Securely Storing and Sharing Memory Cues in Memory Augmentation Systems: A Practical Approach

Agon Bexheti*, Marc Langheinrich*, Ivan Elhart* and Nigel Davies†
**Faculty of Informatics*, *Università della Svizzera italiana (USI)*, Lugano, Switzerland
†*School of Computing and Communication*, *Lancaster University*, Lancaster, United Kingdom
agon.bexheti@usi.ch*, marc.langheinrich@usi.ch*, ivan.elhart@usi.ch*, nigel@comp.lancs.ac.uk†

***Abstract*—A plethora of sensors embedded in wearable, mobile, and infrastructure devices allow us to seamlessly capture large parts of our daily activities and experiences. It is not hard to imagine that such data could be used to support human memory in the form of automatically generated memory cues, e.g., images, that help us remember past events. Such a vision of pervasive "memory-augmentation systems", however, comes with significant privacy and security implications, chief among them the threat of memory manipulation: without strong guarantees about the provenance of captured data, attackers would be able to manipulate our memories by deliberately injecting, removing, or modifying captured data. This work introduces this novel threat of human memory manipulation in memory augmentation systems. We then present a practical approach that addresses key memory manipulation threats by securing the captured memory streams. Finally we report evaluation results on a prototypical secure camera platform that we built.**

***Index Terms*—lifelogging, memory augmentation, memory manipulation, wearable camera, privacy, trusted computing**

I. Introduction

Pervasive computing allows us to digitally capture our daily experiences in increasing numbers and quality – often as part of a process also known as "lifelogging" [1], [2]. Such captured experiences can be used for a variety of purposes, most commonly as part of fitness and health related applications (e.g., the tracking of workouts or sleep), but they can also support human memory [3]. Such systems are typically based on a three-step process [4]: captured experience traces (step 1), e.g., pictures, are used to generate a set of "memory cues" (step 2) – reminders of a past experience that, when being viewed at a later time, trigger the recollection of the event. These cues are repeatedly displayed back to users in an ambient fashion (step 3), e.g., on a mobile phone's lock screen or as a laptop's screen saver, in order to refresh and reinforce existing memories. Ultimately, these memories can then be recalled without the help of any tool.

We have built such a system for capturing visual lifelogs and then using them as memory cues for recalling past memories. An overview of our system and how we envision its use can be found in our prior publications [4]–[8]. One important feature of our system is that it allows users to *share* their memories with other users [9]. While sharing our photos is a process we are very familiar with in order to reminisce together about an event, or show others what they missed (e.g., looking at physical prints together, or sharing digital photos on social media), sharing in our system is meant to enrich memory cue generation by offering a richer set of perspective for a given experience. Here, image sharing happens between people that are experiencing a particular moment *together*. The motivation for this stems from both technical and psychological limitations. First, images taken with a wearable camera (e.g., Sensecam, Narrative Clip) do not always produce good memory cues, as camera lenses can be obscured by hair or clothes, or simply face the wrong way [10]. Second, first-person views from the devices of others may offer a richer view than one's own, due to the narrow field of view of such cameras [11], e.g., a first-person camera will not show who is next to us, while the view captured from the person opposite from us would.

Undoubtedly, such memory-augmentation systems can offer considerable benefits to their users [6]. However, a system that can store all our experiences and influences what we remember about them raises significant privacy and security implications [6], [12]. Apart from the challenge of keeping our memories safe from the prying eyes of others, the threat of memory manipulation might be the most worrisome aspect of this vision: if an attacker is able to remove, add, or change the captured data, the resulting memory cues may implant memories in our heads that never took place, or, in turn, accelerate the loss of other moments by ensuring that no memory cue will ever remind us of them.

There is significant evidence to support the notion that such threats are real – human memory manipulation has been subject to extensive experimental research in psychology, with studies repeatedly demonstrating that human memory is easily manipulated. In recent set of studies, Shaw [13] was able to implant "full false memories" in 70% of her study participants.

Many sources of misinformation have the potential to modify and manipulate one's memories of an event [13], e.g., viewing a set of photos, discussing with others, reading news articles, or even simply reading what others are "tweeting" about an event. In our work, we particularly focus on photographs, since they are easy to capture and make for rich memory cues [4]. Several studies have demonstrated the role of photographs in memory manipulation. Henkel et al. [14] showed that even generic photos (i.e., from a stock catalog)

978-1-5386-9148-9/19/$31.00 ©2019 IEEE

showing a particular task have the potential to trick participants into thinking that they performed such a task (when in fact they did not). Brown and Marsh [15] were able to use photos of different places to manipulate participants' autobiographical experiences, making them believe that they had visited them (while they had not). Wade et al. [16] were able to make participants recall details of a previous hot air balloon ride experience (which never happened) by inserting ("photoshopping") the participant into a photograph of a hot air balloon. Lindsay et al. [17] could incite participants to reminisce about a previous school-related event (which they did not attend) by showing them pictures of the event obtained from their classmates and claiming participants had taken them.

II. Requirements and Attack Model

The aforementioned studies on memory manipulation highlight the fact that our memories can be manipulated with *fake* image data. This introduces significant security implications for memory augmentation systems. In designing a solution for addressing the threat of memory manipulation, we consider the following requirements [9]:

R1: Secure Personal Repositories. Memories (and, eventually, any memory cues generated from them) should be stored in secure and user-controlled repositories. Note that, just like subscribing to an email service or commercial cloud server, not everyone will want (or be able to afford) to host their own memory repository, but will instead subscribe to third-party services able to host their captured memories.

R2: Ensure Integrity and Provenance. Memories captured from personal devices, as well as those received from others, should feature reliable provenance data, i.e., information on the origin and context of capture, as well as information about their integrity, i.e., what changes (if any) have been made to them. This ensures that the captured data is an accurate reflection of what occurred during that experience.

Failing to address these requirements makes a memory augmentation system vulnerable to the following threats that can potentially manipulate user's memories:

T1: Repository Manipulation. An attacker that gains access to a user's memory repository can modify, add, or delete its data. Specifically, an attacker can (1) manipulate *existing* memories, (2) inject *fake* memories, or (3) *delete* a subset of memories (without being noticed).

T2: Receiving Fake Memories. When exchanging memories of an event with other co-located peers, or when obtaining memories of an event that one was not present at, data sharers (either peers with wearable sensors or an infrastructure provider sharing data from a fixed sensor) can behave maliciously by *providing intentionally altered data*.

The envisioned adversary in these attacks can be (1) the repository service provider; (2) a third-party that can compromise a memory repository; or (3) a dishonest user or infrastructure service provider that shares fabricated experience data.

