

ARTICLE



## Review and Redaction-Tolerant Image Verification Using Cryptographic Methods

Robert J. Hughes

AWE, Reading, UK

### ABSTRACT

Verification inspections that support nuclear weapon arms control treaties can require photographs to be taken as part of the inspection evidence. In a nuclear weapon facility, the host would generally want to review images before they are released to the inspector to prevent the accidental release of sensitive information. Currently, giving the host sole custody of the images for review severely impacts the confidence of the inspector that the images are genuine and unmodified. This paper outlines how simple cryptographic methods can be employed to allow the host sole custody for review while maintaining the inspector's confidence in the veracity of the images. The concept is then expanded to propose and illustrate the capability for the host to redact a portion of an image while still allowing the inspector to verify that the remainder of the image is genuine. The ability to allow a host to have sole custody of images for review, and even redaction, without affecting inspector confidence in the veracity of the images they receive would contribute to improved processes for nuclear weapon verification in future arms control treaties.

### ARTICLE HISTORY

Received 9 September 2020  
Accepted 2 February 2021

### The problem

In an arms control verification inspection, the inspectors may wish to take photographs and retain them for analysis after the inspection. Obvious examples include documentation of ambiguities, photographs of seals to check for tampering, and photographs of features that might be considered “unique identifiers” (UID) of items.<sup>1</sup> Historically, photography has been used in arms control verification inspections primarily to document any objects or facilities related to ambiguities that cannot be resolved during the inspection. In both the Conventional Armed Forces in Europe (CFE) Treaty and the Intermediate-Range Nuclear Forces (INF) Treaty, photography has been allowed as long as the camera produces duplicate, instantly developed photographic prints, one each for the inspector and host party.<sup>2</sup>

However, in future nuclear weapon verification scenarios it is probable that digital photography's increased resolution will permit a larger role

beyond the documentation of ambiguities; namely, in verifying the integrity and authenticity of seals and UIDs. The age of instantly developed and duplicate photographic prints has largely passed, with digital images required to provide sufficient resolution for seal and UID comparisons. It is therefore likely that more images would be taken (and potentially in more sensitive facilities) than with previous treaties, prompting host concerns over the unintentional release of sensitive information in the photographs taken during an inspection. To alleviate these concerns, the host may insist on retaining possession of the images to review and may redact sensitive information before releasing the images to the inspectors.

The inspectors are then faced with the issue of confirming that the images released by the host are, in fact, the exact images taken during the inspection. The inspectors will want to rule out several possible modifications, malicious or otherwise, such as image replacement, data corruption, or image manipulation. This paper outlines cryptographic methods that allow host review of imagery while enabling the inspectors to confirm the authenticity of the photos that are supplied by the hosts, post-review. To be useful for nuclear treaty verification, the methods must:

1. Give the inspectors confidence that the images are unaltered, except where clearly redacted
2. Give the hosts confidence that no sensitive information can be derived from the information and images given to the inspectors

This paper begins with an outline of cryptographic hashing, followed by a description of its potential utility in image verification. A hardware and software prototype for following the cryptographic method is described. The ability of Merkle Trees to facilitate verification of partially redacted images is then discussed.

## **Cryptographic hashes**

Cryptographic hashes take an arbitrary length data input and produce a fixed-length output, called a message digest. For SHA-256, a common secure hash function, the message digest is 256 bits long, usually represented as a 64-digit hexadecimal message (see [Table 1](#) for example message digests), and requires no encryption keys.<sup>3</sup> A cryptographic hash function, in this case SHA-256, has four properties that make it useful for validating that a file is unchanged:

- It is a deterministic process, meaning that the same data input (identical at the bit-level) will always produce the same message digest

**Table 1.** Example message digests using SHA-256 (computed using the Python “hashlib” module).

Input message	SHA-256 message digest
Nuclear Treaty Verification	2abf67e751947a71cab8ef33c3f268bfc067d54194247078366334e1b152bc0
Nuclear Treat Verification	3fb55f6190b5602b9c702f810b774179c3c171a88f062e5a78e0f2744d08005a
NTV	b79dfb4f8596daf19d406cb29d7933c9628571b4e5809ea325d44fc3e830034c

