[MS-OXPSVAL]:

Email Postmark Validation Algorithm

Revision Summary

Date	Revision History	Revision Class	Comments
4/4/2008	0.1	New	Initial Availability.
6/27/2008	1.0	Major	Initial Release.
8/6/2008	1.01	Minor	Updated references to reflect date of initial release.
9/3/2008	1.02	Minor	Revised and edited technical content.
12/3/2008	1.03	Minor	Revised and edited technical content.
3/4/2009	1.04	Minor	Revised and edited technical content.
4/10/2009	2.0	Major	Updated technical content and applicable product releases.
7/15/2009	3.0	Major	Revised and edited technical content.
11/4/2009	3.1.0	Minor	Updated the technical content.
2/10/2010	4.0.0	Major	Updated and revised the technical content.
5/5/2010	5.0.0	Major	Updated and revised the technical content.
8/4/2010	5.1	Minor	Clarified the meaning of the technical content.
11/3/2010	5.2	Minor	Clarified the meaning of the technical content.
3/18/2011	6.0	Major	Significantly changed the technical content.
8/5/2011	7.0	Major	Significantly changed the technical content.
10/7/2011	7.0	None	No changes to the meaning, language, or formatting of the technical content.
1/20/2012	8.0	Major	Significantly changed the technical content.
4/27/2012	9.0	Major	Significantly changed the technical content.
7/16/2012	9.0	None	No changes to the meaning, language, or formatting of the technical content.
10/8/2012	10.0	Major	Significantly changed the technical content.
2/11/2013	11.0	Major	Significantly changed the technical content.
7/26/2013	12.0	Major	Significantly changed the technical content.
11/18/2013	12.0	None	No changes to the meaning, language, or formatting of the technical content.
2/10/2014	12.0	None	No changes to the meaning, language, or formatting of the technical content.
4/30/2014	12.0	None	No changes to the meaning, language, or formatting of the technical content.
7/31/2014	12.0	None	No changes to the meaning, language, or formatting of the technical content.
10/30/2014	12.0	None	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
5/26/2015	13.0	Major	Significantly changed the technical content.
9/14/2015	13.0	None	No changes to the meaning, language, or formatting of the technical content.
6/13/2016	13.0	None	No changes to the meaning, language, or formatting of the technical content.
9/14/2016	13.0	None	No changes to the meaning, language, or formatting of the technical content.
7/24/2018	14.0	Major	Significantly changed the technical content.
10/1/2018	15.0	Major	Significantly changed the technical content.

Table of Contents

1 Introduction

The Email Postmark Validation Algorithm enables a client to create an e-mail message with a **postmark** header. This algorithm also enables a client to validate the postmark property on an incoming e-mail message to determine whether it is **spam**.

Sections 1.6 and 2 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

ASCII: The American Standard Code for Information Interchange (ASCII) is an 8-bit character-encoding scheme based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that work with text. ASCII refers to a single 8-bit ASCII character or an array of 8-bit ASCII characters with the high bit of each character set to zero.

base64 encoding: A binary-to-text encoding scheme whereby an arbitrary sequence of bytes is converted to a sequence of printable **ASCII** characters, as described in [RFC4648].

binary large object (BLOB): A discrete packet of data that is stored in a database and is treated as a sequence of uninterpreted bytes.

globally unique identifier (GUID): A term used interchangeably with universally unique identifier (UUID) in Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the value. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the **GUID**. See also universally unique identifier (UUID).

message transfer agent (MTA): An **SMTP** server that accepts mail from a client or another MTA and delivers the mail or relays it to another MTA.

messaging object: An object that exists in a mailbox. It can be only a Folder object or a Message object.

Multipurpose Internet Mail Extensions (MIME): A set of extensions that redefines and expands support for various types of content in email messages, as described in [RFC2045], [RFC2046], and [RFC2047].

non-Unicode: A character set that has a restricted set of glyphs, such as Shift_JIS or ISO-2022-JP.

postmark: A computational proof that is applied to outgoing messages to help recipient messaging systems distinguish legitimate email messages from junk email messages, which reduces the chance of false positives.

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

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

recipient: An entity that can receive email messages.

resource: Any component that a computer can access where data can be read, written, or processed. This resource could be an internal component such as a disk drive, or another computer on a network that is used to access a file.

Simple Mail Transfer Protocol (SMTP): A member of the TCP/IP suite of protocols that is used to transport Internet messages, as described in [RFC5321].

spam: An unsolicited email message.

Unicode: A character encoding standard developed by the Unicode Consortium that represents almost all of the written languages of the world. The **Unicode** standard [UNICODE5.0.0/2007] provides three forms (UTF-8, UTF-16, and UTF-32) and seven schemes (UTF-8, UTF-16, UTF-16 BE, UTF-16 LE, UTF-32, UTF-32 LE, and UTF-32 BE).

