The Hash Box – An Introduction

Robert J. Hughes[1], Thad L Boydston[2], Ryan C Conrad[2], Mitchell J Myjak[2], Glen A Warren[2], Mital Zalavadia[2]
[1]Atomic Weapons Establishment, UK
[2]Pacific Northwest National Laboratory, US

In future nuclear arms control inspections, digital information, e.g., photographs, may be collected to support inspections. This data will likely need to be reviewed by the host before it is released to the inspectors, particularly if the inspection is at a nuclear weapons facility. The inspectors will need to confirm that the data has not been altered to have confidence in the data. Furthermore, the host must be able to redact data before it is released to the inspector to protect sensitive information. An approach was proposed in 2021 to address these concerns by computing simple cryptographic hashes of the data. In this method, hashes of the data are combined into a single root hash that can be reported and confirmed by all parties. In case of redaction, the host can provide the hash(es) of the removed data to enable the inspector to confirm the released data. In addition, the relatively simple implementation resulted in an inexpensive device; on the order of $100. This paper will discuss the approach as well as how it has been implemented on a prototype device.

1.0 Introduction

Future nuclear arms control inspections may involve the collection of digital information, ranging from digital photographs and video to electronic access logs. Even if the data is collected on trusted devices, the transfer of data to the end user needs to be done securely. This requirement is especially true in the situation where the data is reviewed, and potentially redacted by one party before it is given to the other party. For instance, if inspectors record photographs, the host may require the review of the photographs before they are released to the inspectors.

A potential means to securely transfer data from trusted devices to the end user was proposed in (Hughes, 2021). The proposed approach determined cryptographic hashes of the individual data files and then formed a “root hash” of those hashes following a Merkle tree approach. In the Merkle tree approach, a new hash is formed by combining pairs of hashes, then this process is repeated until there is a single hash left, the root hash. As discussed below, we did not implement a Merkle tree but rather a simpler method for forming the root hash. We call the hardware that performs this calculation the “Hash Box.”

The concept of operations is straightforward. Data is collected on a storage device, such as an SD card, from a trusted device. The SD card is then given to the Hash Box, which reads the data, calculates the root hash, and displays the value so all parties can record it. The data can then be taken by one party (presumably the host) for review before it is released. The reviewing party then provides the released data to the receiving party, together with the hashes of any redacted data. Finally, the receiving party recomputes the root hash to confirm that the released data has not been modified. This computation would not be performed on the Hash Box, but rather standalone software that calculates the root hash in the same manner.

Critical to the acceptance of this approach is trust in the Hash Box by both parties. The host will have concerns about the safety of the device, especially if used near nuclear explosives, while both sides will have concerns about the security of the data. Both sides will need to approve of the design of the Hash Box. To hold at risk any alterations from the approved design, the Hash

Box will be given to the other party after it is used so that the other party can conduct detailed inspections of the equipment. For instance, if the inspectors provide the Hash Box, the Hash Box would be provided to the host after it is used. To enable this potentially frequent handing over of equipment, it is important for the Hash Box to be inexpensive to fabricate. The relative low cost of the Hash Box, on the order of $100, is possible by using simple electronics components.

PNNL, under the Quad Nuclear Verification Partnership, has been developing a prototype Hash Box for demonstrating proof of concept. The current version is based on an off-the-shelf microcontroller development board. It will read an SD card and report a 32-byte root hash through an LCD display. Software was also developed on a PC to be able to recompute the root hash with the hashes of any redacted data. The rest of this paper discusses the current design and lessons learned.

Figure 1 depicts a photograph of the proof-of-concept Hash Box. The design uses the Silicon Labs EFM32 Tiny Gecko Series 1 starter kit. The microcontroller on this development board (EFM32TG11) has the following relevant features:

- 48 MHz
- 128 kB program memory
- 32 kB data memory
- On-board cryptography engine
- Integrated LCD controller
- SPI peripherals for the SD card interface

