Workgroup: Network File System Version 4
Internet-Draft:
draft-cel-nfsv4-hash-tree-interchange-
format-03
Published: 4 May 2022
Intended Status: Standards Track
Expires: 5 November 2022
Authors: C. Lever, Ed.
Oracle

Attestation of File Content using an X.509 Certificate

Abstract

This document describes a compact open format for transporting and storing an abbreviated form of a cryptographically signed hash tree. Receivers use this representation to reconstitute the hash tree and verify the integrity of file content protected by that tree.

An X.509 certificate encapsulates and protects the hash tree metadata and provides cryptographic provenance. Therefore this document updates the Internet X.509 certificate profile specified in RFC 5280.

Note

Discussion of this draft occurs on the NFSv4 working group mailing list, archived at https://mailarchive.ietf.org/arch/browse/nfsv4/. Working Group information is available at https://datatracker.ietf.org/wg/nfsv4/about/.

Submit suggestions and changes as pull requests at https://github.com/chucklever/i-d-hash-tree-interchange-format. Instructions are on that page.

Status of This Memo

This Internet-Draft will expire on 5 November 2022.

Table of Contents

1. Introduction

Linear hashing is a common technique for protecting the integrity of data. A fixed-size hash, or digest, is computed over the bytes in a data set using a deterministic and collision-resistant algorithm. An example of such an algorithm is [FIPS.180-4].

Filesystem designers often employ this technique to protect the integrity of both individual files and filesystem metadata. For instance, to protect an individual file's integrity, the filesystem computes a digest from the beginning of its content to its end. The filesystem then stores that digest along with the file content. The integrity of that digest can be further protected by cryptographically signing it. The filesystem recomputes the digest when the file is retrieved and compares the locally-computed digest with the saved digest to verify the file content.

Over time, linear hashing has proven to be an inadequate fit with the way filesystems manage file content. A content verifier must read the entire file to validate its digest. Reading whole files is not onerous for small files, but reading a large file every time its digest needs verification quickly becomes costly.

Filesystems read files from persistent storage in small pieces (blocks) on demand to manage large files efficiently. When memory is short, the system evicts these data blocks and then reads them again when needed later. There is no physical guarantee that a subsequent read of a particular block will give the same result as an earlier one. Thus the initial verification of a file's becomes stale, sometimes quickly.

To address this shortcoming, some have turned to hash trees [Merkle88]. A hash tree leaf node contains the linear hash of a portion of the protected content. Interior nodes in a hash tree contain hashes of the nodes below them, up to the root node which stores a hash of everything in the tree. Validating a leaf node means validating only the portion of the file content protected by that node and its parents in the hash tree.

Hash trees present a new challenge, however. Even when signed, a single linear hash is the same size no matter how much content it protects. The size of a hash tree, however, increases logarithmically with the size of the content it protects.

Transporting and storing a hash tree can therefore be unwieldy. It is particularly a problem for legacy storage formats that do not have mechanisms to handle extensive amounts of variably-sized metadata. Software distribution and packaging formats might not be flexible enough to transport this possibly large amount of integrity

data. Backup mechanisms such as tar or rsync might be unable to handle variably-sized metadata per file.

Moreover, we can readily extend network file storage protocols to exchange a hash tree associated with every file. However, to support such extensions, file servers and the ecosystems where they run must be updated to manage and store this metadata. Thus it is not merely an issue of enriching a file storage protocol to handle a new metadata type.

1.1. Combining These Solutions

The root hash of a hash tree is itself a fixed-size piece of metadata similar to a linear hash. The only disadvantage is that a verifier must reconstitute the hash tree using the root hash and the file content. However, if the verifier caches each tree on local trusted storage, that is as good as storing the whole tree. The verifier can then use the locally cached tree to validate portions of the file it protects without reading each file repeatedly from remote or untrusted durable storage.

To further insulate a root hash from unwanted change, an attestor can protect it with a cryptographic signature. This cryptographic protection then additionally covers the entire hash tree and the file content it protects.

This integrity protection is independent of the file's storage format and its underlying durable media. The file (and the root hash that protects it) can be copied, transmitted over networks, or backed up and restored while it remains protected end-to-end.