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

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[FIPS180] FIPS PUBS, "Secure Hash Standard", FIPS PUB 180-1, April 1995, http://niatec.info/GetFile.aspx?pid=63

[MS-OXCMSG] Microsoft Corporation, "Message and Attachment Object Protocol".

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

[MS-OXOABK] Microsoft Corporation, "Address Book Object Protocol".

[MS-OXOMSG] Microsoft Corporation, "Email Object Protocol".

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

[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.rfc-editor.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

None.

1.3 Overview

Postmark validation is a computational proof that a client applies to outgoing e-mail messages. Postmarks help **recipients** distinguish legitimate e-mail from **spam**. When a recipient has postmark validation enabled, its spam filter parses each incoming e-mail message for a postmark header.

An e-mail message with a postmark header is less likely to be spam than one without a postmark header. This is because a computer does not require significant processing time to solve an individual computational postmark, but the processing time required to do so for large numbers of messages is expected to be prohibitive. This computational cost is expected to discourage spam senders. For examples of postmarked e-mails, see sections 3.1 and 3.2.

1.4 Relationship to Protocols and Other Algorithms

When the e-mail client and **recipient** server are communicating via the Email Object Protocol, as specified in [MS-OXOMSG], the Email Postmark Validation Algorithm uses two properties that the client attaches to an e-mail message. Therefore, the Email Postmark Validation Algorithm 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, as specified in [MS-OXPROPS], provides more information about the data types that are used by this algorithm.

For conceptual background information and overviews of the relationships and interactions between this and other protocols, see [MS-OXPROTO].

1.5 Applicability Statement

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

1.6 Standards Assignments

None.

2 Algorithm Details

2.1 Common Algorithm Details

The following sections specify the properties that are used by the Email Postmark Validation Algorithm. Before sending these requests to the server, the messaging client MUST be logged on to the server.<1> The client MUST open/acquire handles to all **messaging objects** and properties that are set or retrieved by this algorithm.

2.1.1 Abstract Data Model

2.1.1.1 Input Parameters for Generating the Puzzle

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

All **string** values, unless otherwise specified, MUST be in **Unicode** format UTF-16 or UCS-2. It is up to the client implementation to choose which format to use; the algorithm treats both formats identically.<2>

2.1.1.1.1 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**.

Message recipients other than SMTP message recipients MUST NOT be counted.

2.1.1.1.2 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 encoded with **base64 encoding**. Addresses on the "Bcc:" lines MUST NOT be used. Accounts that are compatible with [MS-OXOMSG] MUST reference the following properties:

- **PidTagEmailAddress** ([MS-OXOABK] section 2.2.3.14)
- **PidTagAddressType** ([MS-OXOABK] section 2.2.3.13)

The **recipient** string is calculated by means of the following pseudologic:

```
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.
```

2.1.1.1.3 Algorithm Type

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

This parameter MUST be a formatted as type **string**.

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

2.1.1.1.4 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**.<3>

2.1.1.1.5 Message Identifier

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

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

2.1.1.1.6 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 encoded with **base64 encoding**.

Accounts that are compatible with [MS-OXOMSG] MUST use the **PidTagSenderEmailAddress** property ([MS-OXOMSG] section 2.2.1.49).

2.1.1.1.7 DateTime

This parameter contains the creation time of the puzzle.

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

2.1.1.1.8 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 encoded with **base64 encoding**.

Accounts that are compatible with [MS-OXOMSG] MUST reference the **PidTagSubject** property ([MS-OXCMSG] section 2.2.1.46).

2.1.1.2 Pre-Solver Output Values

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

2.1.1.2.1 "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 2.1.1.1.5.

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

2.1.1.2.2 "X-CR-HashedPuzzle" X-Header Property

The value of the "**X-CR-HashedPuzzle**" x-header property contains the puzzle solution as defined in section 2.2.3.1.

The "**X-CR-HashedPuzzle**" x-header property MUST be formatted as type **string**.

2.1.2 Initialization

None.

2.1.3 Processing Rules

None.

2.2 Submit Message Algorithm Details

2.2.1 Abstract Data Model

None.

2.2.2 Initialization

None.

2.2.3 Processing Rules

2.2.3.1 Generating X-CR-HashedPuzzle

The puzzle P takes the following parameters as input (see section 2.1.1.1):

- Number of **recipients** r.
- E-mail addresses of the recipients t.
- Algorithm type a.
- A 'degree of difficulty' n.
- A message id 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 a **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 2.3.3) 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).

Note In the previous two computations, the o operator denotes string concatenation.

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 pseudocode 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 3 for examples.

2.3 Son-Of-SHA-1 Hash Algorithm Details

2.3.1 Abstract Data Model

None.

2.3.2 Initialization

None.