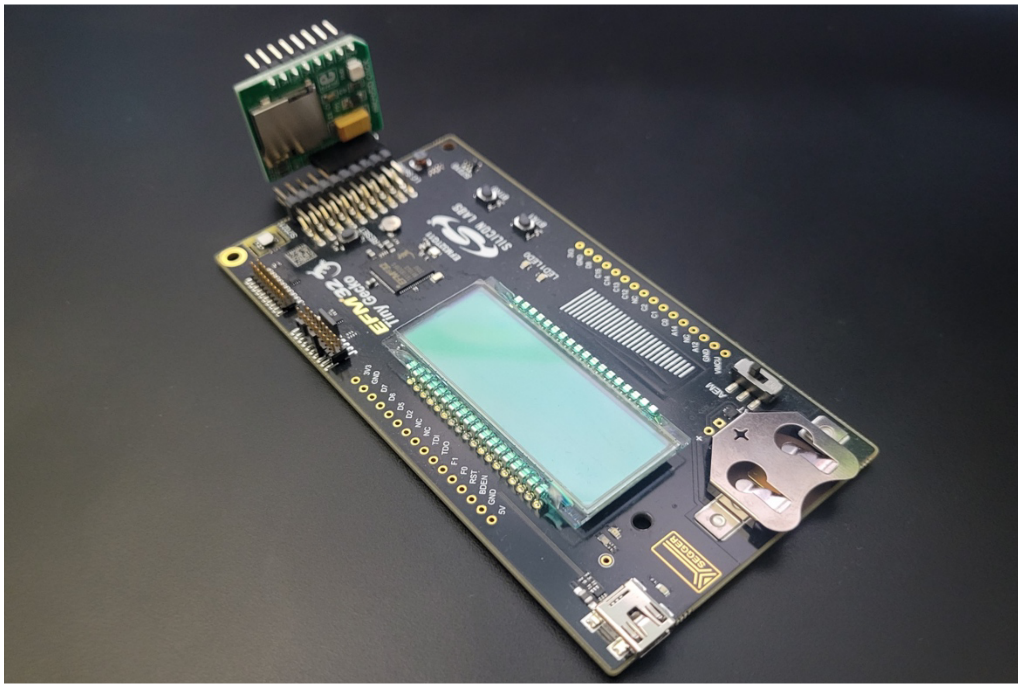


Figure 1: Photograph of the proof-of-concept Hash Box.

The modest processing speed and memory of the microcontroller was a deliberate choice to facilitate authentication. Compared to a general-purpose processor system, malicious actors would not have as large of an attack surface in which to inject additional functionality onto the system. The on-board cryptography engine does accelerate the hash code computations so that they do not become a bottleneck.

Besides the microcontroller, the Hash Box contains a character LCD that displays the root hash as hexadecimal digits, four bytes at a time. The LCD is driven by the microcontroller and does not contain its own controller, which also facilitates authentication. An off-the-shelf SD card adapter is plugged into one of the expansion headers on the board. A USB connector provides power and allows the firmware to be programmed onto the microcontroller. Finally, a reset button causes the microcontroller to restart and execute the firmware. None of the other buttons are used.

2.0 Firmware

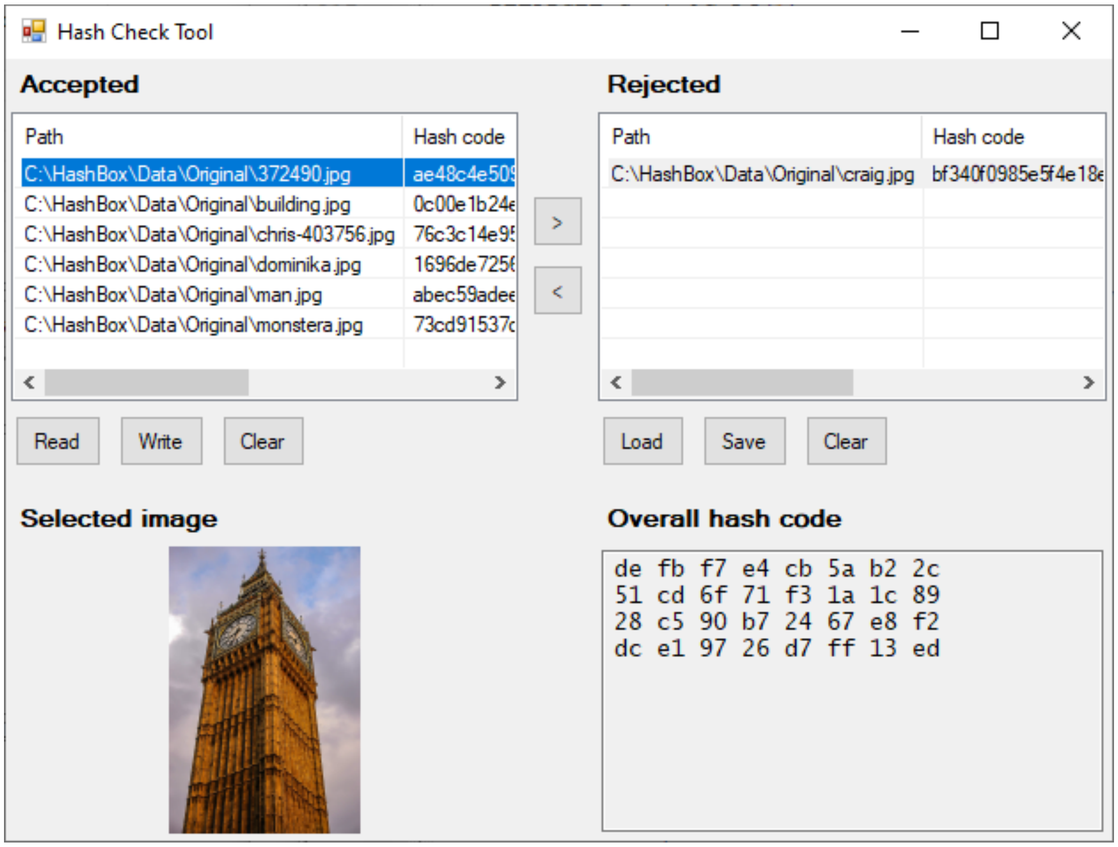
The firmware on the Hash Box is simple by design. After a reset, the microcontroller checks whether an SD card is present. If so, the firmware obtains a list of all files in the root directory. The proof-of-concept system does not check subdirectories at present, although this enhancement is planned for the next iteration. Next, the firmware reads the contents of each file, computes the SHA-256 hash code, and stores the result in a table. These individual hash codes are sorted in numerical order. The firmware then computes the SHA-256 hash code of the sorted table. This result becomes the root hash. Finally, the firmware displays the root hash on the LCD, four bytes at a time, in an infinite loop until the next reset.

