Network Working Group	A. Bryan	
Internet-Draft	T. Kosse	
Intended status: Standards Track	D. Stenberg	
Expires: April 29, 2011	October 26, 2010	

TOC

File Transfer Protocol HASH Command for Cryptographic Hashes
draft-bryan-ftp-hash-08

Abstract

The File Transfer Protocol does not offer any method to verify the integrity of a transferred file, nor can two files be compared against each other without actually transferring them first. Cryptographic hashes are a possible solution to this problem. In the past, several attempts have been made to add commands to obtain checksums and hashes, however none have been formally specified, leading to non-interoperability and confusion. To solve these issues, this document specifies a new FTP command to be used by clients to request cryptographic hashes of files.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.
Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.
Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as “work in progress.”
This Internet-Draft will expire on April 29, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted

from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction
    1.1. Example
2. Document Conventions
    2.1. Basic Tokens
    2.2. Server Replies
3. The HASH Command (HASH)
    3.1. FEAT Command Response for HASH Command
    3.2. OPTS Parameters for HASH
    3.3. User-PI usage of HASH
    3.4. HASH Command Errors
4. IANA Considerations
5. Implementation Requirements
6. Security Considerations
7. References
    7.1. Normative References
    7.2. Informative References

Appendix A. Acknowledgements and Contributors

Appendix B. List of Non-standard Cryptographic Hash or Checksum Commands and Implementations

Appendix C. Document History

§ Authors' Addresses

---

1. Introduction

TOC

The File Transfer Protocol [RFC0959] (Postel, J. and J. Reynolds, “File Transfer Protocol,” October 1985.) does not offer any method to verify the integrity of a transferred file, nor can two files be compared against each other without actually transferring them first. Cryptographic hashes are a possible solution to this problem. In the past, several attempts have been made to add commands to obtain checksums and hashes, however none have been formally specified, leading to non-interoperability and confusion. (See Appendix B for more information). To solve these issues, this document specifies a new FTP command to be used by clients to request cryptographic hashes of files. HTTP has a similar feature named Instance Digests [RFC3230] (Mogul, J. and A. Van Hoff, “Instance Digests in HTTP,” January 2002.) which allows a client to request the cryptographic hash of a file.

[[Discussion of this draft should take place on ftpext@ietf.org (or apps-discuss@ietf.org if necessary).]]

1.1. Example

TOC

Example of HASH client request:

```
C> HASH filename.ext
```

HASH server response with Positive Completion code, the currently selected HASH algorithm, the requested hash of the file, and the pathname of the file:

```
S> 213 SHA-1 80bc95fd391772fa61c91ed68567f0980bb45fd9 filename.ext
```

---

2. Document Conventions

TOC

This specification describes conformance of File Transfer Protocol Extension for cryptographic hashes.
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119] (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.), as scoped to those conformance targets.
This document also uses notation defined in STD 9, [RFC0959] (Postel, J. and J. Reynolds, “File Transfer Protocol,” October 1985.). In particular, the terms or commands "reply", "user", "file", "FTP commands", "user-PI" (user protocol interpreter), "server-FTP process", "server-PI", "mode", "type", "STOR", "RETR", and "ASCII", are all used here as defined there. The term "pathname" is used as defined in Section 2.2 of [RFC3659] (Hethmon, P., “Extensions to FTP,” March 2007.).
In the examples of FTP dialogs presented in this document, lines that begin "C> " were sent over the control connection from the user-PI to the server-PI, and lines that begin "S> " were sent over the control connection from the server-PI to the user-PI. In all cases, the prefixes shown above, including the one space, have been added for the purposes of this document, and are not a part of the data exchanged between client and server.
Syntax required is defined using the Augmented BNF defined in [RFC5234] (Crocker, D. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” January 2008.).