2.3.3 Processing Rules

The Son-of-SHA-1 algorithm is defined as a constrained perturbation of the [FIPS180] 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", as specified in [FIPS180], 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 [FIPS180] only in the specification of these functions. Specifically, where [FIPS180] 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 [FIPS180] is that in [FIPS180], 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 [FIPS180].

2.4 Message Delivery Algorithm Details

2.4.1 Abstract Data Model

None.

2.4.2 Initialization

None.

2.4.3 Processing Rules

2.4.3.1 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 Recipient tags in the presolution data).

2.4.3.2 Validating the Puzzle

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

The validating process is divided into the following 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:
    1. Extract recipient part information from the puzzle string (r & t).
        - The recipient part SHOULD be a subset of the **MIME recipients** extracted from the MIME header of the e-mail message.
        - The recipient part SHOULD contain the recipient's **SMTP** address.
            - If the algorithm is being run on an e-mail client, the client will have a list of e-mail accounts, recipient catalog. At least one e-mail address from the recipient catalog MUST be in recipient part.

- 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 from the RCPT TO command as part of the SMTP [RFC2821] process. The received recipients MUST be a subset of recipient part.

2. Extract the message identifier from the puzzle string m. The identifier MUST match the puzzle ID extracted from the X-CR-PuzzleID header.

3. 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.

4. 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 Algorithm Examples

3.1 Postmark for a Message with One Recipient Using Son-of-SHA-1 Algorithm

The following table describes a message with one **recipient** that is postmarked using the Son-of-SHA-1 algorithm on the indicated input values. For information about the Son-of-SHA-1 algorithm, see section 2.3.

Input	Parameter	Value	Encoded With Base64 Encoding
Input	Number of recipients	1	
	recipient list	"user1@contoso.com"	dQBzAGUAcgAxAEAAZQB4AG EAbQBwAGwAZQAuAGMAbwBtAA==
	Algorithm type	"sosha1_v1"	
	Degree of difficulty	7	
	message IDentifier	"{d04b23f4-b443-453a-abc6-3d08b5a9a334}"	
	From address	"sender@contoso.com"	cwBlAG4AZABlAHIAQABlAH gAYQBtAHAAbABlAC4AYwBvAG0A
	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}"		

3.2 Postmark for a Message with Two Recipients Using Son-of-SHA-1 Algorithm

The following table describes a message with two **recipients** that is postmarked using the Son-of-SHA-1 algorithm on the indicated input values. For information about the Son-of-SHA-1 algorithm, see section 2.3.

Input	Parameter	Value	Encoded With Base64 Encoding
Input	Number of recipients	2	
	recipient list	"user1@contoso.com; user2@contoso.com"	dQBzAGUAcgAxAEAAZQB4AGEAbQBwAGwAZQAuAGMAbwBtAD sAdQBzAGUAcgAyAEAAZQB4AGEAbQBwAGwAZQAuAGMAbwBt AA==
	Algorithm type	"sosha1_v1"	

Input	Parameter	Value	Encoded With Base64 Encoding
	Degree of difficulty	7	
	message IDentifier	"{d04b23f4-b443-453a-abc6-3d08b5a9a334}"	
	From address	"sender@contoso.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;dQBzAGUAcgAxAEAAZQB4AGEAbQBwAGwAZQAuAGMAbwBtADsAdQ BzAGUAcgAyAEAAZQB4AGEAbQBwAGwAZQAuAGMAbwBtAA==;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}"	

3.3 Hash Values from Son-of-SHA-1 Algorithm

The following table provides four examples of hash values that result from using the Son-of-SHA-1 algorithm on the indicated input values. For information about the Son-of-SHA-1 algorithm, see section 2.3.

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" characters	57338A4C C33E70D4 3A3D3AD7 E93C85ED E6996CCD
An empty string	7A790886 F5044A7B DA812BA8 BFC286C4 F51E7B34

4 Security

4.1 Security Considerations for Implementers

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

4.2 Index of Security Parameters

None.

5 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include updates to those products.

- Microsoft Exchange Server 2003
- Microsoft Exchange Server 2007
- Microsoft Exchange Server 2010
- Microsoft Exchange Server 2013
- Microsoft Exchange Server 2016
- Microsoft Office Outlook 2007
- Microsoft Outlook 2010
- Microsoft Exchange Server 2019

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms "SHOULD" or "SHOULD NOT" implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term "MAY" implies that the product does not follow the prescription.

<1> Section 2.1: Outlook 2010 never stamps a **postmark** for outgoing mail.

<2> Section 2.1.1.1: Office Outlook 2007 always formats parameters as UTF-16.

<3> Section 2.1.1.1.4: Office Outlook 2007 always uses "7" as the Degree of Difficulty value.

6 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as Major, Minor, or None.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements.
- A document revision that captures changes to protocol functionality.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **None** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the relevant technical content is identical to the last released version.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Description	Revision class
5 Appendix A: Product Behavior	Updated list of supported products.	Major

7 Index