# Secure Personal Memory-Sharing with Co-located People and Places

**DOI:**
10.1145/2991561.2991577

**Document Version**
Final published version

**Citation for published version (APA):**
Bexheti, A., Langheinrich, M., & Clinch, S. (2016). Secure Personal Memory-Sharing with Co-located People and Places. In *IoT2016 - 6th International Conference on the Internet of Things* (pp. 73-81). Association for Computing Machinery. https://doi.org/10.1145/2991561.2991577

**Published in:**
IoT2016 - 6th International Conference on the Internet of Things

# Secure Personal Memory-Sharing with Co-located People and Places

**Agon Bexheti**
Università della Svizzera italiana
Lugano, Switzerland
agon.bexheti@usi.ch

**Marc Langheinrich**
Università della Svizzera italiana
Lugano, Switzerland
marc.langheinrich@usi.ch

**Sarah Clinch**
The University of Manchester
Manchester, United Kingdom
sarah.clinch@manchester.ac.uk

## ABSTRACT

The abundance of interconnected devices in the Internet of Things (IoT) offers a powerful vision on how automated capture systems can aid humans remember their lives better. Already today, mobile and wearable devices allow people to create rich logs of their daily experiences in the form of photos, videos, GPS traces, or even physiological data. This activity is often called "lifelogging", and has led to the so-called "quantified self" movement where people capture detailed traces of their everyday lives in order to better understand themselves. An interesting avenue to explore in this context is the possibility of capturing lifelog data for the sake of augmenting one's memory. Contemporary psychology theory suggests that captured experiences of daily events can be used to generate cues (hints) which, when reviewed, can improve subsequent long-term recall of these memories. However, limitations of on-body placement of wearable devices can yield poor quality data and restricts capture to a first-person perspective. The focus of this work is to enable the secure and automatic exchange of one's lifelog streams with both co-located peers and any available capture devices in an IoT infrastructure, in order to construct a more comprehensive representation of a previous experience, which can thus help one to create more effective cues. We present a privacy-aware architecture for this exchange and report on a proof-of-concept prototype implementation.

### CCS Concepts

•**Security and privacy** → **Access control; Privacy-preserving protocols; Authorization;** *Authentication;* •**Human-centered computing** → *Ubiquitous and mobile computing;*

### Author Keywords

Lifelogging; Memory augmentation; Sharing; Privacy; Security; Co-located people; Wearables; Infrastructure camera.

*IoT'16,* November 07-09, 2016, Stuttgart, Germany
© 2016 ACM. ISBN 978-1-4503-4814-0/16/11…$15.00
DOI: http://dx.doi.org/10.1145/2991561.2991577

## INTRODUCTION

The emerging Internet of Things (IoT) not only allows us and our devices to be "connected" all the time, but also to capture and visualize an increasing share of our everyday lives in a digital format – a process known as "lifelogging". The number of Internet-connected devices has seen a massive growth in the last 13 years (going from 500 million in 2003, to 12.5 billion in 2010, to 50 billion by 2020)[1], clearly making the number of IoT devices per person more than 5. This coupled with techno-advances in data mining techniques has the potential to bring lifelogging to a next level. When lifelogging pioneer Gordon Bell started his MyLifeBits project [8] in 2001, he initially envisioned capturing mostly archival material (e.g., computer files, scanned books and digitized music), similar to a modern "Memex" [1]. Advances in capture technology, however, quickly allowed him to move from capturing *legacy content* to *real-time and continuous content streams*: "phone calls, meetings, room conversations" as well as "1-2 thousand photos that [his] SenseCam" captured every day.

