PdfParser Design

# Pseudocode

## “Encrypt” Object in PDF File

/Filter/Standard

/V 2

/R 3

/P -1852

/Length 128

/O <3d7d8221e188c1d85e2721cf40ec31aa9c8c38de9e1f77bababcc787ef97a501>

/U <f15047b3b0b50546521d3d8c14247e7a00000000000000000000000000000000>>>

## *Algo7: isValid* AuthenticateOwnerPassword(*OwnerPassword*)

*// Algorithm 7: Authenticating the owner password*

*// 7a) Compute an encryption key from the supplied password string*

hashedOwnerPassword = **CalculatePasswordHash**(OwnerPassword)

*//7b) Do the following 20 times: Decrypt the value of the encryption dictionary’s O entry (first iteration) or the output from the previous iteration (all subsequent iterations), using an RC4 encryption function with a different encryption key at each iteration. The key shall be generated by taking the original key (obtained in step (a)) and performing an XOR (exclusive or) operation between each byte of the key and the single-byte value of the iteration counter (from 19 to 0).*

rc4Data = Encrypt.O

for iteration=0 to 19

for byteIndex=0 to Encrypt.Length

rc4Key[byteIndex] = (byte)(hashedOwnerPassword[byteIndex] ^ iteration)

rc4Data = RC4.Encrypt(rc4Key, rc4Data)

userPassword = rc4DataBytes

*//7c) The result of step (b) purports to be the user password. Authenticate this user password using "Algorithm*

*6: Authenticating the user password". If it is correct, the password supplied is the correct owner password.*

return **authenticateUserPassword**(userPassword)

## *Algo3: hashedPassword* CalculatePasswordHash(*password*)

*// Algorithm 3: Computing the encryption dictionary’s O value*

*// 3*a) Pad or truncate the owner password string

paddedPassword = **PadPassword**(password)

*// 3b) ) Initialize the MD5 hash function and pass the result of step (a) as input to this function.*

hashedPassword = MD5.Hash(paddedPassword)

*// 3c) Do the following 50 times: Take the output from the previous MD5 hash and pass it as input into a new MD5 hash.*

for i=1 to 50

hashedPassword = MD5.Hash(hashedPassword[0..Encrypt.Length])

*//3d) Create an RC4 encryption key using the first Encrypt.Length bytes of the output from the final MD5 hash.*

return hashedPassword[0..Encrypt.Length]

## *Algo6: isValid* AuthenticateUserPassword(*userPassword*)

*// Algorithm 6: Authenticating the user password*

*// a) Perform all but the last step of "Algorithm 5: Computing the encryption dictionary’s U (user password) value" using the supplied password string.*

newEncryptU32 = **CalculateEncryptU**(userPassword)

*// b) If the result of step (a) is equal to the value of the encryption dictionary’s U entry (comparing on the first 16 bytes in the case of security handlers of revision 3 or greater), the password supplied is the correct user password. The key obtained in step (a) (that is, in the first step of "Algorithm 5: Computing the encryption dictionary’s U (user password) value") shall be used to decrypt the document.*

if (newEncryptU32[0..16]==EncryptU[0..16]) then

globalEncryptionKey = newEncryptU32

## *Algo5: EncryptU32* CalculateEncryptU(*userPassword*)

*// Algorithm 5: Computing the encryption dictionary’s U (user password) value (Security handlers of revision 3 or greater)*

*// a) Create an encryption key based on the user password string, as described in "Algorithm 2: Computing an encryption key".*

userEncryptionKey = **CalculateEncryptionKey**(userPassword)

*// b) Initialize the MD5 hash function and pass the 32-byte padding string shown in step (a) of "Algorithm 2: Computing an encryption key" as input to this function.*

*// c) Pass the first element of the file’s file identifier array (the value of the ID entry in the document’s trailer dictionary; see Table 15) to the hash function and finish the hash.*

*// d) Encrypt the 16-byte result of the hash, using an RC4 encryption function with the encryption key from step (a).*

*newEncryptU32 =* RC4.Encrypt(key: userEncryptionKey, data: **PadPassword**(userPassword)+ Trailer.ID)

*// e) Do the following 19 times: Take the output from the previous invocation of the RC4 function and pass it as input to a new invocation of the function; use an encryption key generated by taking each byte of the original encryption key obtained in step (a) and performing an XOR (exclusive or) operation between that byte and the single-byte value of the iteration counter (from 1 to 19).*

for iteration=1 to 19

for byteIndex=0 to userEncryptionKey.Length - 1

rC4Key [byteIndex] = userEncryptionKey[byteIndex] ^ iteration

new*EncryptU32 =* RC4.Encrypt(key: rC4Key, data: new*EncryptU32*)

