#### documentProtection (Document Editing Restrictions)

This element specifies the set of document protection restrictions which have been applied to the contents of a WordprocessingML document. These restrictions shall be enforced by applications editing this document when the enforcement attribute is turned on, and should be ignored (but persisted) otherwise. Document protection is a set of restrictions used to prevent unintentional changes to all or part of a WordprocessingML document - since this protection does not encrypt the document, malicious applications may circumvent its use. This protection is not intended as a security feature and may be ignored.

If this element is omitted, then no protection shall be applied to this document.

When a password is supplied via an application which shall be hashed and stored in this element, that process shall be done in two stages:

First, the password shall be hashed using the following algorithm:

* Truncate the password to 15 characters.
* Construct a new NULL-terminated string consisting of single-byte characters:
* Get the single-byte values by iterating through the Unicode characters of the truncated password. For each character, if the low byte is not equal to 0, take it. Otherwise, take the high byte.
* From now on, the single-byte character string is used.
* If the password is empty, return 0.
* Compute the high-order word of the new key:
* Initialize from the initial code array (see below), depending on the password’s length. For each character in the password:
* For every bit in the character, starting with the least significant and progressing to (but excluding) the most significant, if the bit is set, XOR the key’s high-order word with the corresponding word from the encryption matrix
* Compute the low-order word of the new key:
* Initialize with 0
* For each character in the password, going backwards, low-order word = (((low-order word SHR 14) AND 0x0001) OR (low-order word SHL 1) AND 0x7FFF)) XOR character
* Lastly, low-order word = (((low-order word SHR 14) AND 0x0001) OR (low-order word SHL 1) AND 0x7FFF)) XOR password length XOR 0xCE4B.

Initial code array

The initial code array contains the initial values for the key’s high-order word. The initial value depends on the length of the password, as follows:

|  |  |
| --- | --- |
| Password length | Initial value for the key’s high-order word |
| 1 | 0xE1F0 |
| 2 | 0x1D0F |
| 3 | 0xCC9C |
| 4 | 0x84C0 |
| 5 | 0x110C |
| 6 | 0x0E10 |
| 7 | 0xF1CE |
| 8 | 0x313E |
| 9 | 0x1872 |
| 10 | 0xE139 |
| 11 | 0xD40F |
| 12 | 0x84F9 |
| 13 | 0x280C |
| 14 | 0xA96A |
| 15 | 0x4EC3 |

Encryption matrix

The encryption matrix contains codes used during the calculation of the key’s high-order word. As described in the algorithm above, for every bit of the password’s characters, if the bit is set, a corresponding value is taken from this encryption matrix and is used to XOR the key’s high-order word with it. Each row in the encryption matrix corresponds to a single character from the password, and each of the seven columns corresponds to a particular bit (0-6) in this character.

The values are taken in such a way so that the last character of the password uses the last row in the encryption matrix. The next-to-last character uses the next-to-last row in the matrix, and so on. This means that the beginning of the matrix may be unused, depending on the length of the password.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Bit 0 | Bit 1 | Bit 2 | Bit 3 | Bit 4 | Bit 5 | Bit 6 |
| Last-14 | 0xAEFC | 0x4DD9 | 0x9BB2 | 0x2745 | 0x4E8A | 0x9D14 | 0x2A09 |
| Last-13 | 0x7B61 | 0xF6C2 | 0xFDA5 | 0xEB6B | 0xC6F7 | 0x9DCF | 0x2BBF |
| Last-12 | 0x4563 | 0x8AC6 | 0x05AD | 0x0B5A | 0x16B4 | 0x2D68 | 0x5AD0 |
| Last-11 | 0x0375 | 0x06EA | 0x0DD4 | 0x1BA8 | 0x3750 | 0x6EA0 | 0xDD40 |
| Last-10 | 0xD849 | 0xA0B3 | 0x5147 | 0xA28E | 0x553D | 0xAA7A | 0x44D5 |
| Last-9 | 0x6F45 | 0xDE8A | 0xAD35 | 0x4A4B | 0x9496 | 0x390D | 0x721A |
| Last-8 | 0xEB23 | 0xC667 | 0x9CEF | 0x29FF | 0x53FE | 0xA7FC | 0x5FD9 |
| Last-7 | 0x47D3 | 0x8FA6 | 0x0F6D | 0x1EDA | 0x3DB4 | 0x7B68 | 0xF6D0 |
| Last-6 | 0xB861 | 0x60E3 | 0xC1C6 | 0x93AD | 0x377B | 0x6EF6 | 0xDDEC |
| Last-5 | 0x45A0 | 0x8B40 | 0x06A1 | 0x0D42 | 0x1A84 | 0x3508 | 0x6A10 |
| Last-4 | 0xAA51 | 0x4483 | 0x8906 | 0x022D | 0x045A | 0x08B4 | 0x1168 |
| Last-3 | 0x76B4 | 0xED68 | 0xCAF1 | 0x85C3 | 0x1BA7 | 0x374E | 0x6E9C |
| Last-2 | 0x3730 | 0x6E60 | 0xDCC0 | 0xA9A1 | 0x4363 | 0x86C6 | 0x1DAD |
| Last-1 | 0x3331 | 0x6662 | 0xCCC4 | 0x89A9 | 0x0373 | 0x06E6 | 0x0DCC |
| Last | 0x1021 | 0x2042 | 0x4084 | 0x8108 | 0x1231 | 0x2462 | 0x48C4 |