- It has a property known as the “avalanche effect” which means that a small change to the input (even a single bit) will produce a radically different message digest
- It is a one-way function, meaning that the 64-character message digest cannot be used to reconstruct the original data input
- It is collision resistant, meaning that it would be computationally infeasible to find any two different inputs,  $m_1$  and  $m_2$ , which would produce the same message digest. This is termed “strong collision resistance.”<sup>4</sup>

As an example of the output from hash algorithms, [Table 1](#) shows the output from three separate strings used as the data input. The three strings have different lengths, but consistent 64-character (i.e., 256-bit) hexadecimal message digests. A single character difference between the first two strings produces vastly different message digests. Nothing can be inferred about the original string from the message digest; this illustrates that hashing is considered a “one-way function.”

It is important to remember that the message digests produced in [Table 1](#) (using the Python “hashlib” module) are deterministic, i.e., the SHA-256 algorithm will always produce the message digest given in the first row if the input is exactly the same character string “Nuclear Treaty Verification.”<sup>5</sup> The same deterministic property is true with any data input, including images, as long as the same hash algorithm is used, such as the SHA-256 standard.

### Single image verification

Cryptographic hashes provide a method through which images can be verified as genuine images that have not been modified in any way, even at the bit level. This would enable the images to be retained by the host for private review while the inspectors would still be able to confirm that the images are genuine, and unmodified, when they receive them. The host and inspector would have to agree upon a hash algorithm to use, for which SHA-256 is a candidate as a widely used, standardized, hash algorithm.<sup>6</sup>

### Concept

The basic concept requires the message digests of each image file to be recorded at the time of image capture. This could be achieved through

either the inspectors writing down the digests as the images are captured or, less onerously, the digest strings could be printed out from the camera, either physically or on removable media. The inspectors then retain the message digest strings to compare with the message digests of the images when they are received from the host. It is important to remember here that the message digest is the output of a one-way function and cannot be used to reconstruct the original data. Upon receipt of the reviewed images, if the message digests match those recorded at the time of image capture the inspectors can be confident that the images are genuine and unmodified. The host, meanwhile, can be confident that no sensitive information is passed to the inspector, either through the message digest or the carefully reviewed image. If the host chooses to withhold a particular image, the inspector is left with only a message digest which cannot be used to reconstruct the image.

### ***Hardware and software implementation and prototype***


The generation of the message digests for each image can be driven by the use of peripheral hardware physically attached to the camera. As a prototype, a DSLR camera was attached to a Raspberry Pi running a Python script that triggered the camera and calculated the message digest of the resulting image as soon as it is saved to the camera.<sup>7</sup> It required the use of open software called “gphoto2” to drive the camera and access the image directly, and then the “hashlib” function in Python to generate the message digests.<sup>8</sup>

The prototype setup was used to take a photo and automatically calculate the message digest, the results of which are shown in [Table 2](#).

After the message digest is calculated, the images themselves remain on the camera’s memory card, with the message digests saved to a memory card in the Raspberry Pi. The message digest memory card can be given to the inspector, while the host retains the images on the memory card inside the camera to take for review.