III. Contribution

In this work we present a systematic and practical solution for addressing the threats of memory manipulation (i.e., T1 and T2) by ensuring the integrity and the provenance of memories. Starting from a secure and trusted wearable camera for manipulation-resistant experience capture (section V), we propose *1) a storage protocol to create secure chains of linked images* (section VI) and *2) a zero-knowledge protocol for verifying shared but modified images* (section VII). By using off-the-shelf cryptographic primitives our protocols can efficiently run on low-power wearable cameras, and thus can protect memories from the moment they are captured by a user's device to the time that they are stored in repositories for later review. We assess the protocols' security and demonstrate their practical feasibility (section VIII) using an implementation based on a prototype memory-capture camera.

IV. Related Work

Our work intersects two principal research strands: protocols for securely linking data and zero-knowledge protocols for verifying shared but modified images.

A. Protocols for Securely Linking Captured Data

One way to detect data deletion attacks is to securely link data elements together in a data-chain structure. Traditionally, such schemes are realized using cryptographic *hash chains* [18], [19] where the hash of an item i is calculated using the item itself and the hash of the previous item $i - 1$. Any subsequent data deletion or data modification attempt would invalidate the structure. Prior research has employed this concept for constructing an efficient data authentication protocol suitable for low-power devices [20], [21] in order to prevent fake data injection in broadcast networks.

Other works [22]–[24] have used hash-chain structures to securely link elements together with strong guarantees of their temporal order. They make use of trusted authorities to produce signed timestamp tokens that also depend on tokens issued for previous items. Once an item is timestamped and added to the chain it is impossible to modify its token or remove the item itself from the chain, even by the item owner or the timestamping authority.

While such immutability is a desired property in some contexts, it is a limitation for our envisioned application. We instead propose a lightweight protocol for securely linking captured data that allows *authorized* users to legitimately remove a particular item from their memory repository without invalidating the whole chain structure. However, any instance of *unauthorized* data deletion will invalidate the structure and thus will be detected.

B. Protocols for Verifying Shared but Modified Images

Designing protocols for verification of modified images is an active field of research. In a recent work, Naveh and Tromer propose PhotoProof [25], a protocol for verifying a set of well-defined modifications performed in an original image.

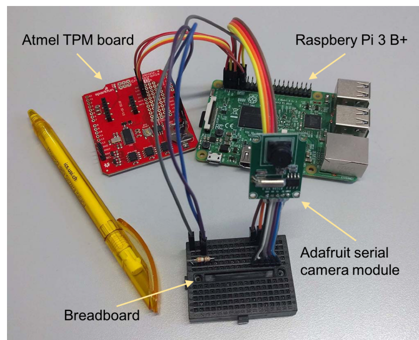

Fig. 1. Our prototypical camera setup used for performance evaluation.

Their solution is based on digital signatures and the *proof-carrying-data* (PCD) concept, a cryptographic primitive for secure execution of distributed computation [26]. After capturing a signed image with a trusted camera, users can sequentially modify the image followed by the computation of a PCD proof for each modification. One can then use the PCD algorithm to verify any modification done on the image. However, while verifying a proof takes less than half a second, generating a PCD proof for even a relatively small image (128×128 pixels) takes about 300 seconds on a relatively powerful[1] machine.

Chabanne et al. [27] propose a similar scheme to verify redacted (obfuscated) pixels of scanned documents. Their solution relies on the *extracted signature* scheme [28], which allows one to remove parts of a previously signed document and re-sign it without the knowledge of the signer's secret key. The extracted signature can be still verified with the signer's public key and without having the removed parts from the original document. While this solution is more efficient than PhotoProof, partially because it supports only one type of image modification (i.e., pixel obfuscation), it is still too complex for low-power wearable cameras. Generating a redaction proof of a gray-scale image with 1200×800 pixels takes 124.5 seconds and 39.7 seconds on a single-core and octa-core system, respectively[2].

We propose a less flexible but more efficient scheme that uses cryptographic hash functions. Similar to the work of Chabanne et al. [27], our protocol only supports the simple "blinding" (i.e., blocking) of certain parts of the image, rather than operations that apply to the whole image (e.g., cropping, color adjustment). Both our own experience, as well as the survey paper from Bettini and Riboni [29], have shown that area blinding is crucial for addressing privacy issues in pervasive systems that capture and store visual data streams.

V. Manipulation-Resistant Memory Capture

The root of trust in our system is a "secure", i.e., trusted, camera. Trusted cameras are a well-known concept in security [30]–[32]. We have built a prototype wearable camera (shown in Figure 1), that uses a Trusted Platform Module (TPM)[3] to securely bootstrap the subsequent authentication and distribution protocols described in sections VI and VII. The TPM enables the camera to guarantee the integrity and provenance of all captured images. In particular, for every captured image I, the camera will log "provenance" data P, such as time and location of capture, and a fingerprint (cryptographic hash) of the captured image. It will then cryptographically "seal" this provenance information by digitally signing a hashed representation of P. The combined raw provenance data and its signature are called the *provenance certificate* $\Pi_I = \{P, Sign_{PK_{priv}}(H(P))\}$. Using the camera's public-key, a third party would then be able to verify the integrity of both the image as well as its provenance certificate at a later time. Any attempt to add external images into users' memory repositories or modify existing images can be easily detected since the suspicious image will lack a valid signature. The rest of this section provides an overall system overview and briefly describes the underlying trust bootstrapping process [33].

A. System Overview

Figure 2 depicts the event flow in our system. At the outset, a user will take "ownership" of a camera (i.e., issue the `TPM_TakeOwnership` command, which triggers the TPM to create its set of root keys). For each captured image of a user's experience, the camera will then provide a signed provenance certificate using its root keys. Later on, when reviewing an experience, the camera owner can then verify that those images have indeed been captured by their camera and that they have not been maliciously modified in the meantime. Moreover, using the file-chain protocol described in section VI below, they can also check if any images have been deliberately deleted, e.g., to reduce the memories of a particular experience.

The camera can also be used to seamlessly *share* its captured images with similar cameras of co-located peers, using a secure protocol that we described in previous work [9]. Our existing system uses a tangible user interface (TUI) with in-situ physical gestures to give users maximum control about both the capture and sharing of experiences – a detailed description of the overall system and its controls can be found in [34]. Since a user might want to modify an image prior to sharing it with co-located others (e.g., block those parts of an image that show their computer screen during a meeting), the camera supports signed image modification proofs (described in section VII). A user that receives an image from a co-located peer can use the provided proof information to verify not only that the image has been captured by a trusted camera during the event, but can also verify which parts of the image have been changed (e.g., blurred) by the peer.

B. Trusting the Camera

In order to establish trustworthiness in our system, we rely on a hardware-based approach for secure platform attestation (ensuring firmware integrity) and secure storage of key material (private keys are not disclosed to unauthorized parties).

[1]Benchmark computer: quad-core CPU at 3.4 GHz and 32 GB of RAM.

[2]Benchmark machines: 1) single-core CPU at 3.6 GHz with 4 GB of RAM; 2) octa-core CPU at 2.9 GHz with 16 GB of RAM.

