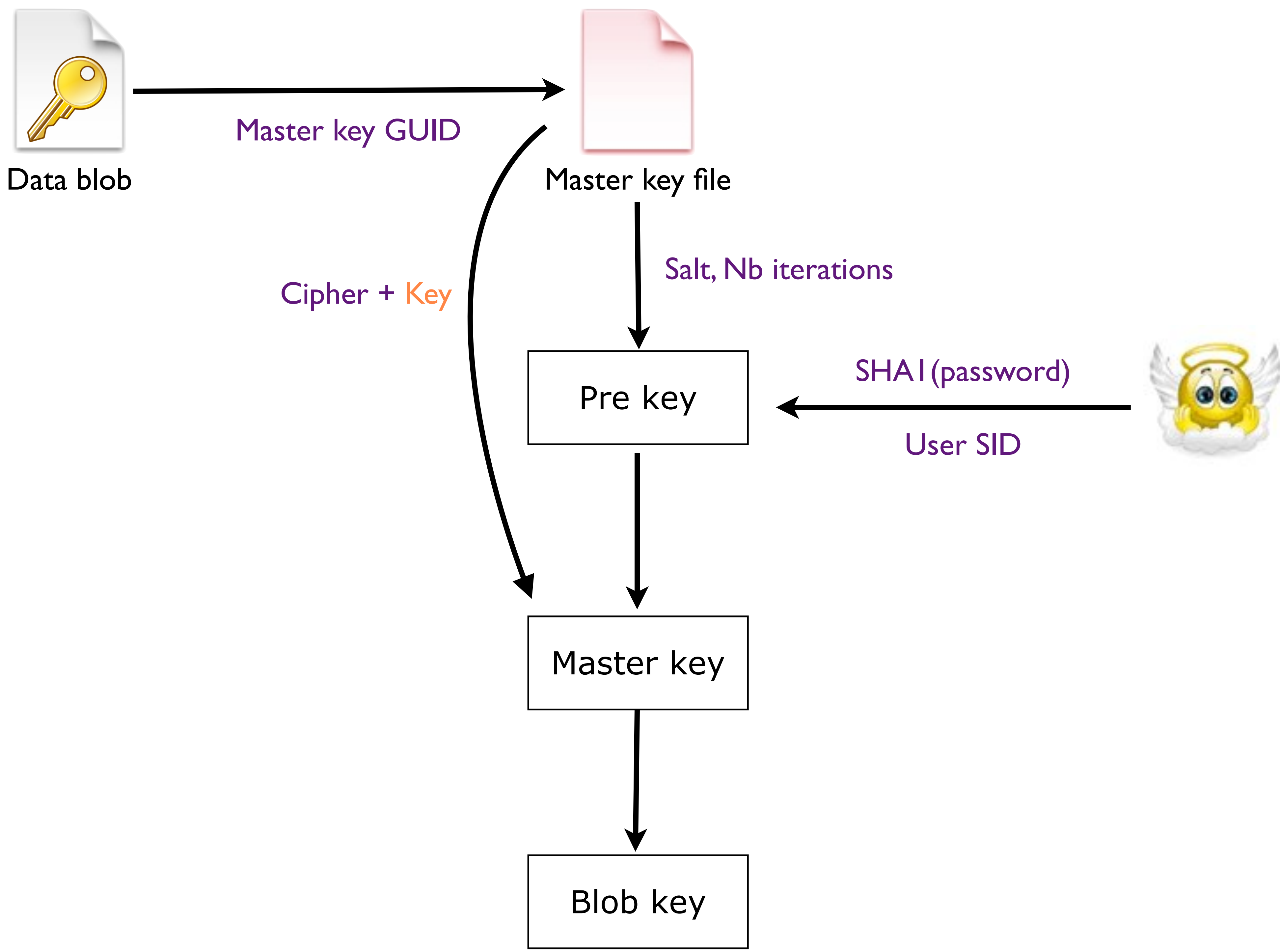
Decrypting a blob



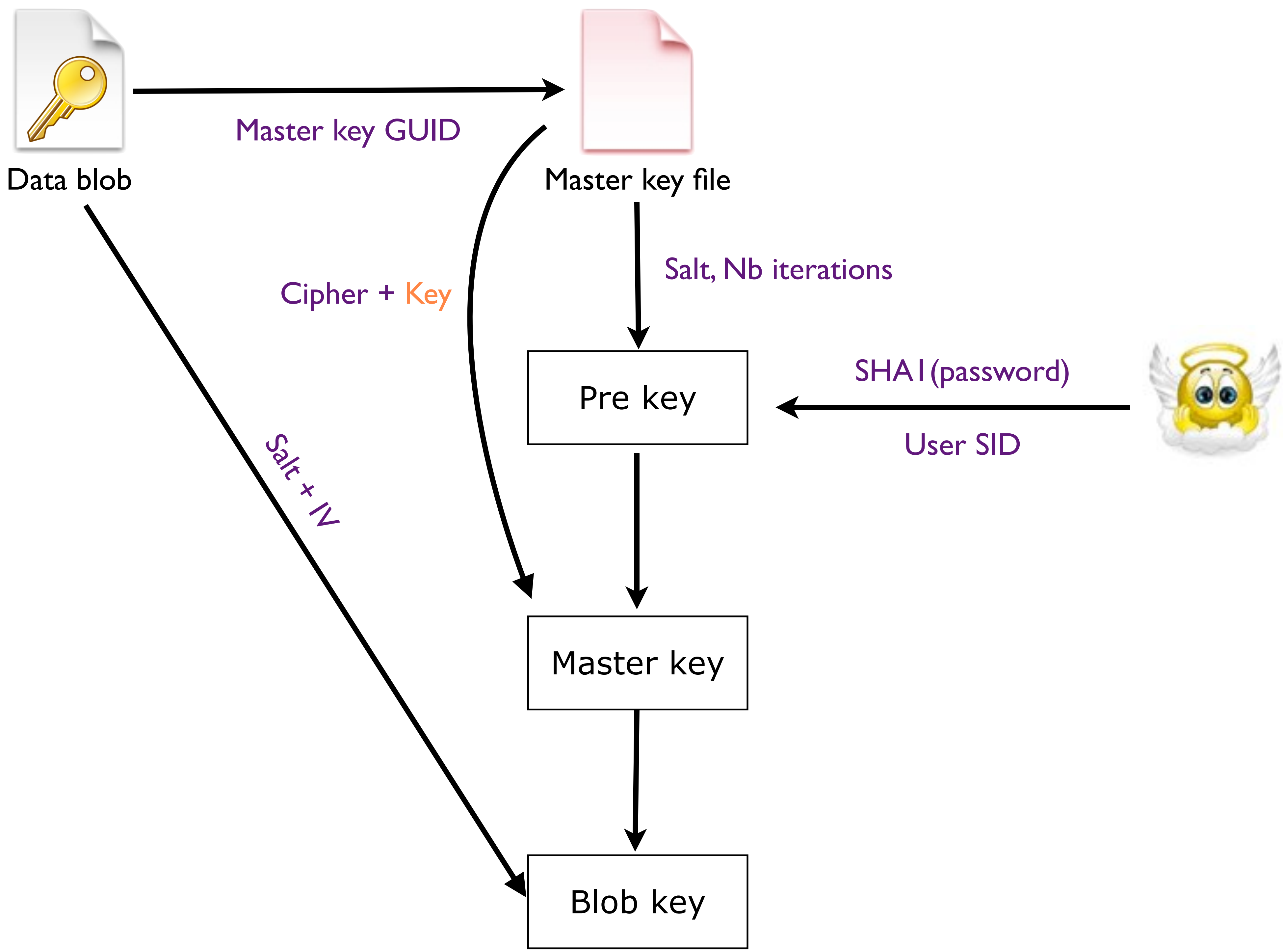
Decrypting a blob



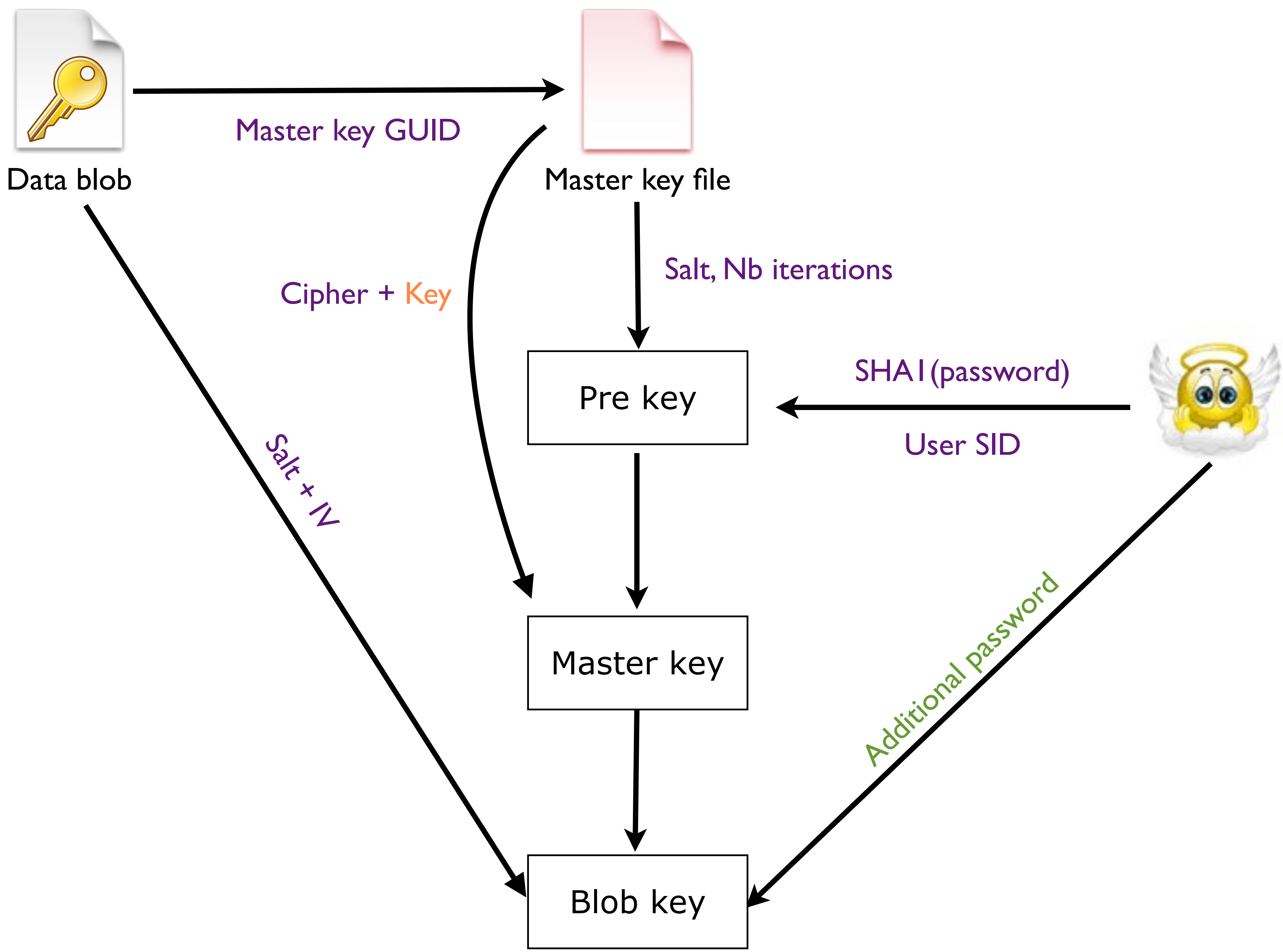
Decrypting a blob