2.1. Basic Tokens

TOC

This document imports the core definitions given in Appendix B of [RFC5234] (Crocker, D. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” January 2008.). There definitions will be found for basic ABNF elements like ALPHA, DIGIT, SP, etc. To that, the following term is added for use in this document.

```
TCHAR = VCHAR / SP / HTAB    ; visible plus white space
```

The VCHAR (from [RFC5234] (Crocker, D. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” January 2008.)) and TCHAR rules give basic character types from varying sub-sets of the ASCII character set for use in various commands and responses.
Note that in ABNF, string literals are case insensitive. That convention is preserved in this document, and implies that FTP commands and parameters that are added by this specification have values that can be represented in any case. That is, "HASH" is the same as "hash", "Hash", "HaSh", etc., and "ftp.example.com" is the same as "Ftp.Example.Com", "fTp.eXample.cOm", etc.

2.2. Server Replies

TOC

Section 4.2 of [RFC0959] (Postel, J. and J. Reynolds, “File Transfer Protocol,” October 1985.) defines the format and meaning of replies by the server-PI to FTP commands from the user-PI. Those reply conventions are used here without change.

```
error-response = error-code SP *TCHAR CRLF
error-code     = ("4" / "5") 2DIGIT
```

Implementers should note that the ABNF syntax (which was not used in [RFC0959] (Postel, J. and J. Reynolds, “File Transfer Protocol,” October 1985.)) used in this document, and other FTP related documents, sometimes shows replies using the one line format. Unless otherwise explicitly stated, that is not intended to imply that multi-line responses are not permitted. Implementers should assume that, unless stated to the contrary, any reply to any FTP command (including QUIT) can be of the multi-line format described in [RFC0959] (Postel, J. and J. Reynolds, “File Transfer Protocol,” October 1985.).
Throughout this document, replies will be identified by the three digit code that is their first element. Thus the term "500 reply" means a reply from the server-PI using the three digit code "500".

3. The HASH Command (HASH)

TOC

A new command "HASH" is added to the FTP command set to allow the client to request the cryptographic hash of a file from a server-FTP process.
The syntax for the HASH command is:

```
hash-command = "HASH" SP <pathname>
```

As with all FTP commands, the "HASH" command word is case independent, and MAY be specified in any character case desired.
The HASH command keyword MUST be followed by a single space (ASCII 32) followed by the pathname.
The pathname argument should reference the same file as other file based commands such as STOR or RETR which the same argument would reference. The pathname argument MUST represent a file path, not a directory path.
The text returned in response to the HASH command MUST be:

```
hash-response = hash-ok / error-response
hash-ok       = "213" SP hashname SP 1*HEXDIGIT SP <pathname> CRLF
```

All hash values MUST be encoded in lowercase hexadecimal format.
The HASH command uses the currently selected hash algorithm. The currently selected hash algorithm can be determined with FEAT or OPTS HASH, and changed with OPTS HASH.
The HASH command is meant to be used for files transmitted in Image type mode (TYPE I) and Stream transfer mode (MODE S). The returned hash MUST be calculated as if a client were to download the full file using TYPE I and MODE S and were to calculate the hash on the received octet data. In other words, if a client were to download a full file using TYPE I and MODE S and were to calculate the hash on the received octet data, it would be identical to the hash returned by HASH.

3.1. FEAT Command Response for HASH Command

TOC

When replying to the FEAT command [RFC2389] (Hethmon, P. and R. Elz, “Feature negotiation mechanism for the File Transfer Protocol,” August 1998.), a server-FTP process that supports the HASH command MUST include a feature line indicating that the HASH command is supported, along with a list of all supported hash algorithms in a semicolon separated list. The hash algorithm that is currently selected MUST be marked with an asterisk. The order of hash algorithms is insignificant. This command word is case insensitive, and MAY be sent in any mixture of upper or lower case, however it SHOULD be sent in upper case. That is, the response SHOULD be:

```
C> FEAT
S> 211-Extensions supported:
S>  ...
S>  HASH SHA-256;SHA-512;SHA-1*;MD5
S>  ...
S> 211 END
```

