[MS-OXPSVAL]: E-Mail Postmark Validation Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

Revision Summary			
Author	Date	Version	Comments
Microsoft Corporation	April 4, 2008	0.1	Initial Availability.
Microsoft Corporation	June 27, 2008	1.0	Initial Release.
Microsoft Corporation	August 6, 2008	1.01	Updated references to reflect date of initial release.
Microsoft Corporation	September 3, 2008	1.02	Revised and edited technical content.
Microsoft Corporation	December 3, 2008	1.03	Revised and edited technical content.
Microsoft Corporation	March 4, 2009	1.04	Revised and edited technical content.
Microsoft Corporation	April 10, 2009	2.0	Updated technical content and applicable product releases.

Table of Contents

1 Introduction

One of the advantages of e-mail is that it is easy and cheap to send. Unfortunately, this also makes it useful to **spammers**, as it enables them to send huge amounts of bulk e-mail.

Postmarking is computational "postage" imposed when sending e-mail messages. This is a small burden for an individual user, but a large burden for spammers. Spammers rely on being able to send thousands of pieces of mail per hour. To send **spam** with postmarking turned on, they would have to invest a large amount of money to expand their computational power.

The E-Mail Postmark Validation protocol specifies the following:

- The process through which a protocol client can create a message that has the **postmark** property.
- The process through which an application can validate the postmark property in the message to help determine whether it is spam.

1.1 Glossary

The following terms are defined in [MS-OXGLOS]:

ASCII

binary large object (BLOB)

GUID

handle

messaging object

Multipurpose Internet Mail Extensions (MIME)

non-Unicode

property

Simple Mail Transfer Protocol (SMTP)

spam

spam confidence level (SCL)

spam filter

Unicode

The following data type is defined in [MS-DTYP]:

byte

The following terms are specific to this document:

Content Filter Agent: A message filter that checks certain conditions in a message to determine a **spam confidence level (SCL)** rating.

postmark: A computational proof that is applied to outgoing messages to help recipient messaging systems distinguish legitimate e-mail messages from junk e-mail messages, reducing the chance of false positives.

presolution header: A string that contains the prepended solutions for the **puzzle**.

Pre-Solver: The component that, given specific inputs, generates a message **postmark**.

puzzle: The computational problem used in this protocol. The **puzzle** is solved by the sending client demonstrating that the message postmark is valid.

x-header: An extended **Simple Mail Transfer Protocol (SMTP)** mail message header.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

[FIP180-1] Federal Information Processing Standards Publication, "Secure Hash Standard", FIPS PUB 180-1, April 1995, http://www.itl.nist.gov/fipspubs/fip180-1.htm.

[MS-OXCNOTIF] Microsoft Corporation, "Core Notifications Protocol Specification", June 2008.

[MS-OXGLOS] Microsoft Corporation, "Exchange Server Protocols Master Glossary", June 2008.

[MS-OXOMSG] Microsoft Corporation, "E-Mail Object Protocol Specification", June 2008.

[MS-OXPROPS] Microsoft Corporation, "Exchange Server Protocols Master Property List Specification", June 2008.

[RFC1123] Braden, R., "Requirements for Internet Hosts – Application and Support", RFC 1123, October 1989, http://www.ietf.org/rfc/rfc1123.txt.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt.

[RFC2821] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001, http://www.ietf.org/rfc/rfc2821.txt.

[RFC2822] Resnick, P., Ed., "Internet Message Format", RFC 2822, April 2001, http://www.ietf.org/rfc/rfc2822.txt.

1.2.2 Informative References

[MSFT-CSRI] Microsoft Corporation, "The Coordinated Spam Reduction Initiative, A Technology and Policy Proposal", February 2004, http://go.microsoft.com/fwlink/?LinkId=112282.