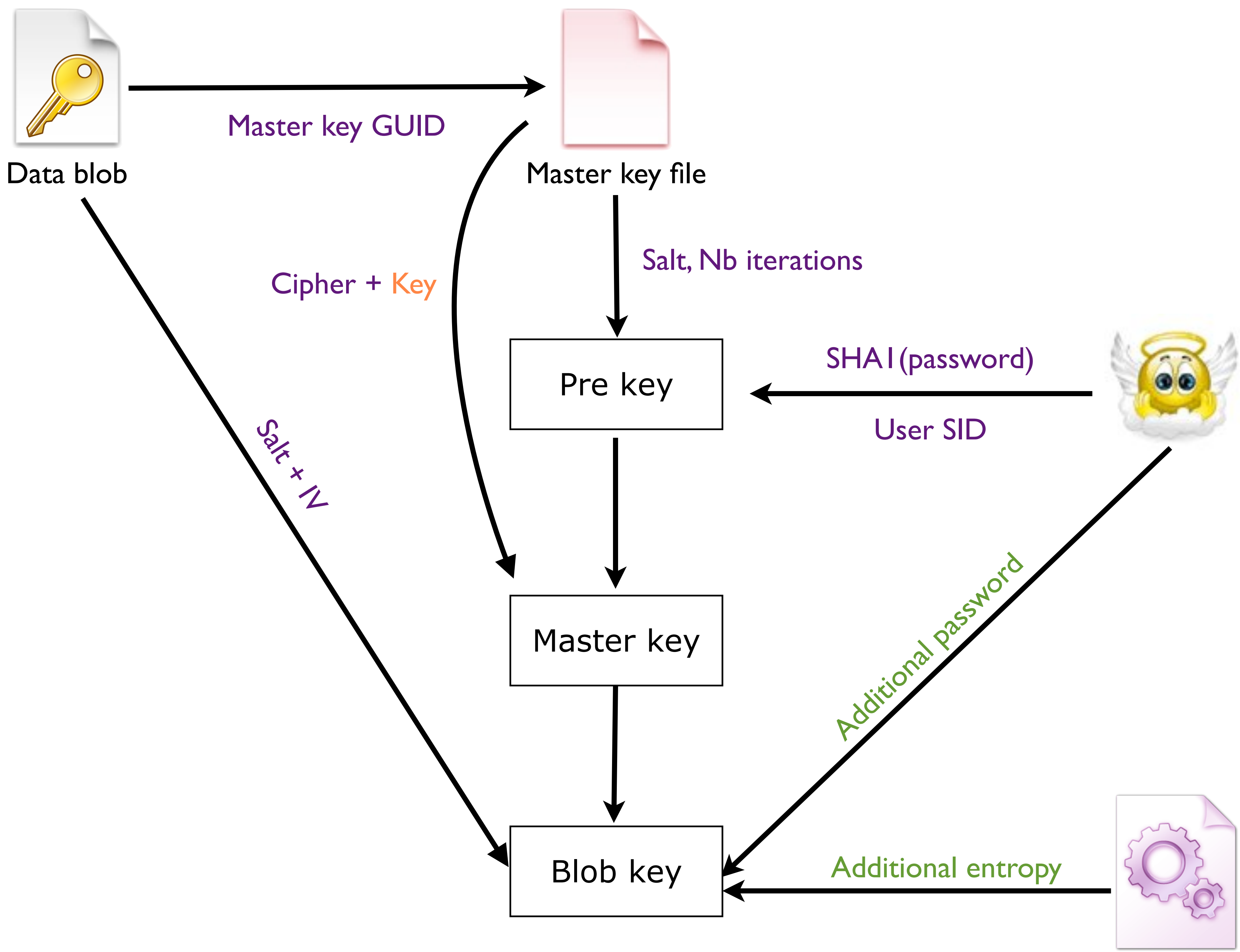
Decrypting a blob



Decrypting a blob



Decrypting a blob



Decrypt blob aka the strange HMAC

```
DecryptBlob() {
kt = SHA1(masterkey)
opad = 0x5c xor kt
ipad = 0x36 xor kt
i = SHA1(opad.SHA1(ipad . salt).entropyCond)
kd = CryptDeriveKey(i) //not reversed (yet)
CryptDecrypt(data, kd)
}
```

Did I miss something ?

Did I miss something ?

- How the OS knows the current master key ?

Did I miss something ?

- How the OS knows the current master key ?
- How the OS decides to renew the master key ?