The ellipses indicate place holders where other features may be included, and are not required. The one-space indentation of the feature lines is mandatory [RFC2389] (Hethmon, P. and R. Elz, “Feature negotiation mechanism for the File Transfer Protocol,” August 1998.). The IANA registry named "Hash Function Textual Names" defines values for hash algorithms. Hash names SHOULD be presented in uppercase, but comparisons should be case-insensitive, e.g. MD5, md5, Md5 are all the same.

```
hash-feat = SP "HASH" SP hashlist CRLF
hashlist  = 1*( hashname ["*"] ";" )
hashname  = 1*( hchar )
hchar     = ALPHA / DIGIT / "-" / "_" / "/" / "." / ","
```

3.2. OPTS Parameters for HASH

TOC

To query the current hash algorithm and to change it, the OPTS command as defined in [RFC2389] (Hethmon, P. and R. Elz, “Feature negotiation mechanism for the File Transfer Protocol,” August 1998.) is used with HASH as the first argument.
If no second argument is passed, OPTS HASH simply returns the currently selected hash algorithm.

```
C> OPTS HASH
S> 200 SHA-1
```

To change the algorithm, a valid hash algorithm MUST be given as second argument. A list of valid hash algorithms is available via the FEAT command. If the command is successful, all future calls to HASH until the next successful OPTS HASH command or until the session is reinitialized (REIN) will use the selected hash algorithm.

```
C> OPTS HASH SHA-512
S> 200 SHA-512
```

Requesting unknown or unsupported algorithms produces an error response.

```
C> OPTS HASH CRC-37
S> 501 Unknown algorithm, current selection not changed
```

The syntax for OPTS HASH:

```
hashopts-cmd      = "OPTS HASH" [ SP hashname ] CRLF
hashopts-response = hashopts-ok / error-response
hashopts-ok       = "200" SP hashname CRLF
```

3.3. User-PI usage of HASH

TOC

The user-PI issues the FEAT command to query the server-PI about which algorithm is currently selected. This also reveals the other algorithms that the server supports. In this example, the SHA-1 algorithm is currently selected, as indicated by the asterisk.

```
C> FEAT
S> 211-Extensions supported:
S>  ...
S>  HASH SHA-256;SHA-512;SHA-1*;MD5
S>  ...
S> 211 END
```

OPTS HASH is an alternative method for the user-PI to query the server-PI about which algorithm is currently selected.

```
C> OPTS HASH
S> 200 SHA-1
```

In this example, we wish to select SHA-256, a different algorithm.

```
C> OPTS HASH SHA-256
S> 200 SHA-256
```

The user-PI requests the cryptographic hash of a file with HASH command. Server-PI replies with the Positive Completion code, the currently selected HASH algorithm, the requested hash of the file, and the pathname of the file.

```
C> HASH filename.ext
S> 213 SHA-256 f0ad929cd259957e160ea442eb80986b5f... filename.ext
```

Client downloads file. Client hashes the downloaded file and compares its hash to the hash obtained from the server. The HASH command could

also be used to verify that an uploaded file has the same hash as the local file.

3.4. HASH Command Errors

TOC

The server-PI SHOULD reply with a 450 reply if the server is busy, e.g. already hashing other files yet inviting the client to retry in the future.
The server-PI SHOULD reply with a 500 reply if the HASH command is unrecognized or unimplemented.
The server-PI SHOULD reply with a 501 reply to the OPTS HASH command if the user-PI has requested an unknown or unsupported algorithm.
The server-PI SHOULD reply with a 550 reply if the HASH command is used on a file that can not be found.
The server-PI SHOULD reply with a 551 reply if the server-PI can not calculate the hash of a file because it is unable to deliver the file with TYPE I and MODE S.
The server-PI SHOULD reply with a 552 reply if the user is not allowed to use the HASH command.
The server-PI SHOULD reply with a 553 reply if the user requests the HASH of a directory, which is not allowed.
The server-PI SHOULD reply with a 556 reply if the HASH command is used on a file that cannot be processed for policy reasons. (For example, if the file size exceeds the server's hashing policy.)