[3]https://trustedcomputinggroup.org/resources/tpm_main_specification

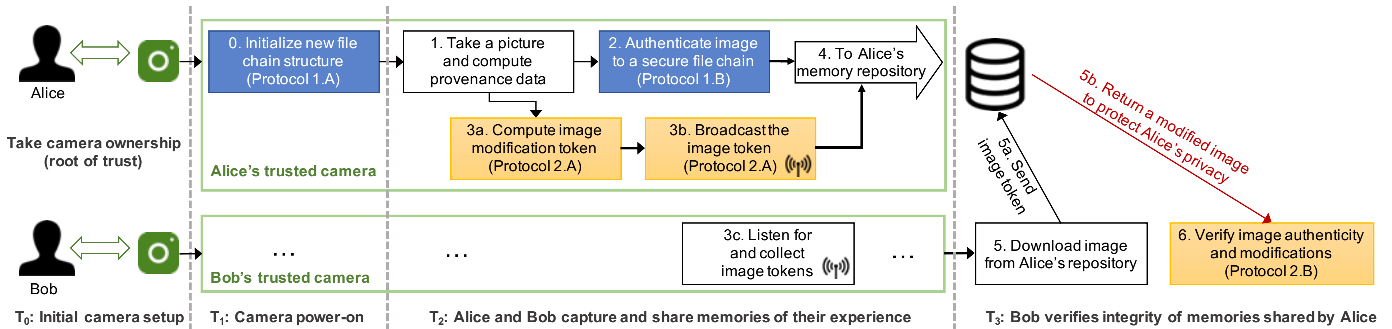

Fig. 2. Overview of the event flow of using our system for securely capturing, storing, and sharing memories with co-located peers.

With the help of the included TPM module, our camera can establish an implicit chain of trust. In this context, the camera proves to its owner – but also to other users in case images are being shared with others – that it is in a *known-good* state. Traditional hardware-based platform attestation schemes [33] allow a user to remotely verify the current state of a device at arbitrary times. However, in our application, it is crucial for a user to know that a camera was in a known-good state at the moment when it captured a picture. Moreover, engaging in a platform attestation protocol each time a new image is captured might introduce significant overhead.

As a result, we modify the traditional interactive platform attestation protocol to a *non-interactive* one as follows. Instead of waiting for a verifier to initiate the protocol, the camera automatically provides an attestation report (obtained from its TPM) every time it captures a picture. In order to ensure that the report is fresh (and not one from a previous time) the camera uses a *fingerprint* of the currently captured image as a nonce parameter when requesting a report from the TPM. The fingerprint is computed right after the image is captured, using a construction that supports also the verification of modified images that the user may receive from co-located peers (the construction is described in detail in section VII). The fingerprint is shared with co-located peers in real-time by means of a short-range wireless broadcast (again with a view towards supporting the verification of images one receives). Finally, the produced platform attestation report for an image is embedded in the image's provenance certificate Π_I.

To check the reported platform state of a camera that captured an image I, in addition to verifying the reports signature, a verifier should also check the report's nonce for freshness. If the image in question was captured by the verifier's own camera, they can then compute a new fingerprint nonce n' from the associated image I, and match it with the nonce of the report (i.e., $n' = n$). When receiving a modified image from a co-located peer, one instead uses the fingerprint that was collected wirelessly during co-location and compare it with the report's nonce.

C. Provisioning and Protecting Key Material

The TPM provides means for ensuring confidentiality and integrity of key material. Every TPM module is provisioned with an Endorsement Key-pair EK, which is stored in tamper-resistant non-volatile memory inside the TPM. The EK is embedded in the TPM as part of its manufacturing, a process which is assumed to be carried out in a secure and trusted environment. During the first-time activation of the camera and its TPM, the EK is used to generate the Storage Root Key SRK, which is also stored inside the TPM. We follow the specifications from the Trusted Computing Group (TCG) and use SRK to derive other application-specific keys. This also happens during the activation of the camera.

D. Implementation

We built a prototype camera to evaluate the feasibility of the proposed protocols (shown in Figure 1). Our camera is powered by the "Raspberry Pi 3 Model B+"[4] IoT platform to which we added a CryptoShield[5] with an "Atmel Trusted Platform Module"[6]. For picture taking we use a serial camera module[7] that provides pre-compressed JPEG images with a maximum resolution of 640×480 pixels. We implemented all of our protocols in Java. Our implementation makes use of jTSS [35], a Java library that implements the software stack proposed by the Trusted Computing Group (TCG)[8] for managing the communication with the TPM.

VI. Protocol 1: Securely Linking Captured Data

Our secure camera prevents an adversary from injecting or manipulating images in a user's repository (i.e., threats T1.1 and T1.2). Each image requires a valid signature that is unique to the user's camera, something an attacker should be unable to produce. The TPM equally prevents an attacker from modifying images directly on the camera, i.e., before even uploading them to the user's repository. However, threat T1.3 – image deletion – can not be prevented this way: an attacker who gains unauthorized access to a user's repository could easily remove important images. Given the large number of captured images and the fact that such data is mainly reviewed a long time after it was captured, it is challenging for a user

[4]https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/
[5]http://learn.sparkfun.com/tutorials/crypto-shield-hookup-guide
[6]https://www.microchip.com/wwwproducts/en/AT97SC3204
[7]https://www.adafruit.com/product/1386
[8]https://trustedcomputinggroup.org

TABLE I
LIST OF VARIABLES USED IN THIS WORK.

Variable	Description
Π_I	Signed provenance certificate for image I.
$EK_{pub/priv}$	TPM endorsement key.
$SRK_{pub/priv}$	TPM storage root key derived from EK.
$PK_{pub/priv}$	Key for signing/verifying Π_I.
Protocol 1	
TK	Symmetric key for computing authentication tokens.
$PTK_{pub/priv}$	Derived from SRK, used to encrypt TK.
S	Camera's unique serial number.
c	Counter initialized to a current Unix time.
Protocol 2	
κ	Tile size for the image splitting procedure.
T_I	Collection of tile fingerprints for image I.
$salt_I$	Random number used when computing tile fingerprints.
τ_I	Hash-based fingerprint of T_I.
$SK_{pub/priv}$	Derived from SRK, used to sign τ_I.
σ_{τ_I}	Signature over τ_I.
R_{ATT}	Camera platform attestation encoded in a TPM-signed report.
Σ_I	Sharing certificate for image I that encodes $T_I, salt_I, \sigma_{\tau_I}$ and R_{ATT}.

to determine if any particular image is missing. While we may not be able to prevent someone who already has unauthorized access from performing any deletion, we want to be able to detect such instances, i.e., we want to know if an image in a series of captured images was removed, and whether it was done by the users themselves or by an unauthorized party.

A. Protocol Description

We propose an efficient scheme to securely link images, captured by users' cameras, in their memory repositories. Protocol 1 describes the underlying steps, Figure 3 provides an overview of the scheme, while Table I itemizes the used identifiers. In our scheme, data is ordered and linked using an incremental counter "c", represented as a hexadecimal number. Two special (empty) files, "*HEAD*" and "*TAIL*", mark the beginning and end of the list, respectively. Removing an element from within the linked list will require either a replacement file to be inserted, or all subsequent elements to be renumbered. Adding a file to the end requires updating the *TAIL* file. Both operations will require secret key material (as described further below) that is only available to authorized users.