[Example: Consider a password which has been supplied - the string “Example”. It is already under 15 characters, so truncation does not affect it. It is then converted to a string of single-byte characters.

* The password is 7 characters long, so, from the initial code array, the initial value for the key’s high-order word is 0xF1CE.
* The key’s high-order word is then computed further depending on the password’s characters:
* The first character is ‘E’ (0x45). This is the first character of a 7-character password, so its corresponding row in the encryption matrix is “Last-6”.
* Bit 0 is set, therefore the key’s high-order word is combined (via XOR) with the corresponding value for Bit 0 on row “Last-6”, which is 0xB861. The new [result](result.docx) is 0xF1CE XOR 0xB861 = 0x49AF.
* Bit 2 is set, so the key’s high-order word is XOR-ed with the corresponding value for Bit 2 on row “Last-6”, which is 0xC1C6. The new [result](result.docx) is 0x49AF XOR 0xC1C6 = 0x8869.
* This process is repeated for each bit.
* The [next](next.docx) character is ‘x’ (0x78). Its corresponding row in the encryption matrix is “Last-5”.
* Bit 3 is set. The value for Bit 3 on row “Last-5” in the encryption matrix is 0x0D42. The current value for the key’s high-order byte is 0x5585, so the new one should be 0x5585 XOR 0x0D42 = 0x58C7.
* This process is repeated for each bit.
* This process is repeated for all characters.
* After the last character has been processed, the above step produced 0x64CE for the key’s high-order word. Now the low-order word needs to be calculated:
* The initial value is 0.
* It is then calculated using the password:
* The last character of the password is ‘e’ (0x65), so, by the formula, low-order word = (((low-order word SHR 14) AND 0x0001) OR ((low-order word SHL 1) AND 0x7FFF)) XOR ‘e’ = (((0 SHR 14) AND 0x0001) OR ((0 SHL 1) AND 0x7FFF)) XOR 0x65 = 0x0065.
* The [next](next.docx) to last character of the password is ‘l’ (0x6C). Again, by the formula, (((0x0065 SHR 14) AND 0x0001) OR ((0x0065 SHL 1) AND 0x7FFF)) XOR 0x6C = (0x0000 OR 0x00CA) XOR 0x6C = 0x00CA XOR 0x6C = 0x00A6.
* This process is repeated for each character.
* After the password’s first character has been processed, we have 0x1199 for the key’s low-order word. Lastly, the password’s length is combined into it: low-order word = (((0x1199 SHR 14) AND 0x0001) OR ((0x1199 SHL 1) AND 0x7FFF)) XOR 0x0007 XOR 0xCE4B = 0x2332 XOR 0x0007 XOR 0xCE4B = 0x2335 XOR 0xCE4B = 0xED7E.
* The end [result](result.docx) for the key is 0x64CEED7E.

end example]

[Rationale: This pre-processing step is necessary for compatibility with [legacy](legacy.docx) word processing applications which hashed their password solely using this mechanism. end rationale]

Second, the byte order of the [result](result.docx) shall be reversed [Example: 0x64CEED7E becomes 7EEDCE64. end example], and that value shall be hashed as defined by the attribute values.