1.3 Protocol Overvew

Postmark validation is a computational proof that a messaging client applies to outgoing messages to help recipient messaging systems distinguish legitimate e-mail messages from junk e-mail messages. This feature helps reduce the chance of the recipient messaging system incorrectly identifying the message as **spam**. In the context of spam filtering, a false positive exists when a **spam filter** incorrectly identifies a message from a legitimate sender as spam. When E-mail Postmark validation is enabled, the **Content Filter Agent** parses the inbound message for a computational postmark header. The presence of a valid, solved computational postmark header in the message indicates that the client computer that is sending the message has solved the computational postmark and has included the **puzzle** solution in the message headers.

Computers do not require significant processing time to solve individual computational postmarks. However, the processing time required to compute individual postmarks for large numbers of messages is expected to be prohibitive, and therefore will discourage malicious e-mail senders. Individual systems that send millions of spam messages are unlikely to invest the processing power required to solve each computational postmark for each message. For that reason, when a sender's e-mail message contains a valid, solved computational postmark, it is unlikely that the sender is a malicious sender.

1.4 Relationship to Other Protocols

When the e-mail client and recipient server are communicating via the E-mail Object protocol, as specified in [MS-OXOMSG], the E-Mail Postmark Validation protocol defines two properties that the client attaches to an e-mail message. Therefore, the E-Mail Postmark Validation protocol relies on the underlying message structures and the handling specified in [MS-OXOMSG].

The Core Notifications protocol, as specified in [MS-OXCNOTIF], provides more information about the properties that are used to send and receive messages.

The Exchange Server Protocols Master Property List Specification, as specified in [MS-OXPROPS], provides more information about the data types that are used in this protocol.

1.5 Prerequisites/Preconditions

The E-Mail Postmark Validation protocol assumes that the client has successfully logged on to the server.

1.6 Applicability Statement

This protocol specification defines how e-mail messaging clients can generate and understand computational **postmarks**. By using this protocol, the client can reduce the number of false positives detected by the recipient server when it tries to identify **spam** e-mail messages.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The transport protocols used by this specification are defined in [MS-OXOMSG].

2.2 Message Syntax

The following sections specify the properties that are specific to the E-Mail Postmark Validation protocol. Before sending these requests to the server, the messaging client MUST be logged on to the server. The protocol client MUST open/acquire **handles** to all **messaging objects** and properties that are set or retrieved.

2.2.1 Input Parameters for Generating the Puzzle

The input parameters specified in the following sections are used to calculate the **puzzle**.

Note: All "*String*" values, unless otherwise specified, MUST be in **Unicode** format UTF-16 or UCS-2<1>.

2.2.1.1 Number of Recipients

This parameter specifies the total count of **SMTP** message recipients on the "To:" and "Cc: " lines.

This parameter MUST be a decimal value formatted as type "*String*".

Note: Non-SMTP message recipients MUST NOT be counted.

2.2.1.2 Message "To: " and "Cc: " Recipients

This parameter is a string that contains a semicolon separated list of **SMTP** [RFC2821] addresses that are found on the "To: " and "Cc: " lines.

This parameter MUST be formatted as type "*String*" and MUST be base-64 encoded.

Note: Addresses on the "Bcc:" lines MUST NOT be used.

Note: Accounts that are compatible with [MS-OXOMSG] MUST reference the following **properties:**

- **PidTagEmailAddress**
- **PidTagAddressType**

The recipient string is calculated by means of the following pseudo-logic:

```
For each of the recipients in the [Recipient List] {
    Get the PidTagAddressType and PidTagEmailAddress properties.
    if (PidTagAddressType  == “SMTP”) {
        Append PidTagEmailAddress value, followed by a semi-colon,
        to recipient string.
    }
}
Remove the last semi-colon at the end of the recipient string.
```

2.2.1.3 Algorithm type

This parameter contains the algorithm type that is used to generate the **puzzle**.

This parameter MUST be a formatted as type "*String*".

Note: The puzzle-solving system SHOULD use "sosha1_v1", as it is currently the only valid algorithm type.

2.2.1.4 Degree of Difficulty

This parameter contains the degree of difficulty for which a **puzzle** solution is sought. A larger Degree of Difficulty value indicates that the **puzzle**-generating application used more computing resources to create the puzzle. Therefore, the receiving system typically assumes that a larger Degree of Difficulty value corresponds to a lower likelihood that the message is **spam**. This is only a generally accepted guideline, and is not a protocol requirement.

