opentext™

Tree Hashing

Specification of a new method and format for tree hashing of forensic images proposed by OpenText

Revision 1.1

The Information Company

Contents

1 Overview

With the advent of solid-state hard drives, especially those with a PCI Express interface to the host, drive speeds have been increasing rapidly. For these drives, the limiting factor when creating a forensic image is no longer the drive itself.

Forensic hard drive images typically include one or more sequential hashes over the contents of the hard drive. As drive speeds have increased, calculating these hashes has become one of the major limiting factors to overall imaging performance.

This specification defines a method of tree hashing for forensic images that allows the hash calculation to be parallelized, while keeping the same level of security and integrity as the conventional sequential hashing in use today.

Tree hashing allows the imaging application to make full use of today's multi-core processors and crypto accelerators. The new limit to hashing performance is now the total processing power available to calculate hashes. For example, if you have a 4-core processor, with tree hashing you get a nearly 4x performance improvement over the traditional sequential hash.

2 Tree hashing mode

This section describes the tree hashing mode that the OpenText Forensic (formerly EnCase) and OpenText Forensic Equipment (formerly Tableau) products have implemented.

We chose to use Sakura encoding [1] to format the input to the underlying hash function. This document does not specify a specific underlying hash function. The described tree hashing mode will function the same regardless of the actual hash function chosen.

The Sakura encoding [1] provides proofs that the resulting hash tree is sound, meaning it does not introduce any weaknesses on top of the underlying hash function. Our products use the same hash functions (MD5, SHA1, SHA256) for tree hashing that we use for sequential hashing. This means if you choose SHA1 as the underlying hash function for the hash tree, the Sakura encoding [1] guarantees that the result will have the same level of security and integrity as a standard sequential SHA1 hash.

We have chosen to implement the "Final node growing" tree hash mode described in section 5.1 of the Sakura specification [1]. In this mode the tree has a single level, the image data is broken into fixed size blocks, and we do not employ kangaroo hopping.

The following mode is defined using the formal grammar specified in Sakura Fig. 4 on page 7 [1]. All nodes are padded so they are a multiple of 8-bits to ensure byte-alignment.

Note: The Sakura grammar is bit-oriented and defined from left (least significant bit) to right (most significant bit). This means when converting from bits to bytes it's necessary to reverse the bits, i.e. [1 10 00] becomes 0b00000011 or 0x03.

2.1 Tree Hash Specification

The tree hash only has one level. The leaves (message hops) of the tree are all combined into a single final node.

The image data is broken into fixed sized blocks. The size of these blocks must be a power of two. The size is stored in the E01/Ex01 files as the exponent value. For example, a value of 12 means the block size is $2^{12} = 4096\ bytes$. The valid values for the exponent are in the range $[12 - 22]$. The lower bound of $2^{12} = 4096$ was chosen because 4096 is the least common multiple of the common hard drive sector sizes. The upper bound of $2^{22} = 4\ MiB$ was chosen as a reasonable upper limit. The only exception is the final message hop; it is allowed to have a smaller block size containing the remainder of the image data.

The image data blocks are formed into message hops as follows:

$$CV_N = \langle block\ data \rangle\ 1\ \langle padSimple \rangle$$

If we expand padSimple and pad to an 8-bit boundary, it results in the following:

$$CV_N = \langle block\ data \rangle\ 1\ 1000000$$

$$CV_N = \langle block\ data \rangle\ 0x03$$

The final node is then constructed from all of the message hops, and is formed as follows:

$$F = \langle CV_0 \rangle \ldots \langle CV_N \rangle\ \langle coded\ nrCVs \rangle\ \langle interleaving\ block\ size \rangle\ 0\ 1\ \langle padSimple \rangle$$

In the final node, the chaining values (hashes) of each message hop are concatenated, and is represented by $\langle CV_0 \rangle \ldots \langle CV_N \rangle$ in the construction above. It is valid to have a single chaining value in the final node in the case where the entire image was smaller than a single block.

The chaining values are followed by the number of chaining values present in the final node. This number is encoded as an integer/length pair. The integer is a byte string encoded using the I2OSP(X) function specified in the RSA Labs standard PKCS#1 [2]. The length is a single byte that encodes the length in bytes of the integer field. While I2OSP and the Sakura specification [1] allow for this value to vary in size, to simplify implementation, we have fixed the size at 8 bytes (64 bits). The puts an upper limit on the total message length of $2^{64} * \langle block\ size \rangle$.