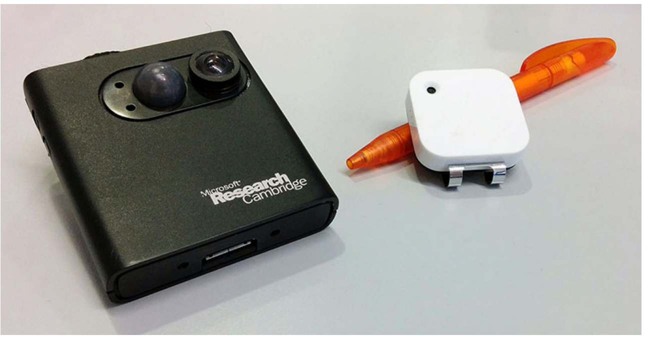
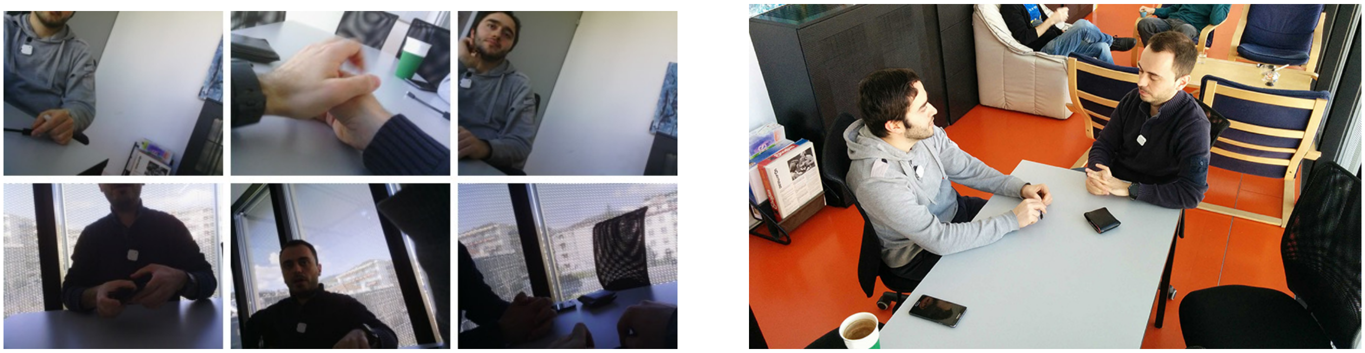


Figure 1: Left: Microsoft SenseCam; Right: Narrative Clip 1.

The SenseCam [9] developed in 2004, was arguably the first "lifelogging camera": a neck-worn front-facing device that captured images periodically (e.g., every 30 seconds) or when the scene changed significantly (such as when moving to a different room). Modern lifelogging cameras have extended the functionality of the SenseCam and followed the IoT paradigm while offering an increasingly compact format.

For example, the 2016 Narrative Clip 2[2] weighs less than 20g (the SenseCam weighed several hundred), includes WiFi connectivity (SenseCam needed a cable), features 30h of battery

[1] Figures according to Cisco IBSG 2011.
[2] See www.getnarrative.com.


(a)

(b)

Figure 2: A coffee-break discussion as captured by (a) the participants' Narrative Clip devices, and (b) a fixed camera.

life (the SenseCam lasted 12h), records 8MP images with 3264x2448 pixels (SenseCam did 640x480), and can record video (which the SenseCam did not). Figure 1 shows a Narrative Clip 1 next to a 2004 SenseCam. Photos and videos are not the only types of lifelogging data that can be captured; modern smartphones and wearables can contain a wealth of sensors to capture, e.g., GPS location, acceleration, light levels, temperature, audio, and a range of physiological parameters, such as heart rate or galvanic skin response.

This work builds on our prior work on *memory augmentation systems* [5], which suggested that, instead of building a memex-like "memory prosthetic" to be queried whenever one's own memory fails, one might instead use captured data to *train* human memory, so that the relevant information can actually be recalled without any external tool whenever it is needed. By using the captured data to build carefully selected *memory cues*[3] and replaying those to users in an ambient fashion (e.g., as a screensaver or a smartwatch card), the memories associated with these cues can be more easily retrieved (i.e., at will) at a later time.