In the interests of satisfying the security and authentication requirements of both the host and inspector parties, it is suggested that the host provide the empty memory card (limited to a very small capacity, e.g., 256 kB) and camera, while the inspector provides the peripheral hardware which calculates the message digest (shown schematically in [Figure 1](#)). The memory card and peripheral unit are small and inexpensive, which allows for random selection and destructive testing to be employed for authentication purposes.<sup>9</sup> Prior to an inspection, several memory cards and peripheral units can be presented. The host randomly selects one peripheral unit for the inspection, and randomly selects another for authentication (which can

**Table 2.** The image captured, and message digest calculated, by the prototype setup.

Image	Message digest
	61405b2816ededaa0442668326206426fc6bbdb4a7493df12cf0d2317496050

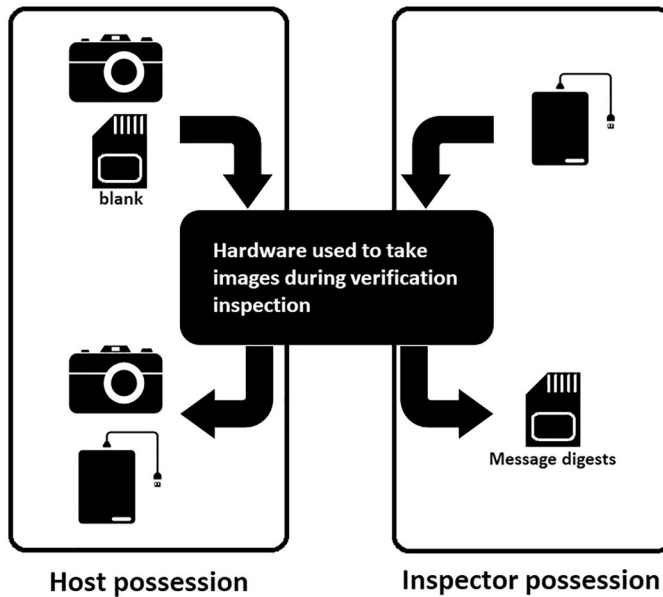
include destructive testing) to ensure that the hardware and software contain no illicit functionality. The inspector gets the same privileges for the memory cards. The items chosen for use in the inspection can be placed under chain of custody measures until the inspection begins. The same process of random selection and destructive testing could also be applied to the camera, albeit at a greater cost. The combination of random selection and destructive testing ensures that any attempts to subvert the correct functioning of hardware and software runs a high risk of detection.

After the inspection, the host party keeps the camera and the peripheral unit for further authentication, while the inspector party can take the low capacity memory card which contains only the message digests. This method assures the host that the memory card cannot contain any imagery, due to the low-memory capacity, and the host is also able to interrogate the peripheral unit during authentication to check for any illicit behavior. The inspector can be certain that the peripheral unit has controlled the camera and calculated the message digests correctly because they have provided the hardware and software to achieve it. By writing the software such that the memory card is formatted at the start of the procedure, the inspector can also be assured that the message digests have not been calculated for pre-staged images and loaded onto the memory card before the inspection (see also the discussion on “replay attacks” in the next section).

The software and hardware requirements described here are minimal and can easily be executed on a Raspberry Pi Zero. For the sake of certification and authentication, it would be preferable to create bespoke hardware and software for the peripheral unit since authentication of the operating system on the Raspberry Pi could be challenging.

### ***Protecting against replay attacks***

There would be a possibility that images could be produced before the inspection and stored on the camera by a malicious host, especially if the most likely images could be predicted, such as closeup images of seals. The genuine message digests of these maliciously prepared images would be



**Figure 1.** A schematic of the proposed process that should assure the inspector and host parties that information is not faked or accidentally disclosed during an inspection. The inspector provides the peripheral unit and receives the low-capacity memory card containing the message digests. The host provides the low-capacity memory card and keeps the peripheral unit for authentication.

provided to inspectors during an inspection such that the malicious images would appear valid when subsequently received by the inspectors. This kind of attack is analogous to a “replay attack” in network security protocols, where a valid message is delayed or repeated by a malicious adversary to circumvent security protocols.<sup>10</sup>

### **Totems**

To prevent this type of attack, there needs to be some element of the data, in this case images, that can be random or controlled by the inspector. An example would be the inclusion of a totem in the image, an object known only to the inspector before the inspection begins. A very crude example would be the inclusion of the inspector themselves in the image. The identity of the inspector is likely not in the host’s control, alongside the inspector’s positioning in the image and any gestures they wish to make. This combination of elements would make it extremely difficult to prepare images (and their message digests) before an inspection begins, thus preventing an easy replay attack.