---

4. IANA Considerations

TOC

This new command is added to the "FTP Commands and Extensions" registry created by [RFC5797] (Klensin, J. and A. Hoenes, “FTP Command and Extension Registry,” March 2010.).
Command Name: HASH
Description: Cryptographic Hash of a file
FEAT String: HASH
Command Type: Service execution
Conformance Requirements: Optional
Reference: This specification

---

5. Implementation Requirements

TOC

All conforming implementations MUST at least support the SHA-1 algorithm [FIPS-180-3] (National Institute of Standards and Technology

(NIST), “Secure Hash Standard (SHS),” October 2008.). Implementations SHOULD NOT make any algorithm the default that is known to be weaker than SHA-1. Support for any additional algorithms is OPTIONAL.

---

6. Security Considerations

TOC

The server MUST only allow the HASH command to be processable for files which the logged in user has a right to access.
Implementing the HASH command may impose a considerable load on the server, which could lead to denial-of-service attacks. Servers have, however, implemented this for many years, without significant reported difficulties. On an affected server a malicious user could, for example, continuously send HASH commands over multiple connections and thus consume most of the FTP server's resources, leaving little room for other operations. To mitigate this risk, a server MAY cache the calculated hashes so that the hash of a file is only calculated once even if multiple hash requests are sent for that file, provided it updates or invalidates the cached hash when the content of the corresponding file changes. A server may refuse to process a HASH command for many reasons, one of which may be a suspected denial-of-service attack. A client MUST be able to understand that refusal to process HASH commands may be transient (if indicated by a 4yz response) and MAY be honoured later if the server so decides. A client MUST allow that a HASH command might take a reasonably long time to complete.
Server operators might wish to allow the HASH command but restrict its use to certain files, for example, if the file size exceeds the server's hashing policy. A client MUST be able to understand that refusal to process HASH commands may be permanent (if indicated by a 556 response) and will not be honoured later.
In addition, the HASH command can be used to draw conclusions about the contents of a file. If the hash of a file on some server matches the hash of some known file, then both files are likely identical. By uploading a file, running HASH against it and running HASH against another file location, the client could infer some filesystem deployment information (e.g. that there is a logical link between a pair of directories in the tree). This is probably not an issue if the user has access to both branches of the directory tree, there is the possibility that this information is exposable. To prevent this scenario it suffices to limit use of the HASH command to users who would already be able to download the file.
This mechanism simply allows the FTP protocol to expose HASH values of files, using the currently chosen mechanism, accessible to the server by the client. The suitability or otherwise of a specific hash algorithm for a specific purpose is an implementation decision.

---

7. References

TOC

7.1. Normative References

TOC

[FIPS-180-3]	National Institute of Standards and Technology (NIST), “Secure Hash Standard (SHS),” FIPS PUB 180-3, October 2008.
[RFC0959]	Postel, J. and J. Reynolds, “File Transfer Protocol,” STD 9, RFC 0959, October 1985.
[RFC2119]	Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997.
[RFC2389]	Hethmon, P. and R. Elz, “Feature negotiation mechanism for the File Transfer Protocol,” RFC 2389, August 1998.
[RFC3659]	Hethmon, P., “Extensions to FTP,” RFC 3659, March 2007.
[RFC5234]	Crocker, D. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” STD 68, RFC 5234, January 2008.

7.2. Informative References

TOC

[RFC3230]	Mogul, J. and A. Van Hoff, “Instance Digests in HTTP,” RFC 3230, January 2002.
[RFC5797]	Klensin, J. and A. Hoenes, “FTP Command and Extension Registry,” RFC 5797, March 2010.
[draft-twine-ftpmd5]	Twine, J., “The MD5 and MMD5 FTP Command Extensions,” draft-twine-ftpmd5-00 (work in progress), May 2002.

Appendix A. Acknowledgements and Contributors

TOC

Thanks to John C. Klensin, Alfred Hoenes, James Twine, Robert McMurray, Mathias Berchtold, Tatsuhiro Tsujikawa, and Paul Ford-Hutchinson. Portions of [RFC3659] (Hethmon, P., “Extensions to FTP,” March 2007.) were wholly reused in this document.