To first get a feeling of how large the counter value can get, consider the following estimation. Constantly recording one's life in pictures (with a frequency of 2 photos per minute) will result in 2,880 pictures produced in a day, 1,051,200 pictures in a year and about 73.5 million pictures during a lifetime of 70 years. In order to ensure that, after a camera reset, the list never overlaps a prior one, we always initialize the counter with

Protocol 1 Securely Linking Captured Data

A. Initialize a secure file chain (on camera power-on)

1) $c = UNIX_timestamp$ //initialize counter
2) $newFile(\text{“}'', f_{HEAD} = \text{“}HEAD''||S||c)$
   //create *HEAD* empty anchor file with filename f_{HEAD}
3) $token_{HEAD} = MAC(TK, f_{HEAD})$;
   $newFile(token_{HEAD}, f_{CERT_H} = f_{HEAD}||\text{“}.cert'')$
   //authenticate *HEAD* and store its token in a separate //certificate file with filename f_{CERT_H}
4) $c = c + 1$ //increment counter
5) $newFile(\text{“}'', f_{TAIL} = \text{“}TAIL'||S||c)$
   //create *TAIL* empty anchor file with filename f_{TAIL}
6) $token_{TAIL} = MAC(TK, f_{TAIL})$;
   $newFile(token_{TAIL}, f_{CERT_T} = f_{TAIL}||\text{“}.cert'')$
   //authenticate *TAIL* and store its token in a separate //certificate file with filename f_{CERT_T}

B. Add an element to the secure chain

for each newly captured image I

7) $c = c + 1$ //increment counter
8) $fileRename(I, f_I = \text{“}IMAGE''||S||c||\text{“}.jpeg\text{”})$
   //add c and S to the image's I filename
9) $token_I = MAC(TK, H(I)||S||c)$
   $newFile(token_I, f_{\Pi_I} = f_I||\text{“}.cert\text{”})$
   //authenticate image and store its token in a separate //provenance certificate file with filename f_{Π_I}
10) Update $TAIL$ by executing steps 4-6
11) $commitToBlockchain(f_{TAIL}, f_{CERT_T})$
    //upload *TAIL* and its token certificate file to an //immutable blockchain ledger (optional)

a current timestamp (e.g., Unix epoch, seconds since 1970), which is currently at $\approx$ 1.5 billion. A hexadecimal counter value of 8 characters ($16^8 \approx 4$ billion) would thus be more than sufficient, with another 8-character serial number to differentiate between cameras. The counter can be part of the file's header (e.g., EXIF tag in case of JPEG images) or simply be part of the filename. In case of the latter, a counter of length 16 should easily fit within the maximum length of 255 characters that most standard file systems support, such as Microsoft's NTFS, Apple's HFS, or Linux's ext.

In order to prevent an adversary from replacing or renaming files, any operation on them must be authenticated by an authorized user. For this we rely on the established cryptographic primitive of a message authentication code (MAC) [36],

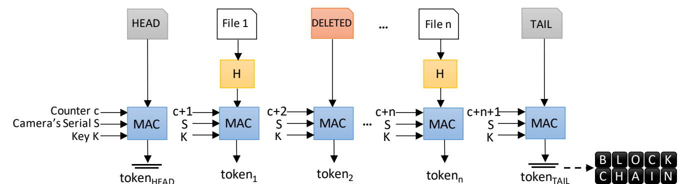

Fig. 3. Each file is linked by a unique counter. The order is then authenticated using a MAC token computed over the file's content and its counter.

which allows one to authenticate a message M by computing an authentication token using a secret key TK. The token can be used later to verify the authenticity of the initial message.

We thus build our linked file structure as follows (also see Protocol 1). At power-on, the camera initializes a counter c (step 1) and creates two empty anchor files for the *HEAD* and the *TAIL*. The anchor files are assigned with a counter value of c and $c + 1$, respectively, and with the camera's unique serial number S, by simply writing these information to the filename (steps 2, 5). This operation is then authenticated by computing a MAC token over their filenames (steps 3, 6).

For each newly captured image I, the camera proceeds as follows. At the outset, the next counter value $c + 1$ (step 7) and the camera's serial number S are assigned to the image by writing them to the image's filename (step 8). This operation is authenticated by first hashing the image's content[9] and then concatenating it with c and S before computing the MAC token over this (step 9). The produced tokens are embedded in the image's provenance certificate Π_I (the provisioning of these certificates was explained previously in section V) which in turn is stored in a separate auxiliary file as explained in step 9 of Protocol 1. Computing the token as a function of the contents of the file and its counter value binds these two together, meaning that an adversary cannot delete a data file and then overwrite the counters of the subsequent files without being noticed. Finally, the structure's tail is updated (step 10).

In case that a legitimate user wants to delete a particular image I_D, and without renumbering all subsequent files from the structure, she can simply use an empty replacement file. After removing both the image file and the corresponding auxiliary certificate file that contains the MAC token, the user creates an empty file and assigns it the same counter as that of the deleted file, by writing it in the filename: $f_{I_D} = [\text{“}DELETED''||S||c]$. She authenticates this by computing a MAC token in a similar way as for the empty anchor files: $token_{I_D} = MAC(TK, f_{I_D})$, and again stores it in a separate file with the same name: $f_{CERT_DELETED} = [f_{I_D}||\text{“}.cert'']$. If the user wants to hide this deletion from a third party, she alternatively can recompute the entire list by replacing all counter values of subsequent images in the list and recomputing the corresponding authentication files.

B. Checking for Missing Images

To check a stream of images for potentially unauthorized deletions, one proceeds as follows. For each image I that was captured between time t_i until time t_j (e.g., all images from the last work meeting): 1) read the filename and corresponding MAC token; 2) using the obtained counter from step 1, recompute a new MAC token and compare it with the token obtained from step 1; 3) check if this file is linked properly with the subsequent file in the given range by verifying that the counter of the previous and subsequent files equal $c-1$ and $c+1$, respectively, and that all serial numbers match the serial number of the desired camera. If, for all files, the computed token matches the token associated to it, one can conclude that the tested data link is valid and intact. A broken link or an unauthenticated one is an indication of a deliberate data deletion attack.

[9]We use a a one-way collision resistant hash function (such as SHA3-256).

C. Generating the MAC token key TK

TK is randomly generated by the TPM during the camera's first-time activation. It is then encrypted with an asymmetric parent key PTK, which in turn is derived from the TPM's storage-root-key SRK. While generating PTK, the camera owner is prompted to provide a password, which allows her later on to obtain a decrypted copy of TK. Both TK and PTK are securely kept inside the camera's internal storage encrypted with each other's parent keys, i.e., PTK and SRK, respectively.