### **EXIF data**

An additional way to protect against replay attacks would be to require EXIF (exchangeable image file format) metadata in every image and ensure

that the EXIF metadata is included in the hashed data. EXIF is a standard for specific metadata tags added to, among other things, JPEG images recorded by digital cameras.<sup>11</sup> The acceptable metadata tags can include camera settings (such as shutter speed), copyright information, location (such as latitude and longitude), and the date and time, down to the second. The EXIF data could then be used as the element of the image that is under the inspector's control.

The inspector can choose and record, down to the second, the time at which any image is taken. The time is then included in the data within the image file that is hashed, with the message digest given to the inspector. This would be a relatively simple method to further complicate the ability of a malicious host to prepare a replay attack.

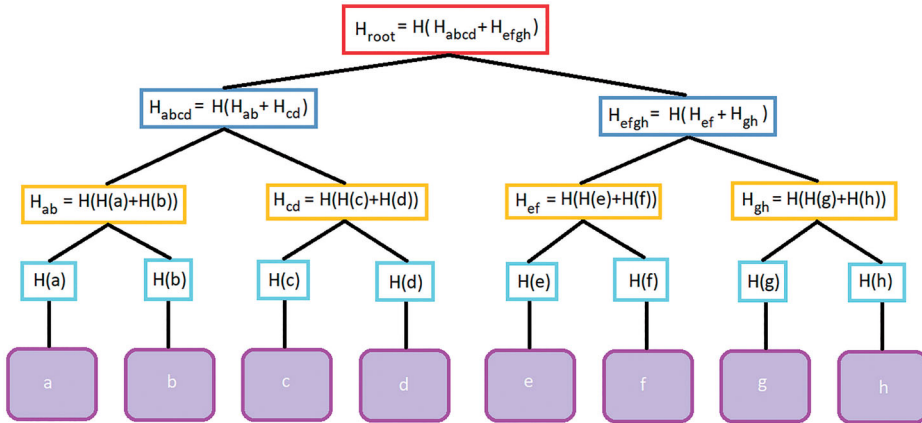
### ***Sophisticated replay attacks***

While this section described two relatively simple ways to protect against a replay attack, it remains feasible that a sophisticated host could preempt every photo and insert the correct totem or EXIF data onto the prepared image as the image is taken. However, the methods outlined here, coupled with the inspector having some influence over the content of the images, and in what order they are taken, increase the cost both financially and technologically in trying to defeat the system. Another potential avenue for mitigating a sophisticated replay attack would be for the inspector to provide the camera and leave it with the host after the event, or for both parties to agree on commercial, off-the-shelf (COTS) cameras. Random selection and destructive testing for authentication, as mentioned previously, could be employed on inspector-provided or COTS cameras to give the host confidence that there is no illicit functionality.

### ***Redaction***

In this case of using simple cryptographic hashes for image verification, any genuine redaction that is required by the host during review of the imagery would render the image untransmissible. Partial redaction of an image, for instance to remove or obscure an object captured accidentally in the background, would constitute a modification of the image data and would result in a different message digest that could not be verified by the inspectors against the list of digests generated during the inspection.

Redaction of images, or even parts of images, may be tolerated by both the inspector and host using another cryptographic tool, Merkle Trees.



**Figure 2.** Schematic of a Merkle Tree for a data file which has been split into 8 blocks (a–h at the foot of the diagram). The “+” sign indicates a concatenation of strings, and  $H(x)$  indicates the message digest of  $x$ .