1.2. Efficient Content Verification

We now have a small fixed-size piece of metadata that can protect potentially huge files. The trade-off is that the verifier must reconstitute the hash tree during file installation or on-demand. File systems or remote filesystem clients can store or cache reconstituted trees in:

*Volatile or non-volatile memory

*A secure database

*A private directory on a local filesystem

*A named attribute or stream associated with the file

An easily accessible copy of a file's hash tree enables frequent verification of file content. Frequent verification protects that content against unwanted changes due to local storage or copying

errors, malicious activity, or data retention issues. When verification is truly efficient, it can take place as often as during every application read operation without a significant impact on throughput.

The current document's unique contribution is the use of an X.509 v3 certificate to encapsulate the representation of a hash tree. The purpose of encapsulation is to enable the hash tree metadata to be exchanged and recognized broadly in the Internet community. Therefore each certificate has to:

*Cryptographically protect the integrity of the hash tree metadata

*Bind the hash tree metadata to the authenticated identity of the file content's attestor

*Provide for a broadly-supported standard set of cryptographic algorithms

*Represent the hash tree data in a commonly recognized format that is independent of storage media

Lastly, we note that a standard representation of hash tree metadata enables opportunities for hardware offload of content verification.

1.3. Related Work

Granted in 1982, expired US patent 4309569 [Merkle82] covers the construction of a tree of digests. Initially, these "Merkle trees" helped improve the security of digital signatures. Later they were used in storage integrity applications such as [Mykletun06]. They have also found their way into other domains. [RFC6962], published in 2013, uses Merkle trees to manage log auditing, for example.

A Tiger tree is a form of a hash tree often used by P2P protocols to verify a file's integrity while in transit. The Tree Hash EXchange format [THEX03] enables the transmission of whole Tiger trees in an XML format. The current document proposes similar usage where a sidecar hash tree protects file content but reduces the integrity metadata's size.

2. Requirements Language

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Hash Tree Metadata

Reconstituting a hash tree (as opposed to building a more generic directed graph of hashes) requires the protected content, a basic set of metadata, and an understanding of how to use the metadata to reconstitute the hash tree:

- *The algorithm used to compute the tree's digests
- *The divergence factor (defined as one for a hash list and two for binary hash trees)
- *The tree height (from root to the lowest leaf node)
- *The block size covered by each leaf node in the tree
- *An optional salt value

More research might be needed to cover recent innovations in hash tree construction; in particular, the use of prefixes to prevent second pre-image attacks.

The digest algorithm used to construct the hash tree **MUST** match the digest algorithm used to sign the certificate. Thus if SHA-2 is used to construct the hash tree, the certificate signature is created with SHA-2. The verifier then uses SHA-2 when validating the certificate signature and reconstituting the hash tree. The object identifiers for the supported algorithms and the methods for encoding public key materials (public key and parameters) are specified in [RFC3279], [RFC4055], and [RFC4491].

The block size value of the tree is specified in octets. For example, if the block size is 4096, then each leaf node of the hash tree digests 4096 octets of the protected file (aligned on 4096-octet boundaries).

The internal nodes are digests constructed from the hashes of two adjacent child nodes up to the root node (further detail needed here). The tree's height is the distance, counted in nodes, from the root to the lowest leaf node.

The leaf nodes are ordered (left to right) by the file offset of the block they protect. Thus, the left-most leaf node represents the first block in the file, and the right-most leaf node represents the final block in the file.

An explanation of the salt value goes here.

Further, when computing each digest, an extra byte might be prefixed to the pre-digested content to reduce the possibility of a second-preimage attack.

4. File Provenance Certificates

RFC Editor: In the following subsections, please replace the letters II once IANA has allocated this value. Furthermore, please remove this Editor's Note before this document is published.

X.509 certificates are specified in [X.509]. The current document extends the Internet X.509 certificate profile specified in [RFC5280] to represent file content protected by hash tree metadata.

File provenance certificates are end-entity certificates. The certificate's signature identifies the attestor and cryptographically protects the hash tree metadata.