[Note: The algorithm above can be stated as follows using diagrams:





end note]

[Example: Consider a WordprocessingML document which specifies that applications shall not allow any modifications to this document other than the addition of comments. This requirement would be specified using the following WordprocessingML in the document settings:

<w:documentProtection w:edit="[comments](comments.docx)" w:enforcement="true" …
 w:cryptAlgorithmClass="hash" w:cryptAlgorithmType="typeAny"
 w:cryptAlgorithmSid="1" w:hash="9oN7nWkCAyEZib1RomSJTjmPpCY=" />

The documentProtection element has an edit attribute value of [comments](comments.docx), specifying that the only modification allowed should be [comments](comments.docx), the enforcement attribute has a value of true, specifying that the document protection specified is to be enforced on the given document. Finally, in order for the hosting application to stop enforcement of the document protection applied to the document, the hosting application would have to be provided with a password that the hosting application would then hash, compare to the value of the hash attribute (9oN7nWkCAyEZib1RomSJTjmPpCY=), and if the two values matched, halt enforcement of any document protection. end example]

|  |
| --- |
| Parent Elements |
| settings (§) |

|  |  |
| --- | --- |
| Attributes | Description |
| algIdExt (Cryptographic Algorithm Extensibility) | Specifies that a cryptographic algorithm which was not defined by this Office Open [XML](XML.docx) Standard has been used to generate the hash value stored with this document.This value, when present, shall be interpreted based on the value of the algIdExtSource attribute in order to determine the algorithm used, which shall be application-defined. [Rationale: This extensibility affords the fact that with exponentially increasing computing power, documents created in the future will likely need to utilize as yet undefined hashing algorithms in order to remain secure. end rationale]If this value is present, the cryptAlgorithmClass, cryptAlgorithmType, and cryptAlgorithmSid attribute values shall be ignored in favor of the algorithm defined by this attribute. [Example: Consider a WordprocessingML document with the following information stored in one of its protection elements:<w:… w:algIdExt="0000000A"  w:algIdExtSource="futureCryptography" w:hash="9oN7nWkCAyEZib1RomSJTjmPpCY=" />The algIdExt attribute value of 0000000A specifies that the algorithm with hex code A shall be used as defined by the futureCryptography application. end example]The possible values for this attribute are defined by the [ST\_LongHexNumber](ST_LongHexNumber.docx) simple [type](type.docx) (§). |
| algIdExtSource (Algorithm Extensibility Source) | Specifies the application which defined the algorithm value specified by the algIdExt attribute.[Example: Consider a WordprocessingML document with the following information stored in one of its protection elements:<w:… w:algIdExt="0000000A"  w:algIdExtSource="futureCryptography" w:hash="9oN7nWkCAyEZib1RomSJTjmPpCY=" />The algIdExtSource attribute value of futureCryptography specifies that the algorithm used here was published by the futureCryptography application. end example]The possible values for this attribute are defined by the [ST\_String](ST_String.docx) simple [type](type.docx) (§). |
| cryptAlgorithmClass (Cryptographic Algorithm Class) | Specifies the class of cryptographic algorithm used by this protection. [Note: The initial version of this Office Open [XML](XML.docx) Standard only supports a single version - hash - but future versions may expand this as necessary. end note][Example: Consider a WordprocessingML document with the following information stored in one of its protection elements:<w:… w:cryptAlgorithmClass="hash"  w:cryptAlgorithmType="typeAny"  w:cryptAlgorithmSid="1" w:hash="9oN7nWkCAyEZib1RomSJTjmPpCY=" />The cryptAlgorithmClass attribute value of hash specifies that the algorithm used for the password is a hashing algorithm. end example]The possible values for this attribute are defined by the [ST\_AlgClass](ST_AlgClass.docx) simple [type](type.docx) (§). |
| cryptAlgorithmSid (Cryptographic Hashing Algorithm) | Specifies the specific cryptographic hashing algorithm which shall be used along with the salt attribute and user-supplied password in order to compute a hash value for comparison.The possible values for this attribute shall be interpreted as follows:

|  |  |
| --- | --- |
| Value | Algorithm |
| 1 | MD2 |
| 2 | MD4 |
| 3 | MD5 |
| 4 | SHA-1 |
| 5 | MAC |
| 6 | RIPEMD |
| 7 | RIPEMD-160 |
| 8 | Undefined. Shall not be used. |
| 9 | HMAC |
| 10 | Undefined. Shall not be used. |
| 11 | Undefined. Shall not be used. |
| 12 | SHA-256 |
| 13 | SHA-384 |
| 14 | SHA-512 |
| Any other value | Undefined. Shall not be used. |

[Example: Consider a WordprocessingML document with the following information stored in one of its protection elements:<w:… w:cryptAlgorithmClass="hash"  w:cryptAlgorithmType="typeAny"  w:cryptAlgorithmSid="1" w:hash="9oN7nWkCAyEZib1RomSJTjmPpCY=" />The cryptAlgorithmSid attribute value of 1 specifies that the SHA-1 hashing algorithm shall be used to generate a hash from the user-defined password. end example]The possible values for this attribute are defined by the [ST\_DecimalNumber](ST_DecimalNumber.docx) simple [type](type.docx) (§). |
| cryptAlgorithmType (Cryptographic Algorithm Type) | Specifies the [type](type.docx) of cryptographic algorithm used by this protection. [Note: The initial version of this Office Open [XML](XML.docx) Standard only supports a single [type](type.docx) - typeAny - but future versions may expand this as necessary. end note][Example: Consider a WordprocessingML document with the following information stored in one of its protection elements:<w:… w:cryptAlgorithmClass="hash"  w:cryptAlgorithmType="typeAny"  w:cryptAlgorithmSid="1" w:hash="9oN7nWkCAyEZib1RomSJTjmPpCY=" />The cryptAlgorithmType attribute value of typeAny specifies that any [type](type.docx) of algorithm may have been used for the password. end example]The possible values for this attribute are defined by the [ST\_AlgType](ST_AlgType.docx) simple [type](type.docx) (§). |
| cryptProvider (Cryptographic Provider) | Specifies the cryptographic provider which was used to generate the hash value stored in this document. If the user provided a cryptographic provider which was not the system's built-in provider, then that provider shall be stored here so it can subsequently be used if available.If this attribute is omitted, then the built-in cryptographic provider on the system shall be used.[Example: Consider a WordprocessingML document with the following information stored in one of its protection elements:<w:… w:cryptProvider="Krista'sProvider" w:hash="9oN7nWkCAyEZib1RomSJTjmPpCY=" />The cryptProvider attribute value of Krista'sProvider specifies that the cryptographic provider with name "Krista's Provider" shall be used if available. end example]The possible values for this attribute are defined by the [ST\_String](ST_String.docx) simple [type](type.docx) (§). |
| cryptProviderType (Cryptographic Provider Type) | Specifies the [type](type.docx) of cryptographic provider to be used.[Example: Consider a WordprocessingML document with the following information stored in one of its protection elements:<w:… w:cryptProviderType="rsaAES" w:hash="9oN7nWkCAyEZib1RomSJTjmPpCY=" />The cryptProviderType attribute value of rsaAES specifies that the cryptographic provider [type](type.docx) shall be an Advanced Encryption Standard provider. end example]The possible values for this attribute are defined by the [ST\_CryptProv](ST_CryptProv.docx) simple [type](type.docx) (§). |
| cryptProviderTypeExt (Cryptographic Provider Type Extensibility) | Specifies that a cryptographic provider [type](type.docx) which was not defined by this Office Open [XML](XML.docx) Standard has been used to generate the hash value stored with this document.This value, when present, shall be interpreted based on the value of the cryptProviderTypeExtSource attribute in order to determine the provider [type](type.docx) used, which shall be application-defined. [Rationale: This extensibility affords the fact that with exponentially increasing computing power, documents created in the future will likely need to utilize as yet undefined cryptographic provider types in order to remain secure. end rationale]If this value is present, the cryptProviderType attribute value shall be ignored in favor of the provider [type](type.docx) defined by this attribute. [Example: Consider a WordprocessingML document with the following information stored in one of its protection elements:<w:… w:cryptProviderTypeExt="00A5691D"  w:cryptProvideTypeExtSource="futureCryptography" w:hash="9oN7nWkCAyEZib1RomSJTjmPpCY=" />The cryptProviderTypeExt attribute value of 00A5691D specifies that the provider [type](type.docx) associated with hex code A5691D shall be used as defined by the futureCryptography application. end example]The possible values for this attribute are defined by the [ST\_LongHexNumber](ST_LongHexNumber.docx) simple [type](type.docx) (§). |