## Merkle trees

Merkle Trees are used for file verification purposes in many decentralized systems, such as Bitcoin and peer-to-peer file sharing networks. It is a concept that allows portions of data to be verified as genuine without having to evaluate the entire data set.<sup>12</sup>

The Merkle Tree breaks a data file into smaller chunks of an arbitrary size. Each chunk of data is hashed to produce a message digest, as described previously. Each message digest is paired with another data chunk’s message digest, concatenating the two digest strings together. The concatenated string is then itself hashed to produce a new message digest. The hierarchical process is then repeated until there is only one message digest, known as the “Merkle Tree Root.” It is possible to concatenate more than two digest strings together, but the “binary” Merkle Tree, illustrated in [Figure 2](#), is the simplest version.

The benefit of a Merkle Tree is its ability to verify that a particular chunk of data is authentic and is a component of the genuine complete data file, without accessing the complete, genuine file. It requires the Merkle Tree message digests (the root digest and each of the child message digests) to be stored somewhere trustworthy such that they can be used for validation, but it does not require the whole genuine data file to be stored anywhere trustworthy for validation.

As an example, if a user downloads data chunk “b” in [Figure 2](#), they would only require three message digest strings ( $H(a)$ ,  $H_{cd}$ ,  $H_{efgh}$ ) to calculate the root digest for themselves, and compare it to the original root digest ( $H_{root}$ ). In this way, they can confirm by themselves that data chunk



“b” is genuinely a part of the whole data file without knowing the remaining contents of the data file.

## Redaction-tolerant image verification

### *Partial redaction of an image*

A key advantage of the Merkle Tree method in image verification is that it enables the host to redact a part of an image, yet release the redacted image to the inspectors such that the inspector can maintain confidence in the remainder of the image. Merkle Trees, as described in the previous section, are typically used to confirm that a portion of data is a genuine part of a larger data file. In the case of redacted image verification, the Merkle Tree will confirm that the non-redacted parts of the image are from the original image.

Conceptually, the image can be broken up into smaller portions with each portion of the image as the base-layer data blocks (a–h in [Figure 2](#)) in the Merkle Tree. The Merkle Tree root digest is then calculated and given to the inspector during the inspection. If the host chooses to redact one or several portions of the image the host would only have to provide the message digest of each missing image portion to the inspector alongside the non-sensitive portions of the image. Given that a cryptographic hash is a one-way function, the sensitive information in the data file (the image portion) cannot be reconstructed from the message digest. When the inspector receives the released image portions and the message digests of the redacted image portions, if the calculated Merkle Tree Root of the redacted image is not exactly equivalent to the  $H_{\text{root}}$  recorded during the inspection, then either the message digest of the missing portion is incorrect, or the remaining portions of the image have been modified.

The use of the Merkle Tree method allows for partial redaction, and yet would require no extra information to be passed to the inspector at the time of image creation. The inspector will only require the Merkle Tree Root ( $H_{\text{root}}$ ) at the time of image creation, which is the same size as the message digest (e.g., 256-bits for SHA-256) discussed for the cryptographic hash method outlined in the discussion about single image verification.

Using the prototype setup discussed in the earlier discussion on hardware and software implementation and prototype, the open source Python Imaging Library has been added to the software on the peripheral device to take an image from the camera, slice it into eight equal-sized portions and calculate the Merkle Tree root digest.<sup>13</sup> The root digest is then saved to the memory card as with the process in hardware and software implementation and prototype.



**Figure 3.** Left, the original image from Letterpress, showing the inspectors being escorted to a weapon storage area. Right, a close-up showing the area of the image which requires redaction for the purposes of this demonstration.

### ***Partial redaction example***

In 2017, the Quad Nuclear Verification Partnership undertook a simulated verification inspection called “Letterpress.”<sup>14</sup> During this simulation, several hundred photos were taken as part of the inspection to confirm that seals remained intact, and that unique identification labels were genuine. Additional photos were taken to record the event, with one shown in [Figure 3](#), to demonstrate verification of a partially redacted image.

[Figure 3](#) is an image from Letterpress of the inspectors (in white) being escorted to a weapon storage area. For the purposes of this demonstration, the piece of paper being carried by the person at the rear has been deemed to be non-transmissible and must be redacted.