This mode does not use interleaving blocks, so the interleaving block size should be set to the maximum value of 0xFF 0xFF.

As an example, if the hash tree has three message hops, the final node would be the following:

$F = \langle CV_0 \rangle \langle CV_1 \rangle \langle CV_2 \rangle$ 0x0000000000000003 0x08 0xFF 0xFF 0 1 100000

$F = \langle CV_0 \rangle \langle CV_1 \rangle \langle CV_2 \rangle$ 0x0000000000000003 0x08 0xFF 0xFF 0x06

2.1.1 Tree hash overhead

Overhead is defined as the amount of data that needs to be hashed in addition to the actual disk data. Tree hashing has more overhead than standard sequential hashing. If you do not have the processing resources to parallelize the hashing of message hops, you should not expect a performance improvement over sequential hashing. However, this small overhead is more than made up for by the ability to parallelize hashing of the message hops.

The amount of overhead in bytes can by calculated by:

$$Overhead = (N \times (\langle digest\ size \rangle + 1)) + 12$$

In this equation, N is the total number of message hops. The digest size is dependent on the underlying hash function used.

As an example, if you are hashing an 8 TiB disk, using a block size of 512 KiB, N would equal 16,777,216 message hops. If you chose SHA1 as the hash function, that would give you a digest size of 20 bytes. Plugging these numbers into the equation above gives us approximately 336 MiB of hashing overhead.

2.1.2 Tree hash example

In this example we'll use the SHA1 hash algorithm. We will specify a block size of 4 bytes. The resulting SHA1 hashes have been truncated for space, the first and last 4 bytes of each hash are shown. The data we will use is shown below.

```
[ 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
  0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13 ]
```

First the data is split into blocks of 4 bytes and the initial message hops are calculated by adding a '1' bit to the data and padding to an 8-bit boundary:

```
CV0 = SHA1(0x00 0x01 0x02 0x03 0x03) = 732a3dbd … b3c67ad0
CV1 = SHA1(0x04 0x05 0x06 0x07 0x03) = 02b5b7a5 … 9d0f8bb1
CV2 = SHA1(0x08 0x09 0x0A 0x0B 0x03) = 662ba6b1 … 5ee8b9e6
CV3 = SHA1(0x0C 0x0D 0x0E 0x0F 0x03) = 0ccf5ada … 4b1127fc
CV4 = SHA1(0x10 0x11 0x12 0x13 0x03) = 03adc471 … bfa260eb
```

Next, the final node is calculated by concatenating the chaining values, the encoded number of CVs, the interleaving block size, the bits '01', and padding to an 8-bit boundary:

```
F = SHA1(CV0 … CV4 0x0000000000000005 0x08 0xFF 0xFF 0x06) =
       ff655172 … e9f2d142
```

3 User messaging

Users of sequential hashes expect to see common algorithm names, such as MD5 and SHA1. To maintain naming consistency, we propose the following naming convention: <algorithm>-FNG-<block size exponent>. FNG stands for “Final Node Growing” which is the name given to our hashing mode in the Sakura specification [1]. If the hash algorithm is SHA1, and the hash block size is $2^{12} = 4096$, the hash name provided to users and in logs should be “SHA1-FNG-12”.

4 Tree hashing for full disk images / clones

Many image formats (DD/DMG), along with clones of hard drives, do not provide the ability to store any additional metadata. This makes it impossible to store the block size of the tree hash or to store the hash tree nodes themselves. In this case, a fixed block size of 512 KiB is used, and only the hash of the final node is reported. This block size was chosen to balance parallelism and overhead.

5 Tree hashing for the OpenText Forensic (formerly Encase) EWF (E01, Ex01) file format

This section describes the new links that are used to store the tree hash values. Three new links have been added to support tree hashing. The first required link is present in every segment file and stores the current hash settings. The second required link stores the final hash of the tree and is present in the final segment file. The third link, which is optional, stores the intermediate chaining values of the hash tree, and one or more of these are present in each segment file.

E01/Ex01 files have one additional constraint: the hash block size must be less than or equal to the image block size.

5.1 Hash settings link

This link stores the hash settings that were used for the image. It records the hashing mode, which hash algorithms were used, and any required hash settings. This link is stored in every image segment file. The data is stored as a simple structure defined in Table 1. All multi-byte fields are encoded little-endian.

