Krypcak

Password Tool

FILE FORMATS SPECIFICATION

version 1.0

Zerosum Security
www.zerosumsecurity.nl

Contents

1 Overview

This document describes the specific file formats as used by the Krypcak Password Tool. Krypcak assists in managing and using your passwords.

1.1 Document history

Version	Date	Comments
1.0	2013-12-26	Initial release

1.2 References

[RFC4648]	S. Josefsson - The Base16, Base32, and Base64 Data Encodings. Dated October 2006.
[PHC]	Password Hashing Competition. See https://password-hashing.net
[ZS13a]	Zerosum Security – Krypcak Source code. Available at https://github.com/zerosumsecurity/Krypcak-for-windows
[ZS13b]	Zerosum Security – Krypcak Password Tool / Cryptographic Specification. Available at www.zerosumsecurity.nl/krypcak.

2 Generic structure

Krypcak uses two dedicated file types:

- Notebooks (with .knb extension), in which passwords are stored;
- Encrypted files (with .kef extension).

Both these file types contain encrypted and authenticated data.

Krypcak can also be used to encrypt messages. All these three forms of encrypted and authenticated data (in notebooks, encrypted files and encrypted messages) follow the same generic structure, as detailed in Table 1.

Size (in bytes)	Content
4	File type identifier (see §2.1)
4	Version information (see §2.2)
16	Salt (see §2.3)
4	Password check (see §2.4)
Variable	Encrypted data
16	Message authentication code (see §2.5)

Table 1: Structure of encrypted and authenticated data

The size of the encrypted and authenticated data is thus exactly 44 bytes larger than the size of the unencrypted data.

The first 28 bytes (consisting of the file type identifier, the version information, the salt and the password check) are called the header of the encrypted and authenticated data.

2.1 File type identifier

The four-byte identifier serves as a soft content signature that can be used as a first check to see if a file "looks like" a Krypcak file.

Type	(4-byte) Identifier (in hex)
Notebook	`5A534E42` (= "ZSNB" in ASCII)
Encrypted file	`5A534546` (= "ZSEF" in ASCII)
Encrypted message	`5A53454d` (= "ZSEM" in ASCII)

Table 2: Encrypted data identifiers

2.2 Version information

The four-byte information block is used to allow Krypcak to decide whether it can parse the file/message and if so, how this parsing should be done. Currently this information is not needed, but in the future some modifications to data encoding or pssword hashing may be devised and added to Krypcak.

The content of the version information block is as follows:

Byte #	Meaning
0	Content encoding scheme id
1	Password hashing scheme id
2	Krypcak application major version
3	Krypcak application minor version

Table 3: Version information

2.2.1 Content encoding scheme id

This byte indicates how the content is formatted. The byte 0x00 indicates the formats as defined in this document are used.

This may be extended in the future. For example, in encrypted message the plain data is represented in UTF-16 encoding. Perhaps for other applications, in which message expansion is an issue, a more efficient encoding such as UTF-8 might be preferable.

2.2.2 Password hashing scheme id

Currently only the password hashing scheme as defined in [ZS13b] is available. This scheme has id = 0x00.

Should the winner of the Password Hashing Competition (see [PHC]) be Keccak-based, we will include this scheme in Krypcak as well.

2.2.3 Major version

This indicates the major version of the Krypcak application used in the encryption process.

2.2.4 Minor version

This indicates the minor version of the Krypcak application used in the encryption process.

2.3 Salt

The salt is a 16-byte random value. The salt is used together with the user supplied password to perform the en-/decryption.

See [ZS13b] for details on how this encryption works, and how the pseudo-random number generator in Krypcak works.

2.4 Password check

The password check is a 4-byte value serving to check whether the password supplied for decryption is correct, without having to go through the entire decryption and MAC-verification process. This password check is derived from the password and the salt in acomputationally expensive way. See [ZS13b] for more details.

2.5 Message authentication code

The 16-byte message authenitcation code is computed over all content preceeding it. It serves as a check that the content has not been changed by anyone not knowing the correct password.

3 Notebooks

A password notebook is encrypted under the user password. The structure of an encrypted notebook is as defined in Table 1.

After decryption a notebook has the following structure:

Size (in bytes)	Content
4	Size of notebook (encrypted, as stored on disc)
4	Number of passwords in the notebook (N)
Variable	Password note #1
...	
Variable	Password note #N

Table 4: Content plain notebook

The maximum number of password notes that can be stored in a notebok is 8192.

3.1 Password notes

In a password note, the password is stored with some context data, relating to its date (of modification) and use.