This parameter MUST be a positive integer value that is formatted as type "*String*".<2>

2.2.1.5 Message Identifier

This parameter contains a unique ID that is represented by a **GUID**.

This parameter MUST be formatted as type "*String*" and MUST be enclosed in brackets "{}".

2.2.1.6 Message "From: "Address

This parameter contains the sender's **SMTP** e-mail "From: " address.

This parameter MUST be formatted as type "*String*" and MUST be base-64 encoded.

Note: Accounts that are compatible with [MS-OXOMSG] MUST use the **PidTagSenderEmailAddress property**.

2.2.1.7 Datetime

This parameter contains the creation time of the **puzzle**.

This parameter MUST consist of **ASCII** characters and MUST be formatted as specified in [RFC1123].

2.2.1.8 Subject Line

This parameter contains the subject of the message, as specified in [RFC2822].

This parameter MUST be formatted as type "*String*" and MUST be base-64 encoded.

Note: Accounts that are compatible with [MS-OXOMSG] MUST reference the **PidTagSubject property**.

2.2.2 Pre-Solver Output values

The **Pre-Solver** will return two values, which are then stored in the message header as **x-header properties**.

2.2.2.1 "X-CR-PuzzleID" X-Header Property

The value of the "**X-CR-PuzzleID**" **x-header property** MUST be the same value as the message identifier specified in section 2.2.1.5.

The "**X-CR-PuzzleID**" **x-header property** MUST be formatted as type "*String*".

2.2.2.2 "X-CR-HashedPuzzle" X-Header Property

The value of the "**X-CR-HashedPuzzle**" **x-header property** contains the **puzzle** solution as defined by section 3.1.4.1.1.

The "**X-CR-PuzzleID**" **x-header property** MUST be formatted as type "*String*".

3 Protocol Details

3.1 Protocol Client Details

3.1.1 Abstract Data Model

None.

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Submit Message Event

3.1.4.1.1 Generating X-CR-HashedPuzzle

The **puzzle** *P* takes the following parameters as input (see section 2.2.12.2.1):

- Number of recipients *r*.
- E-mail addresses of the recipients *t*.
- Algorithm type *a*.
- A 'degree of difficulty' *n*.
- A message identifier *m*.
- An e-mail 'From: address' *f*.
- A datetime *d*.
- A subject line *s*.

From these parameters, a document *D* is formed by concatenating all the parameters together, separating each field with ';'. The constructed document *D* is represented in an **non-Unicode** string.

Given the sequence of **bytes** comprising a document *D,* the computational task involved in the puzzle is to find and exhibit a set of sixteen documents *δ* such that both of the following are true:

- When each *δ* is prepended to the hash under the Son-of-SHA-1 hash algorithm *H* (see section 3.1.4.2) of *D* with its whitespace removed and then hashed again to form *H(δ o H(NWS(D))),* the result is zero in at least the first *n* bits (taken most significant bit first within each **byte** taken in order). Here, *NWS* is the function that takes a sequence of bytes as input, removes all those that are legal characters that could match the FWS production specified in [RFC2822], and produces the remaining as output.

- The last 12 bits of each *H(δ o H(NWS(D))* are the same (the particular 12-bit suffix value shared by these documents does not matter).

That is, the answer to the puzzle *P(t, n, m, f, d, s)* is a set of 16 documents *δ* each with these characteristics. The hash *H(NWS(D))* is used as the suffix to which each *δ* is prepended rather than simply *D* in order to minimize the effect of variation in the length of *D* on the length of time required to solve the puzzle. Whitespace is stripped from *D* before being input to the hash in order to minimize sensitivity to the encoding of *D* in header fields where it can be subjected to folding.

No means other than brute force is known to locate satisfactory *δ*; however, that a given set of *δ* indeed answers the puzzle can be quickly verified. The particular brute force approach of first trying all one-byte solutions, then trying all two-byte solutions, then all three-byte solutions, and so on, is as good a solution algorithm as any other, but has the additional benefit that the solutions found will be as small as possible. Furthermore, for puzzles that have reasonable degrees of difficulty, solutions with four or fewer bytes will be typical.

