Revisiting the Linear Hash

by
Joe Sylve (BlackBag Technologies)

From the proceedings of
The Digital Forensic Research Conference
DFRWS 2020 USA
Virtual -- July 20-24

Revisiting the Linear Hash

Joe T. Sylve, Ph.D.
Director of R&D

Preface and Request for Comments

- This presentation is not yet a formal proposal and will likely be the subject of a future paper
- At this point, I'm actively seeking comments from interested parties
- If you'd like to work on this with me, please contact me
  - joe@blackbagtech.com
  - @jtsylve

BlackBag™ A Cellebrite Company

Cryptographic Hashing

- Cryptographic hashing of evidence has been common practice since the beginning of the Digital Forensics discipline
- Serves multiple purposes
  - Integrity
  - Identification
  - Attribution
- By far the most common approach to applying cryptographic hashes has been “linear”

Linear Hashing

- With linear hashing, we read every bit of logical data (in order) and apply it to a hashing algorithm of our choice.
- A single flipped or misplaced bit of data should result in a drastically different hash value
- To verify a hash we must repeat the process in the same linear fashion

Modern Evidence Looks Different

- Many of our modern evidence types have areas of unmapped or unreadable data
  - Virtual Memory
  - APFS Fusion Containers
  - Sparsely Allocated Drives and Files
- Modern imaging formats like AFF4 support sparse and out-of-order logical images by mapping data across multiple data streams
  - Image only the useful data in one stream and reorder and fill in the gaps when reading linearly from another

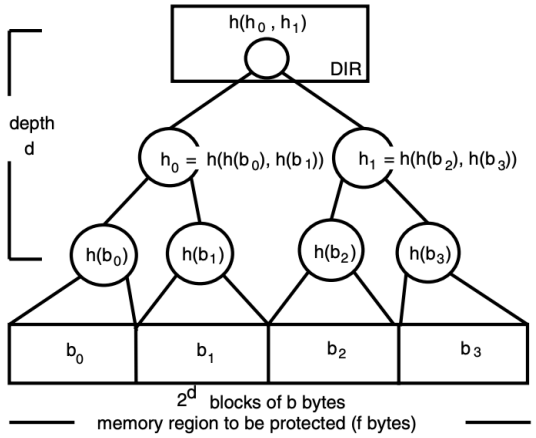
Problems with Linear Hashing

- Serial by nature
  - Limited by single core performance of CPU
  - While single-core speeds have plateaued, evidence sizes and I/O speeds have increased over time.
- Does not handle sparse data well
  - Must feed the algorithm with *something* for the sparse data (usually tons of zero bytes)
- Difficult to "inline" hash sparse and/or out-of-order data

Other Approaches

- Hash the logical image stream (including sparse zeros) after imaging and during validation
  - Linear Hash
- Hash the image file itself
  - Image Hash
- Hash just the logical data that you're adding to the image in the order that it's being imaged
  - Data Hash

Merkle Hash Trees



* Williams and Sirer (2004)

- Hashing can be speed up by hashing individual blocks of data in parallel and then producing a single hash value by hashing the block hashes in a predictable order using binary Merkle Trees
- The final hash value will depend on not only the cryptographic hashing algorithm used, but also the chosen block size

BlackBag™
A Cellebrite Company

The Proposal

- I posit that there is likely no one-size-fits-all hashing solution
- As we start to use non-linear hashing solutions, our tools and evidence image formats need to better communicate our hashing choices
- If investigators are ever going to wrap their heads around this, we probably need to adopt a standardized way of communicating this information in our tools.
  - Rather than just showing the cryptographic algorithm and the hash values alone

What Could This Look Like?

M-<BS>-<A>-<T>

- Red: Merkle Indicator (Optional)
  - BS: Block Size
- Green: Cryptographic Algorithm (Required)
- Blue: Hashing Approach (Optional)
  - L – Linear
  - I – Image
  - D – Data
  - Linear is assumed if not provided

Examples

- SHA-256: d4e1d7b839e49310422d0008feea4d004b621b1b
  - Linear hash of logical data
  - SHA-256 algorithm
- SHA-256-L: d4e1d7b839e49310422d0008feea4d004b621b1b
  - Linear hash of logical data
  - SHA-256 algorithm
- SHA-256-I: 4efe73bdc8e13a137026f64ac13e4879c90032d1
  - Linear hash of image file
  - SHA-256 algorithm
- SHA-256-D: 95eac8871951614d918c8ae80abf2de91b666e34
  - Hash of data stream
  - SHA-256 algorithm
- M-1M-SHA-256: 7cdf3681ece8cd8ca134032045e4e5c09b787687
  - Merkle tree hash of logical data
  - SHA-256 algorithm
  - 1 MiB block size

Extending AFF4

- For our analysis tools to know which methods were used, it will also be important to extend our image formats to support this additional knowledge
- AFF4 currently only supports an *aff4:hash* property that implies a linear hash value on a stream
- MacQuisition supports data hashes by adding a *bbt:integrityStream* property that points to the data stream which contains the linear hash.
  - This could be standardized
- In addition we should consider adopting *aff4:merkleHash* and *aff4:merkleHashBlockSize* properties as alternatives to the *aff4:hash* property when Merkle Tree hashes are used

BlackBag™

A Cellebrite Company

Questions?

joe@blackbagtech.com
@jtsylve

Follow us on social for more webinars, blogs, product releases, tips and tricks and giveaways!