A password note contains the following information:

Size (in bytes)	Content
4	Size content of password note
4	Type of password note identifier (see §3.1.1)
4	Integrity check password note (see §3.1.2)
4	Length of date field (= 0x0a000000)
Variable	Date (see §3.1.3)
4	Length of domain field
Variable	Domain (see §3.1.4)
4	Length of info field
Variable	Info (see §3.1.4)

4	Length of password
Variable	Password (see §3.1.5)
16	Associated salt (see §3.1.6)

Table 5: Content of a password note

3.1.1 Type of password

The following types of passwords are used in Krypcak (given with their corresponding identifiers):

Identifier (in hex)	Type of password
00000000	Passwords used to protect files
01000000	Passwords used to protect messages
02000000	Passwords that are shared with others
03000000	Passwords that are used as (part of) credentials to access an account (e.g. for a webresource or on a local computer)

Table 6: Different types of password notes

3.1.2 Integrity check

The integrity check over a password note is a 4-byte value, computed as a hash (see [ZS13b]) over the content of the password note. This value is checked on decrypting and parsing a notebook.

3.1.3 Date

The date field is exactly 10 characters long and is of the form “YYYY-MM-DD”. This date is automatically set on file or message encryption.

3.1.4 Domain and info field

The domain and info field as stored in the password note provide the context for the domain of use for the password. Per type of password the following information is stored in these two fields:

Password type	Domain	Info
File	Filename	N/A
Message	Message id	N/A
Shared	Contact name	N/A
Account	Domain	Username

Table 7: Content domain and info field

Both the domain and the info field are restricted to 255 characters in size.

3.1.5 Password

There are no restrictions on the characters used in password. The passwords are restricted to 255 characters in size.

3.1.6 Salt

When a file or message is encrypted with Krypcak, and the password is subsequently stored in the loaded notebook, the salt used in the encryption process is also stored inside the resulting password note. This helps Krypcak when decrypting the same file later on – it finds a password note with the same salt as present in the encrypted message/file. In this case it will automatically try to decrypt the message/file with the corresponding password, without the user having to enter a password or select a password from his notebook. The main use case for this is when a user encrypts a message/file which he/she later needs to decrypt himself.

In all other cases, the salt field is filled with 16 zero-bytes.

3.2 Example

Consider the following hexadecimal content of the encrypted notebook “xkcd.knb”:

```
5A534E4200000100537B62757C12621F8400D245DA6CEFBC17BA0913B288F40789D5378715006B0A
6911C192F7DF0DA238FFB54E7AF79A89FABCD19DB1261865BE6E469DEBD6D34476052A19B4FDB763
1D1666FC5338CC054DC0C0B8D0A059B436FC3A01DB94948EA59F1E8411265E90A45B67925890DCB4
3C7B2859F2AF2193708C9B2B4B8F4D4F7FE59B334F9FB8E6A11A515F893938F4B0B2EB951436814D
D017F80459A825A7FDCE68C03088E62D7BADEB7145BDE50D24A148EEF3786112DB93206FB73941EF
8D4C9A4E9391B74082347A5C3EFCEC8C6EDE6D434EBBCB0048D781A9A5951BD398E7C49C7EA134CD
B8821D7C838A17FDFC3C28018357D4B5442CBF6C5AC428AC5228295775913C39B39709053D2CA739
64963B839CD4F95CB53994392179686EE9B1C386D94F705D564E909B65B62C3293E78A768E1E0E5F
D694D27B37D2ED7B8BE89A359CA8A8A1F346141E6BF2ABEB67108CD1A5EB26BAD43BB391CBE884C2
514A873F391E762E1158803F44EF1C8336CBB2B90A3E048375F91F8C3F12DC72359BB568D43C3726
75D6373AFF872161A2B0C3BE741D20BF888586C0FC8C221461942E4F6DEED9B5E89178DB9263A05E
63AD1D3C53F51AF1FFA47FD42F23995E35D0797179FB062C4D7C99BB4B1D76FBC379D80A38D566C5
708F8DAE3F3E7D800F266193CEA6FC6B8762116F2F9AE7AA
```

Following Table 1 we can parse this data as:

Value (in hex)	Content
`5A534E42`	File type identifier = “ZSNB”
`00000100`	Version information
`537B62757C12621F8400D245DA6CEFBC`	Salt
`17BA0913`	Password check

`B288F40789D5378715006B0A6911C192`
`...`
`79FB062C4D7C99BB4B1D76FBC379D80A`
`38D566C5708F8DAE3F3E7D80`
	Encrypted data