Specifically, the following pseudo code describes the brute force algorithm:

```
Solution = 0;

While(true){

    Hash = H(concatenate(Solution, H(NWS(D))))

    If Verify(Solution, Puzzle) succeeds {

        Remember this solution and Hash

        If we have 16 solutions whose last 12 bits of their
        corresponding Hash are the same {

            Return these 16 solutions

        }

    }

Solution ++

}
```

After the solutions for puzzle P are found, a **presolution header** is generated. The presolution header MUST be the concatenation of the solutions string and the document D separated by a semicolon. The solutions string MUST be a "*String*" formed by base64 encoding each of the 16 puzzle solutions and concatenating them together, with a "" (space) delimiter.

The value of **X-CR-HashedPuzzle** MUST be set to the presolution header. See section 4 for examples.

3.1.4.2 Son-Of-SHA-1 Hash Algorithm

The Son-of-SHA-1 algorithm is defined as a constrained perturbation of the [FIP180-1] algorithm. The intent of defining a new hash algorithm that is unique to the proposed use of computational **puzzles** for **spam** reduction is to reduce the ease with which hardware accelerators can be applied to reduce the cost and duration of puzzle solving. In conformant systems, the Son-of-SHA-1 algorithm MUST NOT be implemented in hardware.

In "§5 Functions Used" of the specification of Son-Of-SHA-1, a set of eighty functions are defined that are subsequently used in the core of the algorithm specified in §7 and §8. Each f_t, 0 <= t <= 79, operates on three 32-bit words B, C, D and produces a 32-bit word as output.

The Son-Of-SHA-1 algorithm differs from [FIP180-1] only in the specification of these functions. Specifically, where [FIP180-1] specifies the eighty functions as follows:

f_t(B,C,D) = (B AND C) OR ((NOT B) AND D) (0 <= t <= 19)

f_t(B,C,D) = B XOR C XOR D (20 <= t <= 39)

f_t(B,C,D) = (B AND C) OR (B AND D) OR (C AND D) (40 <= t <= 59)

f_t(B,C,D) = B XOR C XOR D (60 <= t <= 79)

The Son-of-SHA-1 algorithm instead specifies the first of these functions as involving an additional **XOR** operation:

f_t(B,C,D) = g(B,C,D) XOR ((B AND C) OR ((NOT B) AND D)) (0 <= t <= 19)

f_t(B,C,D) = (B XOR C XOR D) (20 <= t <= 39)

f_t(B,C,D) = (B AND C) OR (B AND D) OR (C AND D) (40 <= t <= 59)

f_t(B,C,D) = (B XOR C XOR D) (60 <= t <= 79)

The supporting function g (B,C,D) is defined as follows:

g_t(B,C,D) = n(r(m(B,C), m(C,D)))

The binary function m() takes two 32-bit words as input and produces a non-negative 64-bit integer as output by concatenating the two 32-bits words together with the first word, forming the high-order bits of the following result:

m(B,C) = (B << 32) OR C

The unary function $n()$ takes a single 64-bit integer as input and returns the word consisting of the following lower 32 bits:.

$n(x) = x$ AND FFFFFFFF

Finally, the binary function $r()$ takes two 64-bit integers as input and computes the 64-bit integer that is the remainder of the first when divided by the second (unless the latter is zero). Specifically, $r(x,y)$ is defined by the following relations:

If $y \neq 0$: $x = k\, y + r(x,y)$ for some non-negative integer k, where $0 <= r(x,y) < y$

If $y = 0$: $x = r(x,y)$

Other than the introduction of function g(), another difference between Son-Of-SHA-1 and [FIP180-1] is that in [FIP180-1], the following are the constants that are used:

K = 5A827999 (0 <= t <= 19)
K_t = 6ED9EBA1 (20 <= t <= 39)
K_t = 8F1BBCDC (40 <= t <= 59)
K_t = CA62C1D6 (60 <= t <= 79).