Checking for missing images requires knowledge of the key material TK, which is used to verify MAC tokens. Since this process will be performed outside the camera, (e.g., on a user's personal computer) TK has to be shared with other potentially untrusted computer devices. Managing the storage of TK is outside the scope of this work. However, given a computer with a TPM, a secure key-migration protocol as specified by TCG [37] allows for the secure transfer of TK.

D. Security Analysis

An adversary that wants to delete an image file without the victim noticing has three options: (1) renumber all subsequent files and then create new updated MAC tokens for each of them; (2) create a replacement file ($[\text{“}DELETED''||S||c]$) and compute a valid MAC token for it; or (3) reuse the MAC token of another file that the victim deleted herself. The first two options are prevented by virtue of the secret key TK used to compute MAC tokens. Reusing tokens of other files is not going to help due to the mismatching serial number S and counter c.

An attacker with unauthorized access to the victim's repository can, however, overwrite the whole directory with a previously made backup. In practice, the attacker would "roll back the time" to a much earlier, but valid state, thus making the last n images disappear. To prevent this, the MAC token of the current TAIL can be occasionally committed to an immutable public ledger, i.e., a blockchain (Protocol 1, step 11).

VII. Protocol 2: Verifying Shared but Modified Images

The secure camera and storage protocol described above ensure that an attacker is unable to manipulate a user's memory repository by inserting, modifying, or deleting individual images. However, as we previously argued, being able to share captured experiences with co-located peers can yield tangible benefits for memory cue creation. For a recipient of such shared images it is thus crucial to have some guarantee that the image has not been maliciously (i.e., invisibly) altered.

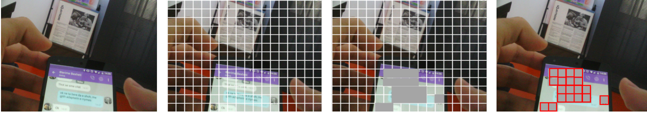

Fig. 4. The process of "blinding" an image region before sharing. From left: (1) the unmodified image I of size 900×650 pixels; (2) the image divided into 18×13 tiles, each made of 50×50 pixels; (3) blocking the tiles that contain information which should not be disclosed; and (4) the final modified image I' which is ready to be shared.

While it is trivial to verify that the image has not been altered (using the secure signature of the peer's camera, which the receiving user can verify), there are perfectly legitimate reasons for a peer to share only a modified version of their captured image. Oftentimes, our own captured images will contain private information (e.g., a document, a phone or laptop screen) that should not be shared with others.

The goal is hence to allow an image recipient to reliably identify all modified parts of an image. The identification of the modified and unmodified parts should obviously be possible without revealing the original, unmodified image – the verifier should only have access to the modified image.

A. Protocol Description

Our proposed protocol builds on the memory augmentation system we reported in [9], which enabled the seamless sharing of lifelogs between co-located users. One element of the sharing protocol we developed for this are *tokens* – random identifiers that each user's camera broadcasts in real-time (using a short-range wireless technology), and which are frequently (e.g., every 5-10 seconds) updated. All images taken by a camera are indexed in a user's repository with the token that was active at the time. In this way, only those who received the tokens are able to query the sharer's memory repository later to retrieve the image.

Our original protocol used random tokens. However, we can also compute them as a function of the actual image content that was just captured. This allows the data sharer to not only regulate access to the image, but to also "commit" the image's content publicly without actually sharing the original image itself. By furthermore signing tokens with the camera's private-key, we can ensure image authenticity. Such "custom" tokens allow us to support the verification of modifications – e.g., obfuscations – to a certain unmodified (but not shared) image. This process is described below (see also Protocol 2), while Table I itemizes the used identifiers.

In order to compute a token τ_I for an image I, the image is first chunked into "tiles" (step 1). A tile is defined as a rectangular area of arbitrary dimensions, and is the smallest area that can be modified (see Figure 4 for an overview). For each such tile we then compute a *hashed fingerprint* following the procedure as in step 3 of Protocol 2. The fingerprints of all tiles are concatenated and hashed again to create the final image token τ_I (step 4), which is immediately announced to co-located peers (step 5) through a short-range BLE broadcast (using a protocol from our previous work [9]).

Protocol 2 Verifying Shared But Modified Images

A. Generate and disseminate image sharing token

for each newly captured image I

1) $tiles_I = splitImage(I, \kappa)$
   //split image into rectangular tiles of size κ
2) $T_I = []$ //empty set for storing tile fingerprints
3) *for* columns i in $tiles_I$
   for rows j in $tiles_I$
   a) $h_{i,j} = H(i||j||\phi(t_{x,y})||salt_I)$
      //where $\phi(t_{x,y}) = p_1||p_2||\ldots p_{m*n}$, a string
      //serialization of all pixels that are in tile $t_{i,j}$
   b) $T_I.add(h_{i,j})$
4) $\tau_I = H(\phi(T_I))$ //where $\phi(T_I) = h_{0,0}||h_{0,1}||\ldots||h_{m,n}$
5) $broadcastToken(\tau_I)$ //broadcast τ_I to co-located peers
   //via BLE using our protocol from [9]
6) $\sigma_{\tau_I} = sign(\tau_I, SK)$
   $R_{ATT} = platformAttest(nonce = \tau_I)$
   // sign τ_I and generate a fresh TPM signed camera
   //platform attestation bound to image I
7) $\Sigma_I = \{T_I, salt_I, \sigma_{\tau_I}, R_{Att}\}$ //encode everything in a
   //sharing certificate Σ_I
8) $uploadData(I, \Sigma_I)$ //upload Σ_I to user's memory
   //repository and link it with image I

B. Verify authenticity and modifications of a shared image

for a token τ_I that was received from a co-located peer

9) $I', \Sigma_I \leftarrow downloadImage(\tau_I)$,
   //obtain an image and its certificate using token τ_I
10) $\{T_I, salt_I, \sigma_{\tau_I}, R_{ATT}\} \leftarrow \Sigma_I$ // extract the certificate
11) if $verifyPlatform(R_{ATT}, nonce = \tau_I)$
    if $verify(\sigma_{\tau_I}, \tau_I, SK)$ && if $H(\phi(T_I)) == \tau_I$
    a) Split the received image I' and compute a tile
       set T'_I following steps 1-3 from above
    b) *for* index i in range of $length(T'_I)$
       if $T'_I[i] \neq T_I[i]$
       $drawFrame(I', T'_I[i])$ //draw a red
       //frame in I' around the area of tile $T'_I[i]$

The set of tile fingerprints, a signature over the final token, as well as the signed platform attestation report R_{Att} obtained from the camera's TPM (step 6), are encoded in a sharing certificate Σ_I (step 7). Note that R_{Att} is generated using token τ_I as a nonce (see section V-B for certificate generation). The final Σ_I together with the captured image are then uploaded to the user's memory repository (step 8).