`0F266193CEA6FC6B8762116F2F9AE7AA`	Message authentication code

Table 8: Content of example encrypted notebook

Decrypting the notebook with the password “CORRECT HORSE BATTERY STAPLE” gives the following data (in hex):

```
C4010000040000007800000000000000B87E8FB50A00000032003000310033002D00310031002D00
310034000C00000074006500730074002E0074007800740002E006B00650066000000000010000000
4F0057006C00650055004D0032007000720036006F007A00760045006C006E003E194A99600D1DFF
F3A31CF98E30EED26A00000001000000F8EA24800A00000032003000310033002D00310031002D00
310034000500000030003400390033003600000000001000000066006D00370039006A0058003000
4300490044006900510065006F0062003100DD6BA60FDF4625EB02A175DDCEDF6703680000000200
00004617C4870A00000032003000310033002D00310031002D003100310004000000480061006E00
7300000000001000000049007700770066006F00610041005A005600660042007700560043003200
51000000000000000000000000000000000007A00000003000000D95B764D0A000000320030003100
33002D00310031002D00310031000900000076006D002D007500620075006E007400750004000000
72006F006F007400100000004C007800390034004A00330069007200630052003600330073004900
61004F0000000000000000000000000000000000
```

Following Table 4 his can be parsed as follows:

Value (in hex)	Content
`C4010000`	Size of combined password notes: 452 bytes
`04000000`	Number of password notes in the notebook: 4
`78000000`	Size of 1st password note: 120 bytes
`00000000`	Type of password note: “File”
`B87E8FB5`	Integrity check on password note
`0A000000`	Length of date field: 10 chars
`32003000310033002D00310031002D0031003400`	Date: “2013-11-14”
`0C000000`	Length of domain field: 12 chars
`74006500730074002E007400780074002E006B0065006600`	Domain: “test.txt.kef”
`00000000`	Length of info field (empty info field)
`10000000`	Length of password: 16 chars
`4F0057006C00650055004D00320070007200360006F007A00760045006C006E00`	Password: “OWleUM2pr6ozvEln”
`3E194A99600D1DFFF3A31CF98E30EED2`	Salt
`6A000000`	Size of 2nd password note: 106 bytes
`01000000`	Type of password note: “Message”
`F8EA2480`	Integrity check on password note
`0A000000`	Length of date field
`32003000310033002D00310031002D0031003400`	Date: “2013-11-14”

`05000000`	Length of domain field
`30003400390033003600`	Domain: “04936”
`00000000`	Length of info field (empty info field)
`10000000`	Length of password (16)
`66006D00370039006A0058003000430049004400 6900510065006F0062003100`	Password: “fm79jX0CIDiQeob1”
`DD6BA60FDF4625EB02A175DDCEDF6703`	Salt
`68000000`	Size of 3rd password note: 104 bytes
`02000000`	Type of password note: “Shared”
`4617C487`	Integrity check on password note
`0A000000`	Length of date field: 10
`320030003100330002D00310031002D0031003100`	Date: “2013-11-11”
`04000000`	Length of domain field: 4
`480061006E007300`	Domain: “Hans”
`00000000`	Length of info field (empty info field)
`10000000`	Length of password: 16
`49007700770066006F00610041005A0056006600 420077005600430032005100`	Password: “IwwfoaAZVfBwVC2Q”
`00000000000000000000000000000000`	Salt
`7A000000`	Size of 3rd password note: 122 bytes
`03000000`	Type of password note: “Account”
`D95B764D`	Integrity check on password note
`0A000000`	Length of date field: 10
`320030003100330002D00310031002D0031003100`	Date: “2013-11-11”
`09000000`	Length of domain field: 9
`76006D002D007500620075006E0074007500`	Domain: “vm-ubuntu”
`04000000`	Length of info field: 4
`72006F006F007400`	Info: “root”
`10000000`	Length of password: 16
`4C00780039003400 4A003300690072006300520036003300730049006100 4F00`	Password: “Lx94J3ircR63sIaO”

`00000000000000000000000000000000`	Salt

Table 9: Content of example parsed notebook

4 Encrypted files

Files encrypted with Krypcak follow precisely the structure as defined in Table 1. The encrypted file gets the .kef extension and is exactly 44 bytes larger than the plain file.

4.1 Example

Consider the following hexadecimal content of the encrypted file "text.txt.kef":

```
5A53454600000100 3E194A99600D1DFFF3A31CF98E30EED2DC9E584D5518C6F07AA26587D52B4123
0CE0F723457A9668
```

Following Table 1 we can parse this data as:

Value (in hex)	Content
`5A534546`	File type identifier = "ZSEF"
`00000100`	Version information
`3E194A99600D1DFFF3A31CF98E30EED2`	Salt
`DC9E584D`	Password check
`5518C6F0`	Encrypted data
`7AA26587D52B41230CE0F723457A9668`	Message authentication code

Table 10: Content of example encrypted file

Decrypting this with the password "OWleUM2pr6ozvEln" gives (in hex) `74657374` (= "test" in ASCII) as content of the file "test.txt".