In Son-Of-SHA-1, the constants are instead the following:

K = 041D0411 (0 <= t <= 19)
K_t = 416C6578 (20 <= t <= 39)
K_t = A116F5B6 (40 <= t <= 59)
K_t = 404B2429 (60 <= t <= 79).

In all other ways, the Son-of-SHA-1 algorithm is identical to [FIP180-1].

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 On Message Delivery

3.1.5.1.1 Determining When to Validate

The presence of the custom **SMTP** header **X-CR-HashedPuzzle** indicates that the message is a presolved message.

The receiving client SHOULD verify that the parameters, as expressed in the **puzzle**, match the fields of the e-mail message as specified in section 2, in order to prevent **spammers** from reusing the same presolved message **binary large object (BLOB)** for multiple recipients, thereby allowing them to get away with doing less computation.

The actual difficulty of computing a presolution can be expressed as the difficulty indicated by n, multiplied by the number of To: and Cc: recipients in the presolved message indicated by r (in other words, the number of To: tags in the presolution data).

3.1.5.1.2 Validating the Puzzle

The process of validating the **puzzle** is performed on the receiving end of the communication. The server-side Mail Transport Authority (MTA) SHOULD validate the puzzle. Also, e-mail clients SHOULD validate the puzzle.

The validating process is divided into the following two steps:

1. Validate the puzzle part inside the presolution, making sure that the puzzle is generated for the received e-mail message. An e-mail message passes this validation if all the following tests pass:

    a. Extract Recipient Part (*RP*) information from the puzzle string (*r & t).*

        i. *RP* SHOULD be a subset of the **MIME** Recipients extracted from the MIME header of the e-mail message.

        ii. *RP* SHOULD contain the recipient's **SMTP** address.

            1. If the algorithm is being run on an e-mail client, the client will have a list of e-mail accounts, Recipient Catalog (*RC*). At least one e-mail address of *RC* MUST be in *RP.*

            2. If the algorithm is being run on an e-mail server, the protocol server will have a list of e-mail addresses, and Received Recipients (*RR*) from the RCPT TO command as part of the SMTP [RFC2821] process. *RR* MUST be a subset of *RP.*

    b. Extract the message identifier from the puzzle string *m.* The identifier MUST match the puzzle ID extracted from the x-cr-puzzleid header.

    c. Extract the Sender Part from the puzzle string *f.* The sender's e-mail address MUST match the FROM address in the MIME header of the e-mail message.

    d. Extract the subject line from the puzzle string *s.* The subject line MUST match the subject extracted from the MIME header of the e-mail message.

2. Validate the solution part inside the presolution. The solution for the puzzle MUST meet the difficulty level *n.*

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Server Details

The server SHOULD validate **postmarks** after the e-mail message arrives at the server. The content specified in 3.1.5.1 is symetrical on both the client and the server when an e-mail message is received.

3.2.1 Abstract Data Model

None.

3.2.2 Timers

None.

3.2.3 Initialization

None.

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Message Processing Events and Sequencing Rules

None.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

4 Protocol Examples

4.1 Example 1

Input	Parameter	Value	Base64 encoded
	Number of **recipients**	1	
	Recipient list	"user1@example.com"	dQBzAGUAcgAxAEAAZQB4AGEAbQBwAGwAZQAuAGMAbwBtAA==
	Algorithm type	"sosha1_v1"	
	Degree of difficulty	7	
	Message identifier	"{d04b23f4-b443-453a-abc6-3d08b5a9a334}"	
	From address	"sender@example.com"	cwBlAG4AZABlAHIAQABlAHgAYQBtAHAAbABlAC4A