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

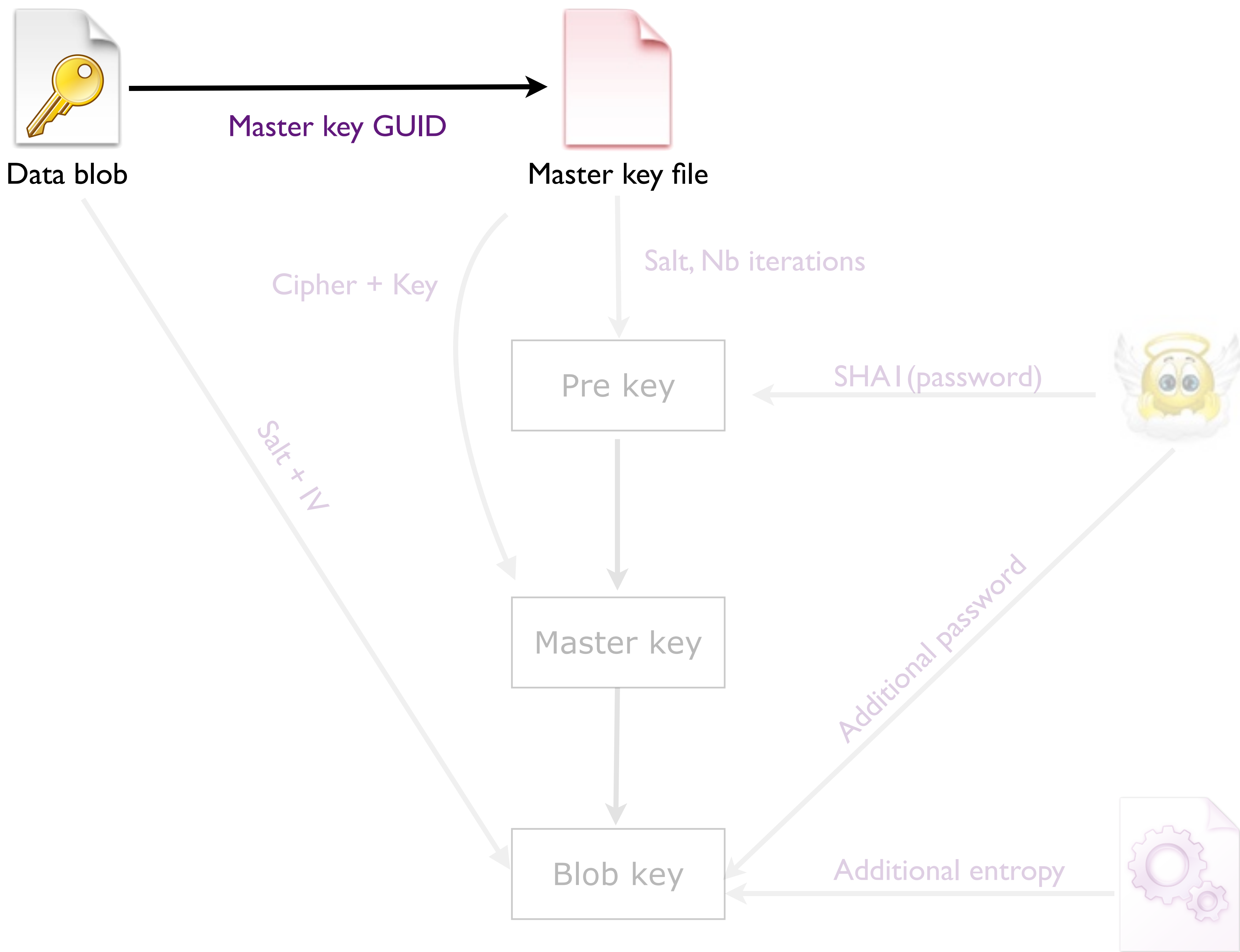
current time > timestamp

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

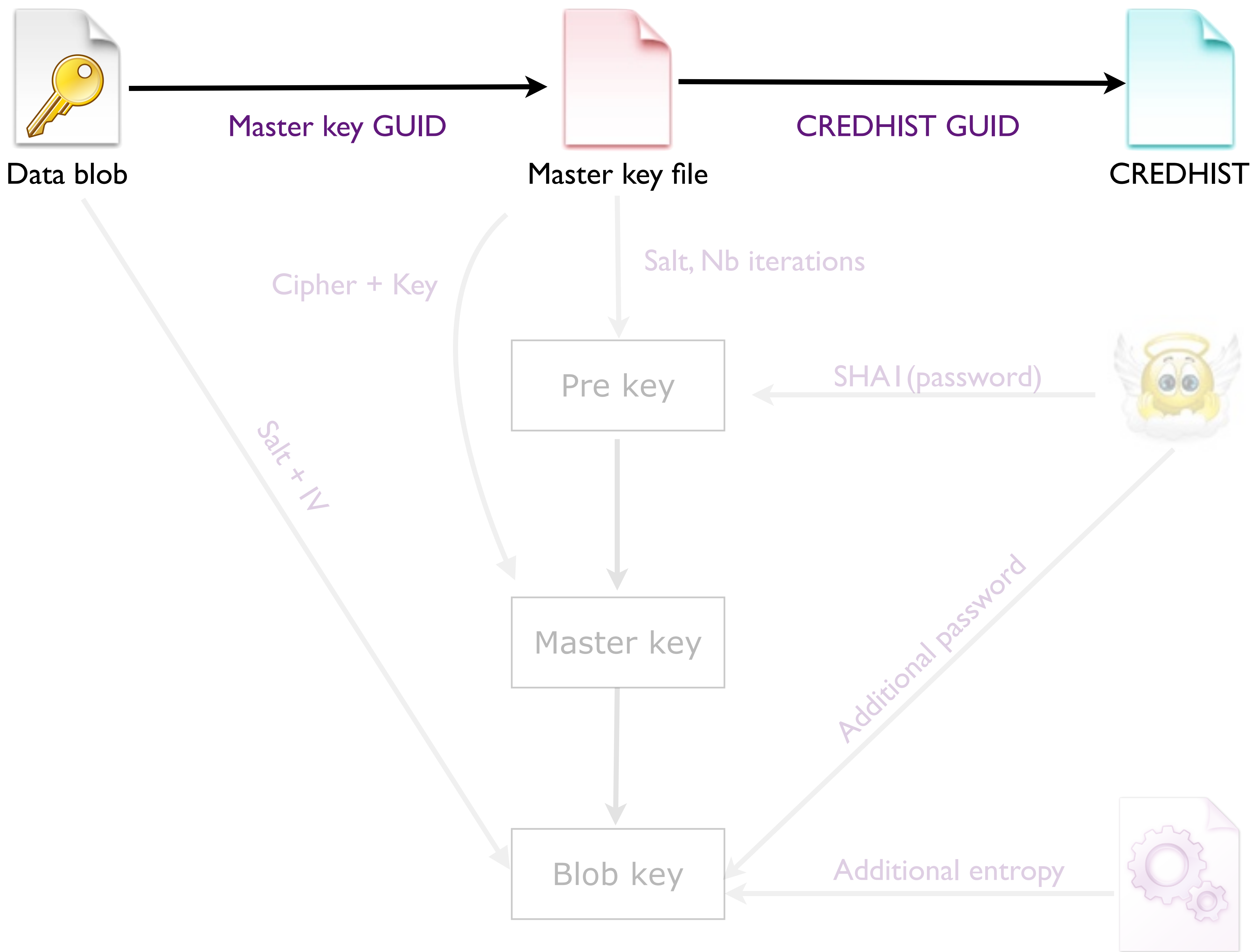