*// f) Append 16 bytes of arbitrary padding to the output from the final invocation of the RC4 function and store the 32-byte result as the value of the U entry in the encryption dictionary.*

return new*EncryptU32[0..16] + [0, 0, … 0] //16 bytes with value 0*

## Algo2: *encryptionKey* CalculateEncryptionKey(*password*)

*// Algorithm 2: Computing an encryption key*

*// a) Pad or truncate the password string to exactly 32 bytes.*

padPassword = **PadPassword**(password)

*// b) Initialize the MD5 hash function and pass the result of step (a) as input to this function.*

*// c) Pass the value of the encryption dictionary’s O entry to the MD5 hash function. ("Algorithm 3: Computing the encryption dictionary’s O (owner password) value" shows how the O value is computed.)*

*// d) Convert the integer value of the P entry to a 32-bit unsigned binary number and pass these bytes to the MD5 hash function, low-order byte first.*

*// e) Pass the first element of the file’s file identifier array (the value of the ID entry in the document’s trailer dictionary; see Table 15) to the MD5 hash function.*

*NOTE The first element of the ID array generally remains the same for a given document. However, in some situations, conforming writers may regenerate the ID array if a new generation of a document is created. Security handlers are encouraged not to rely on the ID in the encryption key computation.*

*// f) (Security handlers of revision 4 or greater) If document metadata is not being encrypted, pass 4 bytes with the value 0xFFFFFFFF to the MD5 hash function.*

*// g) Finish the hash.*

encryptionKey=MD5.Hash(padPassword+EncryptO+P+ID[0])

*// h) Do the following 50 times: Take the output from the previous MD5 hash and pass the first n bytes of the output as input into a new MD5 hash, where n is the number of bytes of the encryption key as defined by the value of the encryption dictionary’s Length entry.*

For iteration=1 to 50

encryptionKey=MD5.Hash(encryptionKey[0.Encrypt.Length])

*// i) Set the encryption key to the first n bytes of the output from the final MD5 hash, where n shall always be 5 for security handlers of revision 2 but, for security handlers of revision 3 or greater, shall depend on the value of the encryption dictionary’s Length entry.*

return encryptionKey[0.Encrypt.Length]

## Algo2a: *paddedPassword* PadPassword(*password*)

*// Algorithm 2: Computing an encryption key*

*// a) Pad or truncate the password string to exactly 32 bytes. If the password string is more than 32 bytes long, use only its first 32 bytes; if it is less than 32 bytes long, pad it by appending the required number of additional bytes from the beginning of the following padding string:*

*< 28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08 2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A >*

*That is, if the password string is n bytes long, append the first 32 - n bytes of the padding string to the end of the password string. If the password string is empty (zero-length), meaning there is no user password, substitute the entire padding string in its place.*

var paddedBytes = new byte[32]

var byteCount = Math.Min(password.Length, 32)

paddedBytes[0..byteCount] = password[0..byteCount]

if (byteCount<32)

paddedBytes[byteCount..32] = password[byteCount..32]

return paddedBytes

# PDF Specification

## Algorithm 7: Authenticating the owner password

a) Compute an encryption key from the supplied password string, as described in steps (a) to (d) of "Algorithm 3: Computing the encryption dictionary’s O (owner password) value".

-----------------------------------------------------------------

### Algorithm 3: Computing the encryption dictionary’s O (owner password) value

a) Pad or truncate the owner password string as described in step (a) of "Algorithm 2: Computing an encryption key". If there is no owner password, use the user password instead.

b) Initialize the MD5 hash function and pass the result of step (a) as input to this function.

c) (Security handlers of revision 3 or greater) Do the following 50 times: Take the output from the previous MD5 hash and pass it as input into a new MD5 hash.

d) Create an RC4 encryption key using the first n bytes of the output from the final MD5 hash, where n shall always be 5 for security handlers of revision 2 but, for security handlers of revision 3 or greater, shall depend on the value of the encryption dictionary’s Length entry.

-----------------------------------------------------------------

b) (Security handlers of revision 2 only) Decrypt the value of the encryption dictionary’s O entry, using an RC4 encryption function with the encryption key computed in step (a).