Field Name	Field Offset (bytes)	Field Size (bytes)	Field Description
Version	0	2	Version of this data structure. 0 = Invalid 1 = Specifies the structure version described in this document.
Mode	2	2	Hash mode of the image. 0 = Sequential 1 = Final Node Growing Tree
Algorithms	4	2	This is a bitfield specifying the hash algorithms used. 0 = MD5 1 = SHA1 2 = SHA256 3-15 = Reserved
HashBlockSizeExponent	6	2	Specifies the exponent to determine the hash block size. Must be <= image block size. Size = $2^{HashBlockSizeExponent}$

Table 1

This link is stored uncompressed and check-summed. See Table 2 for the link tags used.

E01 Link Tag	Ex01 Link Tag	Link Data Size	Link Description
“hash_settings”	0x49	8	Image Hash Settings

Table 2

5.2 Links storing the final hash

These links are required, and there is exactly one link for each hash algorithm selected when performing imaging. These links are stored uncompressed and check-summed. See Table 3 for the list of new links. Each link stores the hash digest of the corresponding hash algorithm. These links are only present in the final image segment file. The bytes of the digest are stored directly, with no additional formatting/padding.

E01 link tags are limited to 16 bytes, all tags should follow the format “fngt_<ALG>” where <ALG> is replaced with the hash algorithm used in the tree.

E01 Link Tag	Ex01 Link Tag	Link Data Size	Link Description
“fngt_md5”	0x50	16	MD5 final hash
“fngt_sha1”	0x51	20	SHA1 final hash
“fngt_sha256”	0x52	32	SHA256 final hash

Table 3

5.3 Links storing the intermediate chaining values

These links are optional. If present, they contain all the chaining values of the message hops. There should be one or more of these links per hash algorithm selected in each segment file when performing imaging. These links are stored uncompressed and check-summed. See Table 5 for the list of new links. The purpose of these links is to facilitate independent verification of image segment files, to locate corrupt regions of an image, and to fail fast during image verification.

These links are stored as tables, like the existing sector table links. They should immediately follow the sector table link. Each table should provide the chaining values for the data present in the preceding sector data link. These links have a table header, are 32 bytes in size, and are defined below in Table 4.

Field Name	Field Offset (bytes)	Field Size (bytes)	Field Description
StartingOffset	0	8	This is the starting hash block offset for the entries in this table
Count	8	4	Number of entries in the table
Padding	12	4	Unused, shall be 0
Adler32	16	4	Adler32 checksum of bytes 0-15
Padding	20	12	Unused, shall be 0

Table 4

The size of each entry is equal to the size of the hash algorithms digest. The location of the entry in the table maps to the corresponding data blocks. For example, if the block size exponent is $12 = 2^{12} = 4096$, and the "StartingOffset" of the table is 100, then entry 42 corresponds to the 4096-byte data block at offset $(100 + 42) * 4096 = 581632$ bytes into the source disk data. The data in the entry is the chaining value (CV) generated by hashing that data block.

E01 link tags are limited to 16 bytes, all tags should follow the format "fngt_cv_<ALG>" where <ALG> is replaced with the hash algorithm used.

E01 Link Tag	Ex01 Link Tag	Size of table entry	Link Description
"fngt_cv_md5"	0x60	16	MD5 Chaining Values
"fngt_cv_sha1"	0x61	20	SHA1 Chaining Values
"fngt_cv_sha256"	0x62	32	SHA256 Chaining Values

Table 5

6 Legal Information

Patent(s) pending. In order to promote the proliferation and advancement of the technology, Open Text hereby grants a perpetual license to the tree hashing ideas and know-how expressed in this document.

7 References

1. Bertoni G., Daemen J., Peeters M., Van Assche G. (2014) *Sakura: A Flexible Coding for Tree Hashing.* In: Boureanu I., Owesarski P., Vaudenay S. (eds) Applied Cryptography and Network Security. ACNS 2014. Lecture Notes in Computer Science, vol 8479. Springer, Cham. https://doi.org/10.1007/978-3-319-07536-5_14
2. RSA Laboratories, *PKCS # 1 v2.2 RSA Cryptography Standard*, 2012.

About OpenText

OpenText enables the digital world, creating a better way for organizations to work with information, on-premises or in the cloud. For more information about OpenText (NASDAQ/TSX: OTEX), visit opentext.com.

Connect with us:

OpenText CEO Mark Barrenechea's blog

Twitter | LinkedIn