User password renewal

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

Decrypting a blob



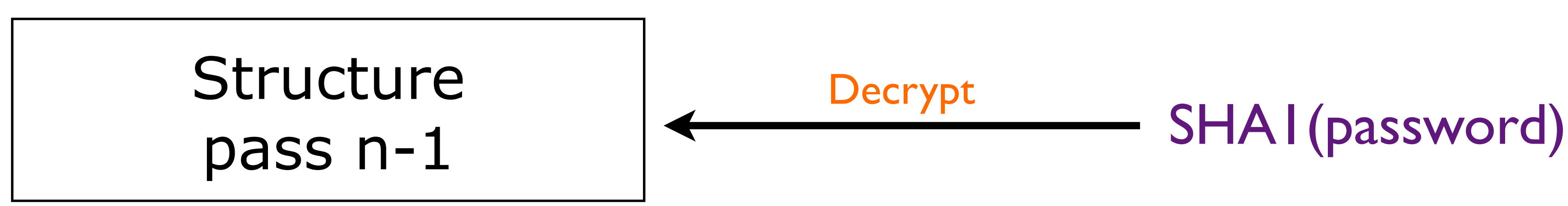
Decrypting a blob



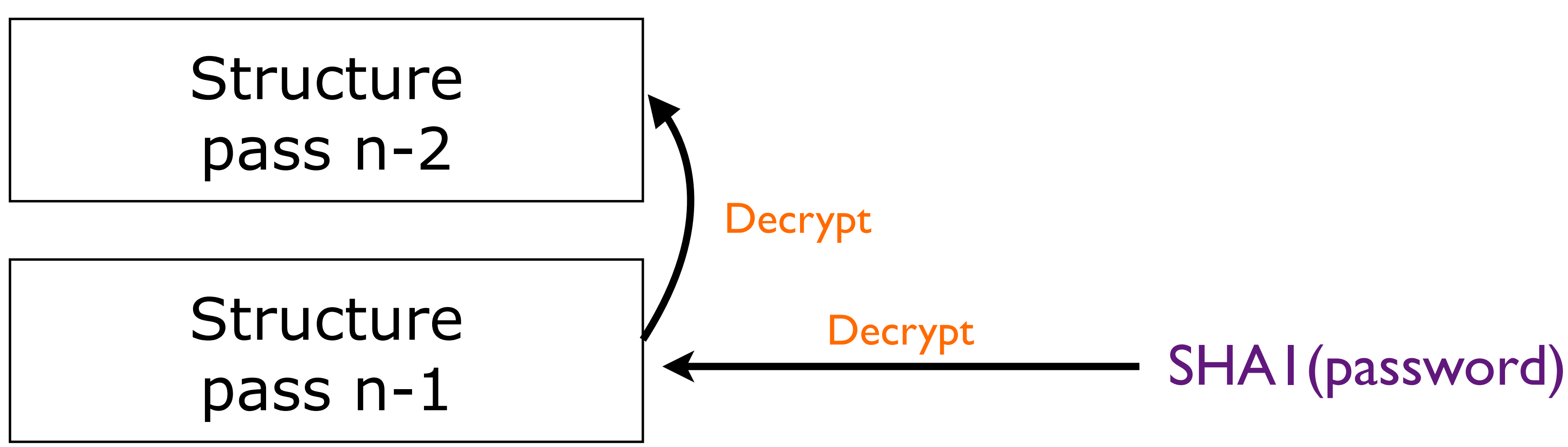
CREDHIST overview

SHA1(password)