Before making the image accessible from their repository at a later time, the data sharer can modify it by obfuscating any tile that contains sensitive information (as shown in Figure 4). Accessing the data sharer's repository at the token address τ_I will then yield the modified image I', the tile hash-set T_I of the unmodified image I and the corresponding $salt_I$ (steps 9, 10).

Following step 11 from Protocol 2, the data recipient can now verify which tiles have seen modifications, and which tiles come from the original unmodified image I. At the outset, she will verify that the token τ_I used to access the image is

indeed a hash of the concatenated tile hash-set T_I. Next, she will chunk the received modified image I' (using the same tile size used in the unmodified image I) and then inspect each tile of I' individually using the following procedure. For each tile $t'_{i,j}$ of I', she checks its integrity by computing the tile's hash $h'_{i,j} = H(i||j||\phi(t'_{x,y})||salt_I)$ and matching it with the value given in the corresponding tile hash-set T_I. Now, all modified tiles (i.e., where the hashes do not match) can be marked, e.g., by drawing a red frame around them in the displayed image I', allowing the receiving user to easily verify which tiles have been obfuscated (or otherwise modified).

B. Security Analysis

In order for an attacker to change any tile's content unnoticed, two options exist: (1) to manipulate the image before the tile's hash is computed, or (2) to manipulate it in such a fashion that the tile's hash does not change. The first option is ruled out by virtue of the secure camera hardware. Here, the camera's firmware is attested by the its trusted computing platform TPM, so changing the camera's principal operations should not be possible. The second approach requires the attacker to perform a *second pre-image attack* on the underlying hash function. Given a secure hash function (we use SHA3-256 in our implementation) this should be equally infeasible.

We also need to ensure that the recipient cannot uncover the original contents of a tile, based on the shared information. In order to do this, a recipient would need to perform a *pre-image attack* on the hashes of the obfuscated tiles: given the hash h of a tile that is blocked in the modified image, find a value t such that $H(t) = h$. Given that the hash function concatenates the tile contents with both the tile indices i, j, and a salt, even identical tiles in the original image should hash to different values. A brute force attack is thwarted by the large search space: a pixel is composed of three bytes, one byte for each RGB color. Trying all pixel colors has a time complexity of $256^3 = 2^{24}$ per pixel, hence iterating through a single tile would take $(2^{24})^{mn}$ time, where $m \times n$ are the tile dimensions in pixels. Even the smallest tile size of 5×5 pixels that we evaluated our system on (see section VIII) would require $(2^{24})^{25} = 2^{600} \approx 4 \times 10^{180}$ operations for a single tile. The salt value further eliminates the ability of an attacker to employ so-called "rainbow tables" that trade off time complexity for storage efficiency [38].

VIII. Evaluation

We evaluated our proposed schemes with two different images sizes: (1) a low-resolution image with 640×480 pixels (the maximum resolution of the installed camera module) and (2) a high-resolution image with 4096×3072 pixels (a resolution offered by most of today's wearable camera devices). Images are interpreted using an 8-bit RGB color model, where each pixel is represented by three bytes.

We benchmarked four processes: (1) capturing and storing a picture in the camera's internal storage, (2) obtaining a TPM-signed platform attestation report, (3) appending an image to a secure link structure (following the process described in section VI, Protocol 1); and (4) computing an image modification proof (as described in section VII, Protocol 2). For the modification proof protocol, we additionally measured the overhead of splitting the image into five differently sized tiles. When selecting a tile size, we considered the trade-off between obfuscation granularity and performance efficiency. For the smaller image we started with a tile of 5×5 pixels and increased it up to 160×160 pixels. We applied proportionally larger tiles to the larger image, from 32×32 pixels up to 1024×1024 pixels.

Tests were conducted with a TPM complying to version 1.2 of the standard. All cryptographic operations were carried out with RSA keys of 2048 bits, while all hash operations were computed using SHA3-256. Digital signatures were calculated over a SHA1 hash representation of the data to be signed.[10]

Figure 5 summarizes both runtime and power consumption results. Our proposed protocols work reasonably well on low-power devices. A low-resolution image is captured and processed in less than 25 seconds (3.5 s for taking the photo, 7 s for Protocol 1 and 12 s for running Protocol 2 with the smallest tile size, i.e., 5×5 pixels). Less than 60 seconds are needed for a high resolution image (34 s for taking the photo, 7 s for Protocol 1 and 16 s for Protocol 2 again with the smallest tile size, i.e., 32×32 pixels). The tested camera module did not support such high resolution capture, however, we estimated that it would take about 34 seconds to download a pre-compressed JPEG image of that size with the module's maximum supported transfer-rate of 115,200 bps. The runtime overhead for both generating a platform attestation and appending an image to a secure data structure is constant irrespective of the image size. Regarding the tile size, there is no notable improvement in terms of computational speed by increasing it beyond 20×20 pixels (125×125 pixels for the larger image). This is due to having to hash fewer but larger tiles.

We also measured the energy consumption of the proposed camera using the "Keweisi KWS-V20" USB power tester[11]. The average power consumption for both low-res and high-res images (with medium tile sizes of 20×20 and 125×125 pixels) is 2.1 mAh and 5.6 mAh, respectively. Out of these values, 1.06 mAh and 3.60 mAh were consumed by the Raspberry Pi 3 B+ module alone. Using these power measurements, we estimated the camera operational time (see Figure 6) with a small battery of 200 mAh (same as that of Narrative Clip 2) and a bigger battery with 500 mAh. With a capture frequency of one low-resolution photo per minute, the camera can be operational from 40 hours (smaller battery) up to 100 hours (bigger battery) on a single charge. As for high-resolution photos, the camera can run between 30 hours and 75 hours.

In additional tests, we benchmarked the verification times of our protocols. We ran the benchmarks using high-resolution images (with 4096×3072 pixels) on a machine with a 2.3 GHz

[10]TPM provides support only for signing SHA1 hashes.

[11]The "Keweisi KWS-V20" USB power meter that we used for these evaluation lists an accuracy error of up to 3%, as reported in [39].