Ordinarily, the redaction of this part of the image would render the whole image non-transmissible. However, if the process outlined in the previous discussion of redaction is followed, then the redacted image can still be transmitted to the inspector, with seven-eighths of the image still verifiable as the original image.

The message digest can be computed for each section of the image using SHA-256. The redaction of element 5 (bottom left in [Figure 4](#)) causes the message digest to change considerably for this element of the image, with the original and new message digests compared below:

**Original element 5:**

*55eac164fbe600586e83b017886f5bb541e10b309a84336a96ccbced1f1a735a*

**Redacted element 5:**

*05afacc5e3286ab83eac367225fe874cd1741cc9832d8c5aafd642e5c745c177*



**Figure 4.** The image split into eight sections, with the redacted section highlighted in red (bottom left).

Using the Merkle Tree process, the Merkle Root for the original (un-redacted) image is:

*9a819007d12033795fedfaf9c0ffed9fd7abc7575c2e5dde3ca6cc9a941d54a4*

This is the message digest that would be recorded and given to the inspector when the photo is taken. Now that element 5 has had to be redacted, the modifications to that element change the Merkle Root such that the image as a whole could not be verified as genuine. The Merkle Root with the modification is now completely different:

*1da5a97efb00d1eba66996dd7298c41e518689c8d2d632fbedcd92b1cf12e803*

However, if we are only given the original message digest for element 5 (and no more information about that element of the image), we are able to reconstruct the entire Merkle Tree and verify the Merkle Root from the other seven-eighths of the image. Thus, the majority of the image can be verified by the inspector even while the information in element 5 can be redacted, or removed entirely, from the information transmitted to the inspector. Due to the nature of the SHA-256 function, neither the new nor original message digest of the sensitive element 5 contains information that could be used to reconstruct the image; thus, the sensitive data is protected.

## Conclusion

Cryptographic techniques can alleviate many of the issues associated with image verification in an arms control inspection context. Typically, allowing the host to have sole custody of data generated during an inspection for any period of time would introduce many concerns for the inspector regarding the integrity and authenticity of the data that is returned.

By utilizing cryptographic hashes of the image data, the inspector can be confident that the returned images are unaltered and genuinely from taken from the inspection. The one-way nature of a cryptographic hash also means that the host can be confident that no sensitive information is put at risk when the message digests for each image taken are given to the inspector before any image review has occurred. These methods are demonstrably able to meet the two requirements outlined in the introduction. The methods must:

1. Give the inspector confidence that the images are unaltered, except where clearly redacted
2. Give the host confidence that no sensitive information can be derived from the information and images given to the inspector

This paper has outlined and prototyped a method by which inspection images can be reliably verified by an inspector even after the host has maintained sole custody for a significant period. At the simplest level, recording the message digests of each image would allow a one-to-one comparison of image data at a later date to give the inspector confidence that the reviewed images are genuine and unaltered at the bit level.

Traditionally, if an image requires partial redaction, then all of the data within the image must be withheld from the inspector. Merkle Trees offer a method through which partial redaction of imagery can be tolerated, with the remaining data able to be confirmed as genuine. If each image is split into a number of sections, then particular sections of an image can be withheld without affecting the ability of an inspector to confirm authenticity of remaining sections.

Key to both of these methods is the ability to record the message digests (or the root digest) as soon as possible after the image is captured. The original image can be altered during the time between the image capture and the recording of the digest. Using the peripheral hardware unit to drive the image capture and the digest calculation immediately after is therefore important. In addition, the inspector must be able to include some element of random, or inspector-controlled, data in the images to complicate and protect against a replay attack by a malicious host.