(Security handlers of revision 3 or greater) Do the following 20 times: Decrypt the value of the encryption dictionary’s O entry (first iteration) or the output from the previous iteration (all subsequent iterations), using an RC4 encryption function with a different encryption key at each iteration. The key shall be generated by taking the original key (obtained in step (a)) and performing an XOR (exclusive or) operation between each byte of the key and the single-byte value of the iteration counter (from 19 to 0).

c) The result of step (b) purports to be the user password. Authenticate this user password using "Algorithm 6: Authenticating the user password". If it is correct, the password supplied is the correct owner password.

-----------------------------------------------------------------

### Algorithm 6: Authenticating the user password

a) Perform all but the last step of "Algorithm 4: Computing the encryption dictionary’s U (user password) value (Security handlers of revision 2)" or "Algorithm 5: Computing the encryption dictionary’s U (user password) value (Security handlers of revision 3 or greater)" using the supplied password string.

-----------------------------------------------------------------

#### Algorithm 5: Computing the encryption dictionary’s U (user password) value (Security handlers of revision 3 or greater)

a) Create an encryption key based on the user password string, as described in "Algorithm 2: Computing an encryption key".

-----------------------------------------------------------------

##### Algorithm 2: Computing an encryption key

a) Pad or truncate the password string to exactly 32 bytes. If the password string is more than 32 bytes long, use only its first 32 bytes; if it is less than 32 bytes long, pad it by appending the required number of additional bytes from the beginning of the following padding string:

< 28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08 2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A >

That is, if the password string is n bytes long, append the first 32 - n bytes of the padding string to the end of the password string. If the password string is empty (zero-length), meaning there is no user password, substitute the entire padding string in its place.

b) Initialize the MD5 hash function and pass the result of step (a) as input to this function.

c) Pass the value of the encryption dictionary’s O entry to the MD5 hash function. ("Algorithm 3: Computing the encryption dictionary’s O (owner password) value" shows how the O value is computed.)

d) Convert the integer value of the P entry to a 32-bit unsigned binary number and pass these bytes to the MD5 hash function, low-order byte first.

e) Pass the first element of the file’s file identifier array (the value of the ID entry in the document’s trailer dictionary; see Table 15) to the MD5 hash function.

NOTE The first element of the ID array generally remains the same for a given document. However, in some situations, conforming writers may regenerate the ID array if a new generation of a document is created. Security handlers are encouraged not to rely on the ID in the encryption key computation.

f) (Security handlers of revision 4 or greater) If document metadata is not being encrypted, pass 4 bytes with the value 0xFFFFFFFF to the MD5 hash function.

g) Finish the hash.

h) (Security handlers of revision 3 or greater) Do the following 50 times: Take the output from the previous MD5 hash and pass the first n bytes of the output as input into a new MD5 hash, where n is the number of bytes of the encryption key as defined by the value of the encryption dictionary’s Length entry.

i) Set the encryption key to the first n bytes of the output from the final MD5 hash, where n shall always be 5 for security handlers of revision 2 but, for security handlers of revision 3 or greater, shall depend on the value of the encryption dictionary’s Length entry.

-----------------------------------------------------------------

b) Initialize the MD5 hash function and pass the 32-byte padding string shown in step (a) of "Algorithm 2: Computing an encryption key" as input to this function.

c) Pass the first element of the file’s file identifier array (the value of the ID entry in the document’s trailer dictionary; see Table 15) to the hash function and finish the hash.

d) Encrypt the 16-byte result of the hash, using an RC4 encryption function with the encryption key from step (a).

e) Do the following 19 times: Take the output from the previous invocation of the RC4 function and pass it asinput to a new invocation of the function; use an encryption key generated by taking each byte of the original encryption key obtained in step (a) and performing an XOR (exclusive or) operation between that byte and the single-byte value of the iteration counter (from 1 to 19).

f) Append 16 bytes of arbitrary padding to the output from the final invocation of the RC4 function and store the 32-byte result as the value of the U entry in the encryption dictionary.

-----------------------------------------------------------------

b) If the result of step (a) is equal to the value of the encryption dictionary’s U entry (comparing on the first 16 bytes in the case of security handlers of revision 3 or greater), the password supplied is the correct user password. The key obtained in step (a) (that is, in the first step of "Algorithm 4: Computing the encryption dictionary’s U (user password) value (Security handlers of revision 2)" or "Algorithm 5: Computing the encryption dictionary’s U (user password) value (Security handlers of revision 3 or greater)") shall be used to decrypt the document.

-----------------------------------------------------------------