Sorting the hash codes in numerical order is a deliberate design choice that simplifies the processing compared to implementing a Merkle tree. Sorting alphabetically by file name, for example, would require the firmware to store extra information for each file in the limited data memory. Information stored in the file metadata, such as the date and location of the picture, is included in the hash calculations. If it is important to preserve the file name or original creation date, they may be added to the calculations as well.

The firmware is designed to have a small memory footprint. Only about 25% of the 128 kB program memory is used. The 32 kB data memory is mostly filled, primarily with the table of individual hash codes. Each hash code is 32 bytes, so the table can contain a few hundred entries, while leaving space for the directory structure and other information. This number should be sufficient for most inspections, although it may impose restrictions if individual portions of each file need to be hashed separately in the future.

3.0 Software

A software utility known as the Hash Check tool allows parties to confirm the root hash computed by the Hash Box, with or without redacted files. The reviewing party first has the software read all files from the SD card and compute the root hash. Files to be redacted are identified and moved to the “Rejected” column. Using the software, the reviewing party copies the accepted files to a new SD card, along with a text file containing the name and hash code of each rejected file.


Hash Check Tool

Accepted

Path	Hash code
C:\HashBox\Data\Original\372490.jpg	ae48c4e50
C:\HashBox\Data\Original\building.jpg	0c00e1b24e
C:\HashBox\Data\Original\chris-403756.jpg	76c3c14e95
C:\HashBox\Data\Original\dominika.jpg	1696de7256
C:\HashBox\Data\Original\man.jpg	abec59adee
C:\HashBox\Data\Original\monstera.jpg	73cd91537c

Rejected

Path	Hash code
C:\HashBox\Data\Original\craig.jpg	bf340f0985e5f4e18e

Selected image

Overall hash code

```
de fb f7 e4 cb 5a b2 2c
51 cd 6f 71 f3 1a 1c 89
28 c5 90 b7 24 67 e8 f2
dc e1 97 26 d7 ff 13 ed
```

Figure 2: Screenshot of the Hash Check software tool.

The receiving party then opens another instance of the Hash Check tool and loads both the accepted files and rejected information. The overall hash code displayed by the software is then compared to the value recorded from the Hash Box. The values should match, as the Hash Check tool performs the same calculations, but inserts the individual hash codes of the redacted files into the table.

The Hash Check tool is presently written as a .NET application for a Windows PC. However, the computations are quite straightforward and can be ported to other languages and platforms easily. In particular, the SHA-256 algorithm is well known and has reference implementations in numerous libraries.