TOC

Appendix B. List of Non-standard Cryptographic Hash or Checksum Commands and Implementations

[[to be removed by the RFC editor before publication as an RFC.]]
A number of similar checksum or hash commands exist, but are not formally specified, leading to non-interoperability and confusion. The commands, any specifications, and relevant details:

*CKSM: GridFTP v2 Protocol Description http://www.ogf.org/documents/GFD.47.pdf Usage: OPTS CKSM <algorithm> CRLF. Supports ADLER32, MD5, CRC32.

*MD5/MMD5: Expired Internet Draft [draft-twine-ftpmd5] (Twine, J., "The MD5 and MMD5 FTP Command Extensions," May 2002.) from 2002. Usage: MD5 <filepath> Algorithm specific command. Response codes: 251 positive completion, 500 Command Not Recognized, 502 Command Not Implemented, 504 Command Not Implemented for the Specified Argument.

*SITE CHECKSUM: Usage: SITE check_login SP CHECKSUM SP pathname CRLF. Supports CRC32 and MD5.

*SITE SHOHASH: Usage: site shohash [filename]. Supports MD5. Response codes: 200 positive completion.

*XCRC: By GlobalSCAPE in 2001. http://help.globalscape.com/help/secureserver2/File_Integrity_Checking.htm Usage: XCRC <filename> SP EP. SP is starting point and EP is ending point in bytes and are optional parameters. Algorithm specific command. Response codes: 250 positive completion, 450 Requested file action not taken. (File is busy), 550 Requested action not taken. (File not found, no read permission, SP or EP not correct).

*XMD5: XMD5 <filename> SP EP. Similar to XCRC. Algorithm specific command.

*XSHA, XSHA1, XSHA256, XSHA512: Usage similar to XCRC, although SP/EP usage unknown. Algorithm specific commands.

An incomplete list of FTP clients and servers that have implemented these commands:

*Akamai NetStorage (supports SITE CHKHSH/SITE SHOHASH) p17-18 http://pigdogslow.dyndns.org/NetStorage_UserGuide.pdf

*Apache Ftp Server (supports MD5/MMD5 from draft-twine-ftpmd5) http://cwiki.apache.org/FTPSERVER/documentation.html

*Backup4all Pro (supports XCRC)

*Backup to FTP (supports XCRC)

*BlackMoon FTP Server (supports XCRC) http://
www.blackmoonftpserver.com/portal/readmore/features.html

*C.P.A. Secure (supports XCRC) http://www.cpasecure.com/
CPASecureVsSecureFTP.html

*Cerberus FTP server (supports XCRC, XMD5, XSHA1, XSHA256,
XSHA512) http://www.softpedia.com/progChangelog/Cerberus-FTP-
Server-Changelog-1904.html

*Core FTP Pro (supports XCRC)

*Cross FTP Server (supports MD5/MMD5)

*FileCOPA FTP Server (supports XCRC, XMD5, XSHA1) http://
www.filecopa-ftpserver.com/features.html

*File Watchdogs FTP Server (supports XCRC, XMD5, XSHA1, XSHA256,
XSHA512) http://www.filewatchdogs.com/ftpsitehosting/help/
15559.htm

*FireFTP (supports XMD5, XSHA1) http://fireftp.mozdev.org/
features.html

*FTP Daemon (supports SITE CHECKMETHOD/SITE CHECKSUM) http://
www.pro-bono-publico.de/projects/ftpd.html

*FTP Voyager (supports XCRC) http://www.ftpvoyager.com/XCRC.asp

*Gene6 FTP Server http://www.g6ftpserver.com/en/
information#features

*GlobalSCAPE's Secure FTP Server / EFT Server / CuteFTP clients
(supports XCRC)

*Globus FTP client / Globus Toolkit(supports CKSM) http://
www.globus.org/toolkit/releasenotes/3.2.0/gridftp_notes.html

*GoldenGate FTP (Ftp Full Java Server) (supports XCRC, XMD5,
XSHA1)

*IceWarp FTP Server http://www.icewarp.com/products/ftp_server/

*ICS FTP client (supports XCRC, XMD5) http://www.magsys.co.uk/
delphi/magics.asp

*ioFTPD (supports XCRC)