| cryptProviderTypeExtSource (Provider Type Extensibility Source) | Specifies the application which defined the provider [type](type.docx) value specified by the cryptProviderTypeExt attribute.[Example: Consider a WordprocessingML document with the following information stored in one of its protection elements:<w:… w:cryptProviderTypeExt="00A5691D"  w:cryptProvideTypeExtSource="futureCryptography" w:hash="9oN7nWkCAyEZib1RomSJTjmPpCY=" />The cryptProvideTypeExtSource attribute value of futureCryptography specifies that the provider [type](type.docx) used here was published by the futureCryptography application. end example]The possible values for this attribute are defined by the [ST\_String](ST_String.docx) simple [type](type.docx) (§). |
| cryptSpinCount (Iterations to [Run](Run.docx) Hashing Algorithm) | Specifies the number of times the hashing function shall be iteratively run (using each iteration's [result](result.docx) as the input for the [next](next.docx) iteration) when attempting to compare a user-supplied password with the value stored in the hash attribute. [Rationale: Running the algorithm many times increases the cost of exhaustive search attacks correspondingly. Storing this value allows for the number of iterations to be increased over time to accommodate faster hardware (and hence the ability to run more iterations in less time). end rationale][Example: Consider a WordprocessingML document with the following information stored in one of its protection elements:<w:… w:cryptSpinCount="100000" w:hash="9oN7nWkCAyEZib1RomSJTjmPpCY=" />The cryptSpinCount attribute value of 100000 specifies that the hashing function shall be run one hundred thousand times to generate a hash value for comparison with the hash attribute. end example]The possible values for this attribute are defined by the [ST\_DecimalNumber](ST_DecimalNumber.docx) simple [type](type.docx) (§). |
| edit (Document Editing Restrictions) | Specifies the set of editing restrictions which shall be enforced on a given WordprocessingML document, as defined by the simple [type](type.docx) referenced belowIf this attribute is omitted, the consumer shall behave as though there are no editing restrictions applied to this document; equivalent to an attribute value of none.[Example: Consider a WordprocessingML document that contains the following WordprocessingML specifying that hosting applications shall enforce read-only protection for a given document:<w:documentProtection w:edit="readOnly" w:enforcement="1" />The edit attribute has a value of readOnly and a enforcement attribute with a value of 1, specifying that read-only document protection shall be enforced on the given document. end example]The possible values for this attribute are defined by the [ST\_DocProtect](ST_DocProtect.docx) simple [type](type.docx) (§). |
| enforcement (Enforce Document Protection Settings) | Specifies if the document protection settings shall be enforced for a given WordprocessingML document. If the value of this element is off, 0, or false, all the WordprocessingML pertaining to document protection is still preserved in the document, but is not enforced. If the value of this element is on, 1, or true, the document protection is enforced.If this attribute is omitted, then document protection settings shall not be enforced by applications.[Example: Consider a WordprocessingML document that contains the following WordprocessingML specifying that hosting applications shall apply read-only protection for a given document: <w:documentProtection w:edit="readOnly" w:enforcement="1" />The enforcement attribute has a value of 1, specifying that the document protection specified shall be enforced on the given document. end example]The possible values for this attribute are defined by the [ST\_OnOff](ST_OnOff.docx) simple [type](type.docx) (§). |