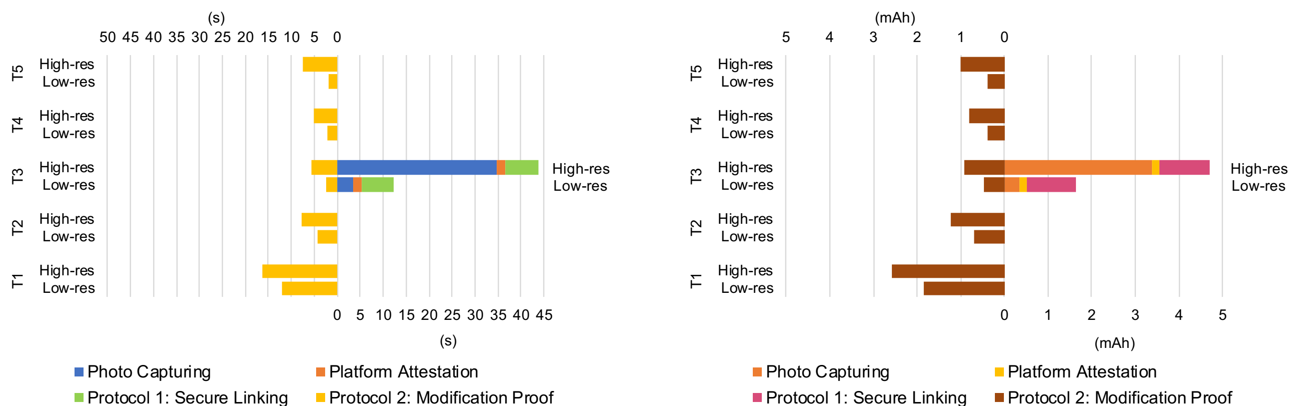

Fig. 5. Left: Implementation runtime overhead (in s); Right: average power consumption (in mAh). Protocols were benchmarked with low-resolution images (640 × 480 pixels) and high-resolution images (4096 × 3072 pixels). In addition, Protocol 2 was evaluated with five different tile sizes. For the low-res images we tested the following tile sizes in pixels (T1: 5 × 5; T2: 10 × 10; T3: 20 × 20; T4: 40 × 40; and T5: 160 × 160), and for the high-res images (T1: 32 × 32; T2: 64 × 64; T3: 125 × 125; T4: 256 × 256; and T5: 1024 × 1024).

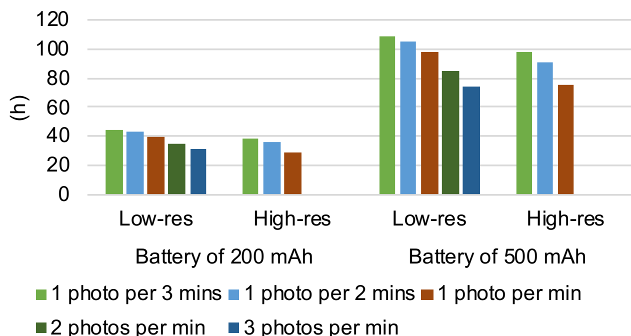

Fig. 6. Estimated camera operational time when running our protocols with various photo capturing frequencies and powered with a battery of 200 mAh and 500 mAh, respectively. For Protocol 2 we used medium tile sizes (i.e., T3).

quad-core processor. Specifically, we evaluated only the storage protocol for securely linking captured images (Protocol 1) since the runtime of the other parts, i.e., verifying image authenticity, platform attestation, and the zero-knowledge protocol for checking modified images (Protocol 2), is negligible (i.e., < 3 seconds) compared to the link protocol. We ran the verification procedure from section VI-B and observed how long it took to verify file-chains of different lengths. We could verify a stream of 2,880 images (one day worth of images captured in 30 second intervals) in about 22 seconds; 20,000 images (approx. one week of images) in about 150 seconds; and 86,000 images (approx. one month) in about 670 seconds.

Ultimately, our solution may be ported to the smaller "Raspberry Pi Zero W" (which is about 2.5 times smaller than the Pi 3 B+) in order to make it more portable and manageable for real-life applications. While the slower Pi Zero will run our solution less efficiently, our results will easily match the performance of any next-generation Pi Zero boards.

IX. Limitations

Using a TPM-based camera as a root of trust has its own security caveats, as discussed in [25]. With physical access to the camera, an attacker may modify its picture-taking sensor, thus feeding the camera's software with already modified image data. Possible remedies include modifying the sensor to provide encrypted image data to the rest of the camera components (proposed also by [25]), or authenticating the sensor using pattern noise data that may serve as a unique identifier [40]. All this would still be unable to prevent *staging attacks*. A malicious user simply enacts a fake version of an experience, more or less recording a "movie" with look-alike actors. By "borrowing" the target user's camera for a day (e.g., swapping it with a fake device to hide the fact) the attacker can simply record the fake experiences with the target user's own camera device. Other low-tech alternatives are holding up images in front of the target's camera device.

X. Conclusion and Future Work

As pervasive data capturing becomes a reality, our daily experiences may soon be easily and continuously captured by wearable capture systems and infrastructure devices. By automatically creating "memory cues" from these captured experiences, which are then played back to us in an ambient fashion (e.g., on our lock screens), we could train our memory in order to better remember important events and experiences.

The idea of building such a memory augmentation system entails high risk, however, as it makes our memories vulnerable to attacks: malicious parties able to access our captured experiences could surreptitiously change our memories by inserting fictitious experiences, modify existing experiences, or simply deleting recorded experiences. The secure camera presented here addresses two of the most important direct memory manipulation attacks: deleting stored memory cues and sharing manipulated captured images. Our results show that our scheme is practical even for low-powered devices.

A natural direction for forthcoming research is to account for future attacks on the underlying cryptographic primitives (i.e., hash functions and key lengths), as well as to allow for a seamless transition to a different camera after a lost or damaged hardware.

ACKNOWLEDGEMENT

The authors acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the 7th Framework Programme for Research of the European Commission, under FET Grant Number: 612933 (RECALL).

REFERENCES

[1] J. Gemmell, G. Bell, and R. Lueder, "MyLifeBits: A Personal Database for Everything," *Commun. ACM*, vol. 49, no. 1, pp. 88–95, Jan. 2006.

[2] C. Gurrin, A. F. Smeaton, and A. R. Doherty, "LifeLogging: Personal Big Data," *Found. Trends Inf. Retr.*, vol. 8, no. 1, pp. 1–125, Jun. 2014.

[3] M. Harvey, M. Langheinrich, and G. Ward, "Remembering through lifelogging: A survey of human memory augmentation," *Pervasive and Mobile Computing*, vol. 27, pp. 14–26, 2016.

[4] T. Dingler, P. E. Agroudy, H. V. Le, A. Schmidt, E. Niforatos, A. Bexheti, and M. Langheinrich, "Multimedia Memory Cues for Augmenting Human Memory," *IEEE MultiMedia*, vol. 23, no. 2, pp. 4–11, Apr. 2016.

[5] A. Bexheti, E. Niforatos, S. A. Bahrainian, M. Langheinrich, and F. Crestani, "Measuring the Effect of Cued Recall on Work Meetings," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, ser. UbiComp '16. New York, NY, USA: ACM, 2016, pp. 1020–1026.

[6] N. Davies, A. Friday, S. Clinch, C. Sas, M. Langheinrich, G. Ward, and A. Schmidt, "Security and Privacy Implications of Pervasive Memory Augmentation," *IEEE Pervasive Computing*, vol. 14, no. 1, pp. 44–53, Jan. 2015.

[7] E. Niforatos, M. Laporte, A. Bexheti, and M. Langheinrich, "Augmenting Memory Recall in Work Meetings: Establishing a Quantifiable Baseline," in *Proceedings of the 9th Augmented Human International Conference*, ser. AH '18. New York, NY, USA: ACM, 2018, pp. 4:1–4:7.