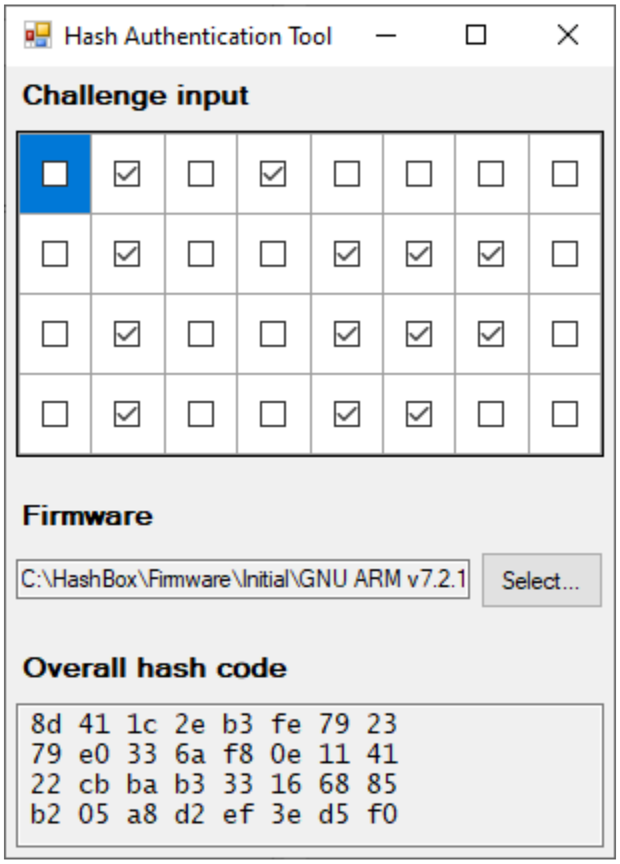
4.0 Demonstration

The prototype hardware has been used to demonstrate the three concepts required by the application: confirming the original data, confirming with redacting data, and identifying altered data. First, an SD card was provided to the Hash Box, which determined and reported the root hash. The SD card was then provided to the PC, which also reported the root hash for confirmation. Next, a second SD card was generated with all but one of the files and the hash of the redacted data. This card was read by the PC, which determined the root hash to demonstrate that the remaining data on the SD card was unaltered. Third, after the root hash of the SD card was reported by the Hash Box, one image of the data was slightly altered (a red circle was added to a jacket lapel), and that data was passed to the PC to determine the root hash. The PC reported a different root hash than the one reported by the Hash Box, indicating that the data had been altered.

5.0 Next Steps

PNNL is currently building a custom board as the next stage in developing the Hash Box. In addition to the custom board, two new features will be evaluated. First, a small printer will be used to print the root hash, in addition to having the value displayed on an LCD screen. The benefit of not recording the root hash by hand will be evaluated against the potential security risks of connecting a printer. Second, ways to validate the firmware on the Hash Box will be explored. In this vision, if the microcontroller is reset without an SD card being present, it would instead read its own firmware from program memory, compute the hash code, and display that value instead.

Simply having the microcontroller compute the hash code of its firmware is insufficient to authenticate the firmware. Because the firmware is fixed and known in advance, malicious parties could alter the firmware to simply display the expected value. For this reason, a set of 32 switches will be added to the hash box to provide a challenge number. These 32 bits or 4 bytes will be inserted before the actual firmware. The challenge number will make it difficult to store all possible hashes of the firmware. Additional techniques for improving the resilience of the system, such as filling unused space in the program memory with pseudorandom numbers, will be explored in future work.



Challenge input

☐	☑	☐	☑	☐	☐	☐	☐
☐	☑	☐	☐	☑	☑	☑	☐
☐	☑	☐	☐	☑	☑	☑	☐
☐	☑	☐	☐	☑	☑	☐	☐

Firmware

C:\HashBox\Firmware\Initial\GNU ARM v7.2.1

Overall hash code

```
8d 41 1c 2e b3 fe 79 23
79 e0 33 6a f8 0e 11 41
22 cb ba b3 33 16 68 85
b2 05 a8 d2 ef 3e d5 f0
```

Figure 3: Screenshot of the Hash Authentication software tool.

6.0 References

Hughes, Robert J. "Review and Redaction-Tolerant Image Verification Using Cryptographic Methods." *Science & Global Security* 29, no. 1 (2021/01/02 2021): 3-16. https://doi.org/10.1080/08929882.2021.1894810.

7.0 Acknowledgements

This work is supported by the U.S. Department of Energy, National Nuclear Security Administration, Warhead Verification Program.