The Subject field **MUST** be an empty sequence. The SubjectAltName list carries a filename and the root hash, encoded in a new otherName type-ID, shown below. The current document requests allocation of this new type-ID on the id-on arc, defined in [RFC7299]. The following subsections describe how the fields in this new type-ID are used.

```
id-pkix OBJECT IDENTIFIER ::=
          { iso(1) identified-organization(3) dod(6) internet(1)
                    security(5) mechanisms(5) pkix(7) }
id-on OBJECT IDENTIFIER ::= { id-pkix 8 }
id-on-fileContentAttestation OBJECT IDENTIFIER ::= { id-on II }
FileContentAttestation ::= SEQUENCE {
   treeRootDigest OCTET STRING,
   treeDivergenceFactor INTEGER  (1..2),
   treeHeight INTEGER,
   treeBlockSize INTEGER,
   treeSaltValue OCTET STRING
}
```

4.1. New Certificate Fields

4.1.1. Root Hash

The root digest field stores the digest that appears at the root of the represented Merkle tree. The digest appears as a hexadecimal integer.

4.1.2. Divergence Factor

The value in the tree divergence factor field represents the maximum number of children nodes each node has in the represented Merkle tree. A value of two, for example, means each node (except the leaf nodes) has no more than two children.

4.1.3. Tree Height

The tree height field stores the distance from the represented Merkle tree's root node to its lowest leaf node. A value of one, for example, means the tree has a single level at the root.

4.1.4. Block Size

The block size field contains the number of file content bytes represented by each digest (node) in the Merkle tree. A typical value is 4096, meaning each node in the tree contains a digest of up to 4096 bytes, starting on 4096-byte boundaries.

4.1.5. Salt Value

The tree salt value is a hexadecimal integer combined with the digest values in some way that I have to look up. If the tree salt value is zero, salting is not to be used when reconstituting the represented Merkle tree.

4.2. Extended Key Usage Values

Section 4.2.1.12 of [RFC5280] specifies the extended key usage X.509 certificate extension. This extension, which may appear in end-entity certificates, indicates one or more purposes for which the certified public key may be used in addition to or in place of the basic purposes indicated in the key usage extension.

The inclusion of the codeSigning value (id-kp-codeSigning) indicates that the certificate has been issued for the purpose of allowing the holder to verify the integrity and provenance of file content.

4.3. Validating Certificates and their Signatures

When validating a certificate containing hash tree metadata, validation **MUST** include the verification rules per [RFC5280].

The validator reconstitutes a hash tree using the presented file content and the hash tree metadata in the certificate. If the root hash of the reconstituted hash tree does not match the value contained in the treeRootHash, then the validation fails.

5. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs.

Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

There are no known implementations of the X.509 certificate extensions described in the current document.

6. Security Considerations

It is important to note the narrow meaning of the digital signature in X.509 certificates as defined in this document. That signature connotes that the data content of the certificate has not changed since the certificate was signed, and it identifies the signer cryptographically. The signature does not confer any meaning or guarantees other than the integrity of the certificate's data content.

6.1. X.509 Certificate Vulnerabilities

The file content and hash tree can be unpacked and then resigned by someone who participates in the same web of trust as the original content creator. Verifiers should consult appropriate certificate revocation databases as part of validating attestor signatures to mitigate this form of attack.

6.2. Hash Tree Collisions and Pre-Image Attacks

A typical attack against digest algorithms is a collision attack. The usual mitigation for this form of attack is choosing a hash algorithm known to be strong. Implementers **SHOULD** choose amongst digest algorithms that are known to be resistant to pre-image attacks. See [RFC4270] for a discussion of attacks on digest algorithms typically used in Internet protocols.

Hash trees are subject to a particular type of collision attack called a "second pre-image attack". Digest values in intermediate nodes in a hash tree are generated from lower nodes. Executing a

collision attack to replace a subtree with content that hashes to the same value does not change the root hash value and is more manageable than replacing all of a file's content. This kind of attack can occur independently of the strength of the tree's hash algorithm. The tree height is included in the signed metadata to mitigate this form of attack.

6.3. File Content Vulnerabilities

There are two broad categories of attacks on mechanisms that protect the integrity of file content:

Overt corruption An attacker makes the file's content dubious or unusable (depending on the end system's security policies) by corrupting either the file's content or its protective metadata in a detectable manner.

Silent corruption An attacker alters the file's content and its protective metadata in synchrony such that any changes remain undetected.