*JAFS (supports XCRC and MD5) http://www.sbbi.net/site/jafs/features.html

*Kellerman FTP (supports XCRC) http://sharptoolbox.com/tools/kellerman-ftp

*Limagito FTP server (supports XCRC, XMD5, XSHA1) http://www.limagito.com/file-mover-features.html

*MOVEit DMZ (supports XSHA1)

*Nofeel FTP server (supports XCRC, XMD5, XSHA1) http://www.nftpserver.com/history.php

*Null FTP (supports XCRC, XMD5, XSHA) http://www.sharewareconnection.com/null-ftp-client-pro.htm

*Orenosv FTP Client (supports XCRC, XMD5) http://www.orenosv.com/orenosv/ftpcli_en.html

*ProFTPD module mod_digest (supports XCRC, XMD5, XSHA1, SHA256) http://www.smartftp.com/oss/proftpd/mod_digest.html

*PSFTPd Secure FTP Server (supports XCRC, XMD5, XSHA) http://www.psftp.de/psftpd_fo.php

*Quick 'n Easy FTP Server (supports XCRC) http://www.pablosoftwaresolutions.com/html/quick__n_easy_ftp_server_pro.html

*RaidenFTPD32 FTP server (supports XCRC, XMD5)

*Robo-FTP Server (supports XCRC, XMD5, XSHA1) http://kb.robo-ftp.com/change_log/show/61

*SyncBackPro and SyncBackSE (supports XCRC) http://www.2brightsparks.com/syncback/sbpro-changes.html

*Secure FTP Factory (supports XCRC)

*Serv-U FTP Server (supports XCRC) http://www.serv-u.com/help/serv_u_help/additional_ftp_commands_supported_by_serv_u.htm

*SmartFTP client (supports XCRC, XMD5, XSHA, CKSM) http://www.smartftp.com/features/

*Starksoft Ftp Component for .NET / Mono (supports XCRC, XMD5, XSHA1) http://www.starksoft.com/prod_ftp.html

*Titan FTP Server (supports XCRC)

*Turbo FTP (supports XCRC)

*WISE-FTP (supports XCRC) http://www.wise-ftp.com/news/

*WS_FTP client / server (supports XSHA1, server also XMD5, XSHA1, XSHA256, XSHA512) http://ipswitchft.custhelp.com/app/answers/detail/a_id/671/kw/xmd5/r_id/166/sno/1

*wuftpd (supports SITE CHECKMETHOD/SITE CHECKSUM)

*wzdFTPd (supports XCRC, XMD5) http://www.wzdftpd.net/wiki/index.php/Commands

*Zalman FTP Client (supports XCRC) http://www.zalmansoftware.com/download.html

*zFTPServer

Appendix C. Document History

TOC

[[to be removed by the RFC editor before publication as an RFC.]]
Known issues concerning this draft:

*Section needs improvement: which files should a server process a HASH command for a logged in user?

-08 : October 25, 2010.

*New server reply 556: Servers that allow HASH but restrict its use to certain files.

-07 : August 5, 2010.

*Clarify that HASH is only for files, not directories.

-06 : July 9, 2010.

*Change server reply format.

-05 : June 29, 2010.

*Add Basic Tokens and Server Replies subsections from RFC 3659.

-04 : June 11, 2010.

*User-PI usage and command errors sections updated.

-03 : May 21, 2010.

*List of non-standard checksum and hash commands and their implementations.

-02 : April 16, 2010.

*Error codes section.

-01 : April 7, 2010.

*Changing HASH algorithm with OPTS.

*Reference RFC 5797 and add IANA Considerations section.

*Informative Reference to expired Internet Draft (draft-twine-ftpmd5) which attempted to address this issue (it only supported one hash, MD5).

-00 : October 19, 2009.

*Initial draft.

Authors' Addresses

TOC

	Anthony Bryan
	Pompano Beach, FL
	USA
Email:	anthonybryan@gmail.com
URI:	http://www.metalinker.org
	Tim Kosse
Email:	tim.kosse@filezilla-project.org
URI:	http://filezilla-project.org/
	Daniel Stenberg
Email:	daniel@haxx.se
URI:	http://www.haxx.se/