[8] K. Wolf, A. Schmidt, A. Bexheti, and M. Langheinrich, "Lifelogging: You're Wearing a Camera?" *IEEE Pervasive Computing*, vol. 13, no. 3, pp. 8–12, Jul. 2014.

[9] A. Bexheti, M. Langheinrich, and S. Clinch, "Secure Personal Memory-Sharing with Co-located People and Places," in *Proceedings of the 6th International Conference on the Internet of Things*, ser. IoT'16. New York, NY, USA: ACM, 2016, pp. 73–81.

[10] S. Clinch, P. Metzger, and N. Davies, "Lifelogging for 'Observer' View Memories: An Infrastructure Approach," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, ser. UbiComp'14 Adjunct. New York, NY, USA: ACM, 2014, pp. 1397–1404.

[11] D. Byrne, A. Kelliher, and G. J. Jones, "Life Editing: Third-party Perspectives on Lifelog Content," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '11. New York, NY, USA: ACM, 2011, pp. 1501–1510.

[12] N. Davies, N. Taft, M. Satyanarayanan, S. Clinch, and B. Amos, "Privacy Mediators: Helping IoT Cross the Chasm," in *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '16. New York, NY, USA: ACM, 2016, pp. 39–44.

[13] D. J. Shaw, *The Memory Illusion: Remembering, Forgetting, and the Science of False Memory*. Random House, Jun. 2016.

[14] L. A. Henkel, "Photograph-induced memory errors: When photographs make people claim they have done things they have not," *Applied Cognitive Psychology*, vol. 25, no. 1, pp. 78–86, Jan. 2011.

[15] A. S. Brown and E. J. Marsh, "Evoking false beliefs about autobiographical experience," *Psychonomic Bulletin & Review*, vol. 15, no. 1, pp. 186–190, Feb. 2008.

[16] K. A. Wade, M. Garry, J. D. Read, and D. S. Lindsay, "A picture is worth a thousand lies: Using false photographs to create false childhood memories," *Psychonomic Bulletin & Review*, vol. 9, no. 3, pp. 597–603, Sep. 2002.

[17] D. S. Lindsay, L. Hagen, J. D. Read, K. A. Wade, and M. Garry, "True Photographs and False Memories," *Psychological Science*, vol. 15, no. 3, pp. 149–154, Mar. 2004.

[18] K. Bicakci and N. Baykal, "Infinite length hash chains and their applications," in *Proceedings. Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2002, pp. 57–61.

[19] Y.-C. Hu, M. Jakobsson, and A. Perrig, "Efficient Constructions for One-Way Hash Chains," in *Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science, J. Ioannidis, A. Keromytis, and M. Yung, Eds. Springer Berlin Heidelberg, 2005, pp. 423–441.

[20] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "The TESLA Broadcast Authentication Protocol," p. 11, 2002.

[21] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "SPINS: Security Protocols for Sensor Networks," *Wireless Networks*, vol. 8, no. 5, pp. 521–534, Sep. 2002.

[22] D. Bayer, S. Haber, and W. S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," *Sequences II: Methods in Communication, Security and Computer Science*, pp. 329–334, 1993.

[23] S. Haber and W. S. Stornetta, "How to Time-Stamp a Digital Document," in *Advances in Cryptology-CRYPT0' 90*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Aug. 1990, pp. 437–455.

[24] P. Maniatis, T. J. Giuli, and M. Baker, "Enabling the Long-Term Archival of Signed Documents through Time Stamping," *arXiv:cs/0106058*, Jun. 2001.

[25] A. Naveh and E. Tromer, "PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 255–271.

[26] A. Chiesa and E. Tromer, "Proof-Carrying Data and Hearsay Arguments from Signature Cards," in *Proceedings of the 1st Symposium on Innovations in Computer Science*, 2010, pp. 310–331.

[27] H. Chabanne, R. Hugel, and J. Keuffer, "Verifiable Document Redacting," in *Computer Security – ESORICS 2017*, S. N. Foley, D. Gollmann, and E. Snekkenes, Eds. Cham: Springer International Publishing, 2017, vol. 10492, pp. 334–351.

[28] R. Steinfeld, L. Bull, and Y. Zheng, "Content Extraction Signatures," in *Information Security and Cryptology – ICISC 2001*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Dec. 2001, pp. 285–304.

[29] C. Bettini and D. Riboni, "Privacy protection in pervasive systems: State of the art and technical challenges," *Pervasive and Mobile Computing*, Oct. 2014.

[30] S. Saroiu and A. Wolman, "I Am a Sensor, and I Approve This Message," in *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, ser. HotMobile '10. New York, NY, USA: ACM, 2010, pp. 37–42.

[31] T. Winkler, A. Erdélyi, and B. Rinner, "TrustEYE.M4: Protecting the sensor – Not the camera," in *2014 11th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, Aug. 2014, pp. 159–164.

[32] T. Winkler and B. Rinner, "TrustCAM: Security and Privacy-Protection for an Embedded Smart Camera Based on Trusted Computing," in *2010 7th IEEE International Conference on Advanced Video and Signal Based Surveillance*, Aug. 2010, pp. 593–600.

[33] B. Parno, J. M. McCune, and A. Perrig, *Bootstrapping Trust in Modern Computers*, ser. SpringerBriefs in Computer Science. New York, NY: Springer New York, 2011, vol. 10.

[34] A. Bexheti, A. Fedosov, I. Elhart, and M. Langheinrich, "Memstone: A Tangible Interface for Controlling Capture and Sharing of Personal Memories," in *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services*, ser. MobileHCI '18. New York, NY, USA: ACM, 2018, pp. 20:1–20:13.

[35] K. Dietrich, M. Pirker, T. Vejda, R. Toegl, T. Winkler, and P. Lipp, "A Practical Approach for Establishing Trust Relationships between Remote Platforms Using Trusted Computing," in *Trustworthy Global Computing*, G. Barthe and C. Fournet, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, vol. 4912, pp. 156–168.

[36] M. Bellare, R. Canetti, and H. Krawczyk, "Keying Hash Functions for Message Authentication," in *Advances in Cryptology – CRYPTO'96*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Aug. 1996, pp. 1–15.

[37] T. Hardjono and G. Kazmierczak, "Overview of the TPM Key Management Standard," p. 15, 2008.

[38] P. Oechslin, "Making a Faster Cryptanalytic Time-Memory Trade-Off," in *Advances in Cryptology - CRYPTO 2003*, G. Goos, J. Hartmanis, J. van Leeuwen, and D. Boneh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, vol. 2729, pp. 617–630.

[39] Gough, Lui, "Review, Teardown: Keweisi KWS-V20 USB Tester," http://goughlui.com/2016/08/20/review-teardown-keweisi-kws-v20-usb-tester/, Aug. 2016.

[40] J. Lukas, J. Fridrich, and M. Goljan, "Digital camera identification from sensor pattern noise," *IEEE Transactions on Information Forensics and Security*, vol. 1, no. 2, pp. 205–214, Jun. 2006.