			YwBvAG0A
	DateTime	"Tue, 01 Jan 2008 08:00:00 GMT"	
	Subject	"Hello"	SABlAGwAbABvAA==
Result	"X-CR-HashedPuzzle: BjHi CbbP CsE4 DoWO EhAv FJE7 FMx3 FOJO FjsQ HDPJ IFAE IRyJ I5E3 I+BV KBb7 L+gd;1;dQBzAGUAcgAxAEAAZQB4AGEAbQBwAGwAZQAuAGMAbwBtAA==;Sosha1_v1;7;{d04b23f4-b443-453a-abc6-3d08b5a9a334};cwBlAG4AZABlAHIAQABlAHgAYQBtAHAAbABlAC4AYwBvAG0A;Tue, 01 Jan 2008 08:00:00 GMT;SABlAGwAbABvAA==X-CR-PuzzleID: {d04b23f4-b443-453a-abc6-3d08b5a9a334}"		

4.2 Example 2

Input	Parameter	Value	Base64 encoded
	Number of **recipients**	2	
	Recipient list	"user1@example.com;user2@example.com"	dQBzAGUAcgAxAEAAZQB4AGEAbQBwAGwAZQAuAGMAbwBtADsAdQBzAGUAcgAyAEAAZQB4AGEAbQBwAGwAZQAuAGMAbwBtAA==
	Algorithm type	"sosha1_v1"	
	Degree of difficulty	7	
	Message identifier	"{d04b23f4-b443-453a-abc6-3d08b5a9a334}"	
	From address	"sender@example.com"	cwBlAG4AZABlAHIAQABlAHgAYQBtAHAAbABlAC4AYwBvAG0A
	DateTime	"Tue, 01 Jan 2008 08:00:00 GMT"	
	Subject	"Hello"	SABlAGwAbABvAA==
Result	"X-CR-HashedPuzzle: AejA Arsz Bwjf DuSf Een1 Et0s FrxA GmCG HaiQ It8u Jpqj QdZB R6vS SDZh SrAv UANK;2;dQBzAGUAcgAxAEAAZQB4AGEAbQBwAGwAZQAuAGMAbwBtADsAdQBzAGUAcgAyAEAAZQB4AGEAbQBwAGwAZQAuAGMAbwBtAA==;Sosha1_v1;7;{d04b23f4-b443-453a-abc6-3d08b5a9a334};cwBlAG4AZABlAHIAQABlAHgAYQBtAHAAbABlAC4AYwBvAG0A;Tue, 01 Jan 2008 08:00:00 GMT;SABlAGwAbABvAA==X-CR-PuzzleID: {d04b23f4-b443-453a-abc6-3d08b5a9a334}"		

4.3 Example 3

The following table provides four examples of hash values that result from using the [FIPS180-1] Son-of-Sha-1 algorithm on the indicated input values.

Input	Son-of-Sha-1 hash value
The string "abc"	FA12E295 9DB79C97 25338C0F D4DE3E01 78C286BD
The string "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq"	48F6CE9F DCF53F40 89200091 ED9739E1 7D73D975
A string consisting of 1,000,000 a's	57338A4C C33E70D4 3A3D3AD7 E93C85ED E6996CCD
An empty string	7A790886 F5044A7B DA812BA8 BFC286C4 F51E7B34

5 Security

5.1 Security Considerations for Implementers

There are no special security considerations specific to the E-Mail Postmark Validation protocol. General security considerations that pertain to the underlying E-Mail Object protocol, as specified in [MS-OXOMSG], apply.

5.2 Index of Security Parameters

None.

6 Appendix A: Office/Exchange Behavior

The information in this specification is applicable to the following versions of Office/Exchange:

- Microsoft Exchange Server 2003
- Microsoft Office Outlook 2007
- Microsoft Exchange Server 2007
- Microsoft Outlook 2010
- Microsoft Exchange Server 2010

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Office/Exchange behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies Office/Exchange does not follow the prescription.

The following table lists the product, along with the presolution generation and verification.

Product	Presolution generation	Presolution verification
Outlook 2010	Yes	Yes
Outlook 2007	Yes	Yes
Exchange 2003	No	Yes (both patches "KB 922105" and "KB 912064" have to be installed)
Exchange 2007	No	Yes

<1> Section 2.2.1: Outlook 2007 and Outlook 2010 always format parameters as UTF-16.

<2> Section 2.2.1.4: Outlook 2007 and Outlook 2010 always use "7" as the Degree of Difficulty value.

Index