| formatting (Only Allow Formatting With Unlocked Styles) | Specifies if formatting restrictions are in [effect](effect.docx) for a given WordprocessingML document. This enables the document to restrict the types of [styles](styles.docx) that may exist in a given WordprocessingML document. Specifically, by setting this attribute's value equal to true, every style whose [locked](locked.docx) element (§) has a value of true (or latent [styles](styles.docx) (§) whose [locked](locked.docx) attribute is true) shall not be available for use in the application, nor should any direct formatting. Only [styles](styles.docx) with a [locked](locked.docx) value of false may be used.If this attribute is omitted, then no formatting restrictions shall be applied, even when document protection is enforced.[Example: Consider a WordprocessingML document that shall apply formatting protection. This requirement would be specified using the following WordprocessingML in the document settings:<w:documentProtection w:formatting="true" w:enforcement="true" /> If the following definition for a style was also present in the document:<w:[style](style.docx) w:[type](type.docx)="paragraph" w:styleId="Heading1"> <w:name w:val="heading 1" />  <w:[locked](locked.docx)="1" />  …</w:[style](style.docx)>The formatting attribute has a value of true specifying that the applications shall not allow the style above to be added to the WordprocessingML document. This does not preclude previous uses of that style (which shall not be removed), but does prevent new uses of this style from being added. end example]The possible values for this attribute are defined by the [ST\_OnOff](ST_OnOff.docx) simple [type](type.docx) (§). |
| hash (Password Hash) | Specifies the hash value for the password stored with this document. This value shall be compared with the resulting hash value after hashing the user-supplied password using the algorithm specified by the preceding attributes and parent [XML](XML.docx) element, and if the two values match, the protection shall no longer be enforced.If this value is omitted, then no password shall be associated with the protection, and it may be turned off without supplying any password.[Example: Consider a WordprocessingML document with the following information stored in one of its protection elements:<w:… w:cryptAlgorithmClass="hash"  w:cryptAlgorithmType="typeAny"  w:cryptAlgorithmSid="1" w:hash="9oN7nWkCAyEZib1RomSJTjmPpCY=" />The hash attribute value of 9oN7nWkCAyEZib1RomSJTjmPpCY= specifies that the user-supplied password shall be hashed using the pre-processing defined by the parent element (if any) followed by the SHA-1 algorithm (specified via the cryptAlgorithmSid attribute value of 1) and that the resulting has value must be 9oN7nWkCAyEZib1RomSJTjmPpCY= for the protection to be disabled. end example]The possible values for this attribute are defined by the [XML](XML.docx) Schema base64Binary datatype. |
| salt (Salt for Password Verifier) | Specifies the salt which was prepended to the user-supplied password before it was hashed using the hashing algorithm defined by the preceding attribute values to generate the hash attribute, and which shall also be prepended to the user-supplied password before attempting to generate a hash value for comparison. A salt is a random string which is added to a user-supplied password before it is hashed in order to prevent a malicious party from pre-calculating all possible password/hash combinations and simply using those precalculated values (often referred to as a "dictionary attack").If this attribute is omitted, then no salt shall be prepended to the user-supplied password before it is hashed for comparison with the stored hash value.[Example: Consider a WordprocessingML document with the following information stored in one of its protection elements:<w:… w:salt="ZUdHa+D8F/OAKP3I7ssUnQ==" w:hash="9oN7nWkCAyEZib1RomSJTjmPpCY=" />The salt attribute value of ZUdHa+D8F/OAKP3I7ssUnQ== specifies that the user-supplied password shall have this value prepended before it is run through the specified hashing algorithm to generate a resulting hash value for comparison. end example]The possible values for this attribute are defined by the [XML](XML.docx) Schema base64Binary datatype. |

The following [XML](XML.docx) Schema fragment defines the contents of this element:

<complexType [name](name.docx)="CT\_DocProtect">

 <attribute [name](name.docx)="edit" [type](type.docx)="[ST\_DocProtect](ST_DocProtect.docx)" use="optional"/>

 <attribute [name](name.docx)="formatting" [type](type.docx)="[ST\_OnOff](ST_OnOff.docx)" use="optional"/>

 <attribute [name](name.docx)="enforcement" [type](type.docx)="[ST\_OnOff](ST_OnOff.docx)"/>

 <attributeGroup ref="AG\_Password"/>

</complexType>