The goal of the current document's mechanism is to turn as many instances of the latter as possible into the former, which are more likely to identify corrupted content before it is consumed.

7. IANA Considerations

RFC Editor: In the following subsections, please replace RFC-TBD with the RFC number assigned to this document, and please replace II with the number assigned to this new type-ID. Furthermore, please remove this Editor's Note before this document is published.

7.1. Object Identifiers for Hash Tree Metadata

Following the "Specification Required" policy as defined in Section 4.6 of [RFC8126], the author of the current document requests several new type-ID OIDs on the id-on arc defined in Section 2 of [RFC7299]. The registry for this arc is maintained at the following URL: https://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml#smi-numbers-1.3.6.1.5.5.7.8

Following [RFC5280], the current document requests newly-defined objects in the following subsections using 1988 ASN.1 notation.

```
id-pkix OBJECT IDENTIFIER ::=
          { iso(1) identified-organization(3) dod(6) internet(1)
                    security(5) mechanisms(5) pkix(7) }
id-on OBJECT IDENTIFIER ::= { id-pkix 8 }
id-on-fileContentAttestation OBJECT IDENTIFIER ::= { id-on II }
```

IANA should use the current document (RFC-TBD) as the reference for these new entries.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, DOI 10.17487/RFC3279, April 2002, <https://www.rfc-editor.org/rfc/rfc3279>.

[RFC4055] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, DOI 10.17487/RFC4055, June 2005, <https://www.rfc-editor.org/rfc/rfc4055>.

[RFC4491] Leontiev, S., Ed. and D. Shefanovski, Ed., "Using the GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms with the Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 4491, DOI 10.17487/RFC4491, May 2006, <https://www.rfc-editor.org/rfc/rfc4491>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26,

RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[X.509] International Telephone and Telegraph Consultative Committee, "ITU-T X.509 - Information technology - The Directory: Public-key and attribute certificate frameworks.", January 2019.

8.2. Informative References

[FIPS.180-4] Dang, Q., "Secure Hash Standard", National Institute of Standards and Technology report, DOI 10.6028/nist.fips.180-4, July 2015, <https://doi.org/10.6028/nist.fips.180-4>.

[Merkle82] Merkle, R., "Method of providing digital signatures", January 1982.

[Merkle88] Merkle, R., "A Digital Signature Based on a Conventional Encryption Function", Advances in Cryptology - CRYPTO '87 pp. 369-378, DOI 10.1007/3-540-48184-2_32, 1988, <https://doi.org/10.1007/3-540-48184-2_32>.

[Mykletun06] Mykletun, E., Narasimha, M., and G. Tsudik, "Authentication and integrity in outsourced databases", ACM Transactions on Storage Vol. 2, pp. 107-138, DOI 10.1145/1149976.1149977, May 2006, <https://doi.org/10.1145/1149976.1149977>.

[RFC4270] Hoffman, P. and B. Schneier, "Attacks on Cryptographic Hashes in Internet Protocols", RFC 4270, DOI 10.17487/RFC4270, November 2005, <https://www.rfc-editor.org/rfc/rfc4270>.

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013, <https://www.rfc-editor.org/rfc/rfc6962>.

[RFC7299] Housley, R., "Object Identifier Registry for the PKIX Working Group", RFC 7299, DOI 10.17487/RFC7299, July 2014, <https://www.rfc-editor.org/rfc/rfc7299>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/rfc/rfc7942>.

[THEX03]
Chapweske, J. and G. Mohr, "Tree Hash EXchange format (THEX)", January 2003, <http://www.nuke24.net/docs/2003/draft-jchapweske-thex-02.html>.

Acknowledgments

The editor is grateful to Bill Baker, Eric Biggers, James Bottomley, Russ Housley, Benjamin Kaduk, Rick Macklem, Greg Marsden, Paul Moore, Martin Thomson, and Mimi Zohar for their input and support.

Finally, special thanks to Transport Area Directors Martin Duke and Zaheduzzaman Sarker, NFSV4 Working Group Chairs David Noveck and Brian Pawlowski, and NFSV4 Working Group Secretary Thomas Haynes for their guidance and oversight.

Author's Address

Charles Lever (editor)
Oracle Corporation
United States of America

Email: chuck.lever@oracle.com