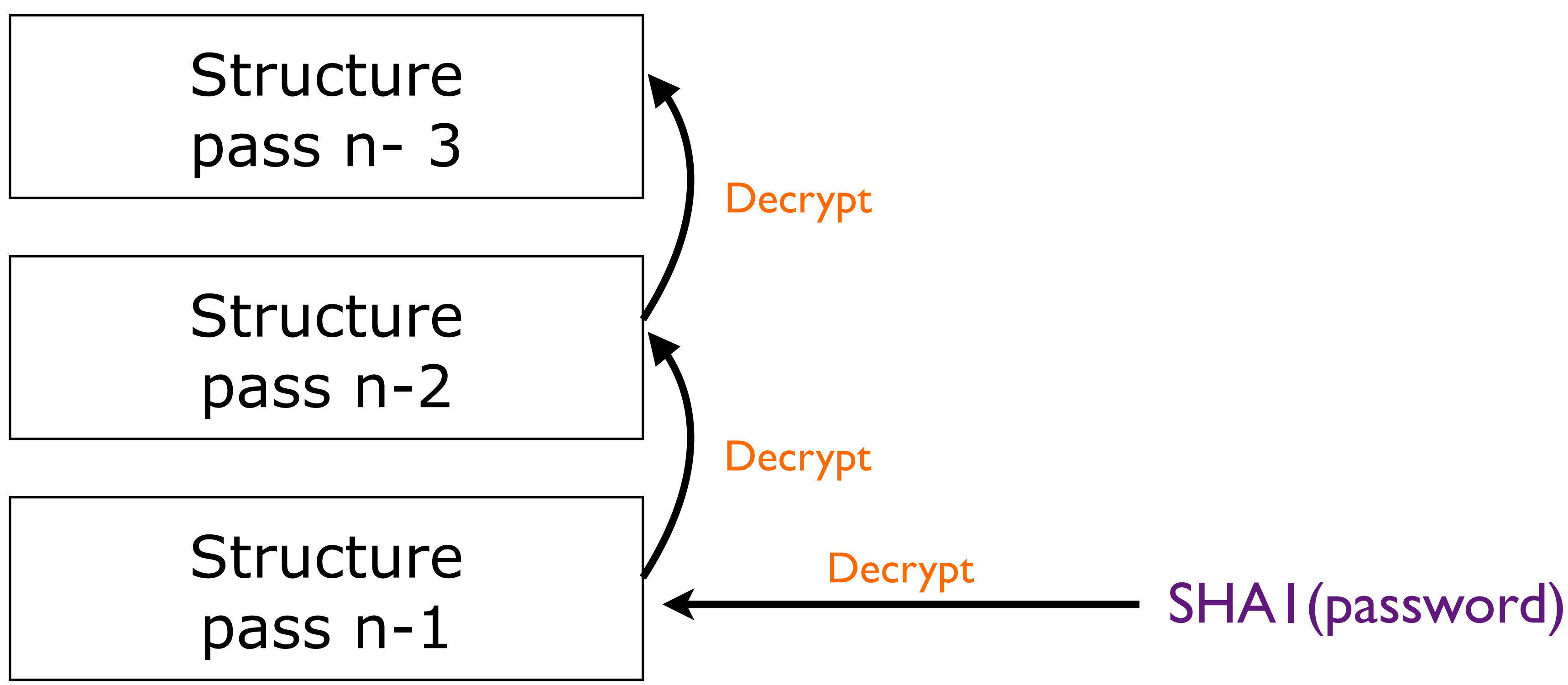
CREDHIST overview



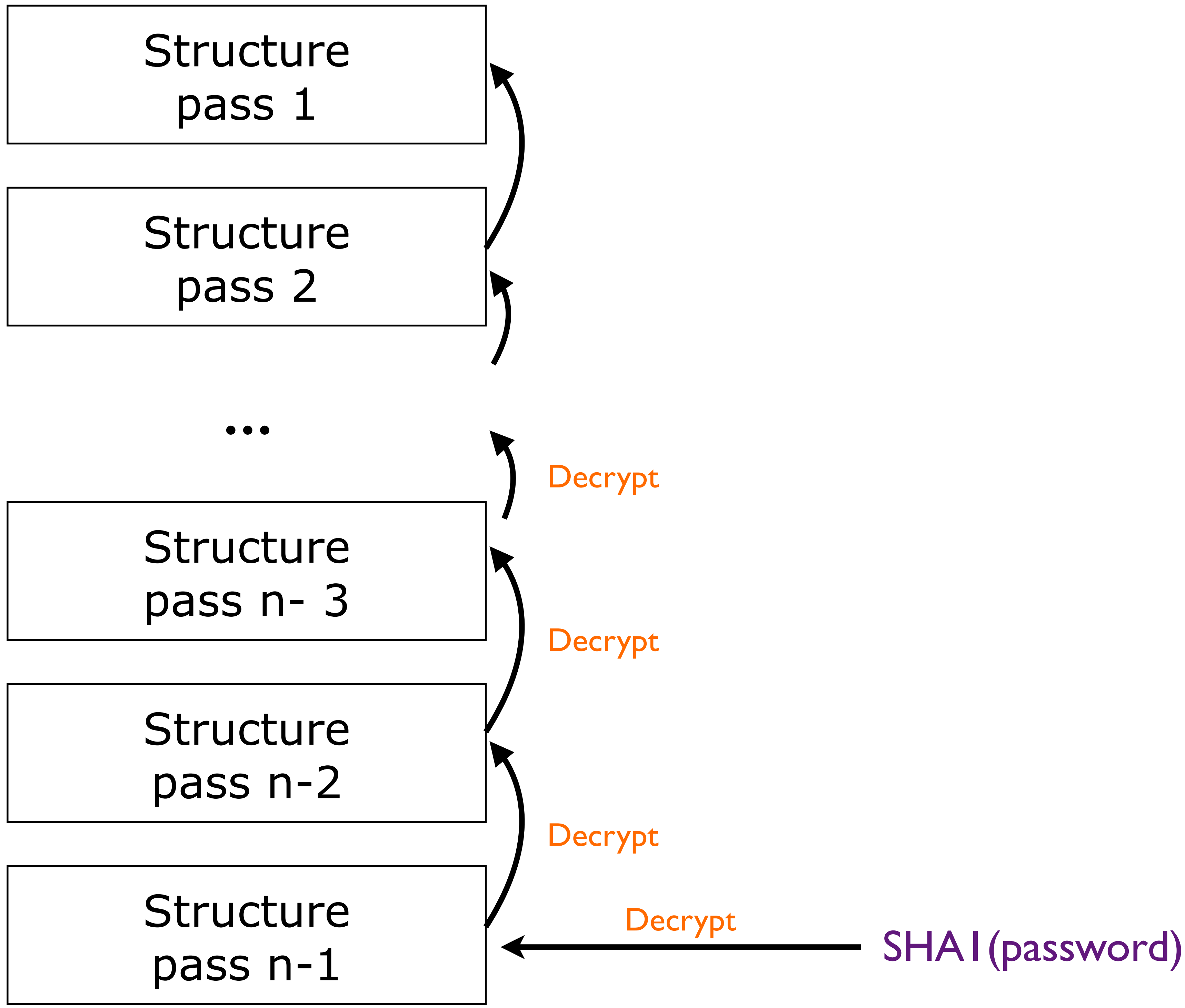
CREDHIST overview



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]; ⟵ Computer SID

dwAccountID;

bData[28];

bPasswordID[16]

CREDHIST entry structure main fields

idHashAlgo;

dwRounds;

dwCipherAlgo;

bSID[12];

dwComputerSID[3];

dwAccountID; ← Account ID

bData[28];

bPasswordID[16]

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

Warning

- DPAPIck is in ALPHA stage. Use it at your own risk ! You have been warned. It is just a POC
- Know bugs :
  - No HMAC checks -> No key check.
  - No Seven support, tested only on XP
  - No conditional entropy / strong password in UI
  - Don’t choose the correct master key by itself
  - Buffer overflows :)

DPAPIck future

- We made the choice to release early so you know we are telling the truth and everyone can start playing.
- We will provide a more robust version and eventually open the source code so one day Linux will read EFS files :)
- It just too soon for this.

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