5 Encrypted messages

When Krypcak encrypts a message, it first transforms the characters to a raw byte encoding. These bytes are then encrypted according to the way represented in Table 1.

This encrypted and authenticated data is transformed via a Base64-encoding (see §5.1) into human-readable text. Finally a header, message index and footer are added (see §5.2).

5.1 Base64 encoding

For the Base64 encoding we employ a slightly different alphabet then generally used and defined in e.g. [RFC4648]. The reasons are as follows:

- the '+' character serves as a word-break in the MFC CEdit controls, which are used in the (current) Krypcak Windows GUI;

- as we are already using a different alphabet because of the reason above, we choose the alphabet such that an encrypted (and Base64-encoded) message always start with "KRYP>". Note the first five characters of the Base64-encoded encrypted message are completely determined by the first four bytes of the raw encrypted message. These four bytes are constant and defined in Table 2.

Instead of the two characters '+' and '/' we use the two characters '>' and '<'. The complete alphabet is then:

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	l	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	>	36	k	53	1
3	D	20	U	37	R	54	2
4	E	21	V	38	m	55	3
5	P	22	K	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	N	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	W	27	b	44	s	61	9
11	L	28	c	45	t	62	<
12	M	29	d	46	u	63	T
13	Y	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	F	32	g	49	x		
16	Q	33	h	50	y		

After each 64 characters a newline is inserted in the Base64 encoded text.

5.2 Header, message id and footer

The Base64 encoded message is preceded by the header

```
--BEGIN KRYPCAK ENCRYPTED MESSAGE--
```

and the 5-digit message id. This message id is randomly generated on message encryption. It's sole purpose is to serve as a reference that can be stored with password in the Krypcak notebook.

Finally the message is followed by the footer

```
---END KRYPCAK ENCRYPTED MESSAGE---
```

5.3 Data expansion

An encrypted message is a lot larger than the original unencrypted message. This is because of the following three reasons:

- internally the message is encoded in Unicode, taking up two bytes per character;
- the encryption adds a total of 44 bytes of overhead (as explained in §2);
- during Base64 encoding, every three bytes is encoded into four characters, with potentially one or two padding characters ('=') at the end.

For example, take a message consisting of 37 characters. The Unicode, encoding of this message takes up 74 bytes. Encryption expands this to 118 bytes. Applying Base64 encoding then expands this even further to 160 characters (including two padding characters).

5.4 Example

Consider the following encrypted message:

```
--BEGIN KRYPCAK ENCRYPTED MESSAGE--
Message id: 04936

KRYP>QAAAQDda6NF30NR6wWhdd3O32cDHTPL9K1ckevvmgLNwOTEPMZh>l2ziw00
C31SVLBkZXi1H7gjqO>sA0AtwXUXZ3dvSZpAWPNqbVWNuQFyNX>Osf9QbtZO1M5L
gmHjVh64Qi5LiiqNwgWA26hOslyfHw==

---END KRYPCAK ENCRYPTED MESSAGE---
```

Removing the Base64 encoding according to the table given in §5.1 gives:

```
5A53454D00000100DD6BA60FDF4625EB02A175DDCEDF67031FF14BF56D5C91EBEF9A02D8C0EFC414
C6614D1DB38B0D340B7D5254B0646578B51FB823A8E4EC03402DC1751767776F499A4028562A6D52
98B903F26174CEB1FF506ED64ED4CE4B8261E3561EB8422E4B8A2A98C20280DBA84EB11C9F1F
```

Following Table 1 we can parse this data as:

Value	Content
5A53454D	File type identifier = "ZSEM"
00000100	Version information
DD6BA60FDF4625EB02A175DDCEDF6703	Salt
1FF14BF5	Password check
6D5C91EBEF9A02D8C0EFC414C6614D1DB38B0D340B7D5254B0646578B51FB823A8E4EC03402DC1751767776F499A4028562A6D5298B903F26174CEB1FF506ED64ED4CE4B8261E3561EB8	Encrypted data

`422E4B8A2A98C20280DBA84EB11C9F1F`	Message authentication code

Table 11: Content of example encrypted message

Decrypting with the password “fm79jX0CIDiQeob1” gives (in hex):

```
540068006500200006D0061006700690063002000770006F00720064007300200061007200650020000
2200420065006C006700690061006E002000570061006600660065006C0073002200
```

or “The magic words are Belgian Waffels”.