## Notes

1. Unique identifiers (UID) can be unique features that are intrinsic to an object (such as weld patterns), or an extrinsic feature that is applied to an object (such as an adhesive containing random distributions of reflective material).
2. See *Treaty on Conventional Armed Forces in Europe: Protocol on Inspection*, 1990, 100–101, <https://www.osce.org/files/f/documents/4/9/14087.pdf>, and *the Intermediate-Range Nuclear Forces Treaty: Inspection Protocol*, 1987, <https://www.acq.osd.mil/asda/iipm/sdc/tc/inf/INF-InspProtocol.htm>.
3. National Institute of Standards and Technology, Federal Information Processing Standards Publication, *Secure Hash Standard (SHS)*, FIPS PUB 180-4 (Gaithersburg, MD: U.S. Dept. of Commerce, 2015), <https://csrc.nist.gov/publications/detail/fips/180/4/final>.
4. Harran, Martin, William Farrelly, and Kevin Curran, “A Method for Verifying Integrity & Authenticating Digital Media,” *Applied Computing and Informatics*, 14(2017): 145–158.
5. “15.1. hashlib - Secure hashes and message digests,” The Python Standard Library, Cryptographic Services, <https://docs.python.org/3.5/library/hashlib.html>.
6. SHA-256 is currently used in internet security protocols such as SSL/TLS and in the creation of secret-key message authentication codes. See Ahmad, Imtiaz, and A Shoba Das, “Hardware Implementation Analysis of SHA-256 and SHA-512 Algorithms on FPGAs,” *Computers and Electrical Engineering*, 31(2005): 345–360.
7. Specifically, a Raspberry Pi Zero W, measuring approximately  $37 \times 80 \times 12$  mm, small enough to attach directly to the Canon EOS 1000D camera. See <https://www.raspberrypi.org/products/raspberry-pi-zero-w/> for technical details regarding the Raspberry Pi.
8. “gPhoto<sup>2</sup>, Digital Camera software,” [www.gphoto.org](http://www.gphoto.org), libgphoto2, version 2.5.25. Distributed under the terms of the GNU General Public License.
9. Authentication is the process by which one obtains confidence that the “equipment has not been altered, removed or replaced, and functions such that it provides accurate and reproducible results at all times,” taken from the International Partnership for Nuclear Disarmament Verification, “A Framework Document with Terms and Definitions, Principles, and Good Practices,” Working Group 1: Monitoring and Verification Objectives, November 2017, <http://ipndv.org/wp-content/uploads/2017/11/WG1-Deliverable-One-Final-.pdf>.
10. Kaspersky Resource Center, “What is a Replay Attack?” <https://www.kaspersky.com/resource-center/definitions/replay-attack>.
11. JEITA, Japan Electronics and Information Technology Industries Association Standard, “Exchangeable image file format for digital still cameras: Exif Unified Version 2.32,” JEITA CP-3451E, 2002, Revised May 2019, [https://www.jeita.or.jp/japanese/standard/book/CP-3451E\\_E](https://www.jeita.or.jp/japanese/standard/book/CP-3451E_E).
12. Merkle, Ralph, “A Digital Signature Based on Conventional Encryption Function,” *Lecture Notes in Computer Science*, 293 (1988): 369–378; Yi-Cheng Chen, Yueh-Peng Chou, and Yung-Chen Chou, “An Image Authentication Scheme Using Merkle Tree Mechanisms,” *Future Internet*, 11 (2019): 149.
13. Pillow, a fork of the Python Imaging Library, developed by Alex Clark and Contributors, <https://pillow.readthedocs.io/en/stable/>.
14. Quad Nuclear Verification Partnership: A collaboration between Norway, Sweden, the United Kingdom, and the United States, <https://quad-nvp.info/>; U.K. Ministry of Defence, “UK hosts international nuclear disarmament verification exercise,”

Published October 2017, <https://www.gov.uk/government/news/uk-hosts-international-nuclear-disarmament-verification-exercise>.

## **Acknowledgements**

The author would like to acknowledge Dan Shepherd for discussions on cryptographic methods and reviewing an early draft.

## **Disclaimer**

The views expressed in this document are those of the author and do not necessarily represent those of AWE, the Ministry of Defence, or the Government of the United Kingdom.