When looking at images captured by SenseCam-like devices, however, we have previously found [4] that in many circumstances, the captured first-person perspective may not lend itself well as a memory cue. In a multi-day experiment, we found that camera lenses were often covered by clothes or hair, or simply faced the wrong way due to the way they were affixed to the body. Even when unobstructed, the first-person-view typically showed only a small part of the scene, potentially never capturing a person sat right next to the person recording (see Figure 2a for examples from our own experience). Fixed infrastructure cameras can compensate for such failures: a high vantage point usually allows them to capture comprehensive scenes, completely unobstructed (see Figure 2b). Furthermore, first-person views from the devices of *others* may also offer a richer view than one's own [2]. In addition, we conducted an experiment [3] where we collected lifelog traces of a group of twenty researchers, and identified the necessity for an automatic and seamless way of exchanging self-captured data in order to create a comprehensive lifelogging dataset.

In an IoT network of infrastructure sensors (e.g., fixed cameras) capable of capturing people interactions and users equipped with personal lifelogging devices, our work investigates a technique both for seamless acquisition of self-captured lifelogging data of others with whom we interact, and data from local infrastructure devices. Note that while this work focuses on the exchange of captured photos, it could work equally well with other media and data types, and hence is able to support any of the above-mentioned lifelog data, as long as they are packaged into discrete files that can be exchanged between peers. In this paper, we elicit initial system requirements and provide an architecture for sharing personal lifelog data with others, with a view to preserving the privacy of all involved users, as well as ensuring confidentiality, authenticity, integrity, and provenance for shared memories. With a focus on the technical side of this work we report a prototypical implementation of this architecture and present initial system performance measurements. Note that we do not report on the impact of this architecture on augmenting users' memory or on any other usability related study as these are out of scope of this paper.

## REQUIREMENTS

To gain an understanding of system requirements, we consider the following scenario:

- **Morning walk to the office:**
  *We want access to any fixed infrastructure camera capturing our walk, yet must also prevent their owners from easily tracking our location.*
- **Enter department building, meet a colleague in the hallway for a chat:**
  *We want access to our colleague's captured data, and data from any hallway cameras.*
- **Attend a work meeting:**
  *We want in-room sensors to provide access to high-quality captured data (camera, microphone, board contents, etc.), as well as captured data from co-located colleagues. People*

[3] A memory cue is an item, e.g., a photograph, but also a sound or a smell, that can act as a hint to a past experience.

*who simply pass in front of the meeting room should not have access to this data.*

- **After the meeting, while packing our bag in the room, chat with a colleague:**
  *Although high-quality capture of the meeting room is stopped, we still want to capture data from our colleague's wearable camera. Colleagues who have already left the room should not have access to this data.*

Ultimately, our goal is to build a system that would enable the seamless and secure sharing of captured lifelog data in such a fully connected IoT scenario. However, at the outset, this work focuses on the implementation of peer-based data exchange and less on acquiring data captured by fixed infrastructure sensors (e.g., fixed cameras).

## ENVISIONED FUNCTIONALITY

Figure 3 gives an overview of foreseen data flows. The envisioned core functionality is as follows:

- *Advertise Sharing Capabilities:* A capture device that is willing to share lifelog data with others needs to indicate its availability to close-by peers, yet without making it trivial to passively track it this way.
- *Detect Peers:* Once proximate peer devices are detected, the device needs to negotiate the secure exchange of captured data for the duration of co-location. Peer devices may include both the personal capture devices of other users and fixed capture installations (e.g., room cameras).
- *Secure Exchange of Capture Data:* Actual data exchange need not (necessarily) happen in real-time. Instead, a device must communicate to co-located peers how they can securely access the images that the device has captured.
- *Detect End of Co-Location:* If a peer leaves, the device needs to cease sharing its captured data with this peer, though it may continue to do so with peers that continue to be co-located. Once all peers have left, the device needs to stop sending out access information regarding the captured data.
- *Support Time-Limited Exchange and/or Access Revocation:* The system should support time-based access control of shared data, e.g. a peer must download the shared data within a week.

## ARCHITECTURE

This section describes the full life cycle of a shared capture session – from peer discovery to data exchange and access control. Table 1 itemizes identifiers used in this section. In a nutshell, a user's device advertises its willingness to both share its self-captured data and to acquire data of other co-located peer devices by broadcasting periodically updated access tokens and its temporal public key (section *"Advertise sharing availability using memory beacons"*). Tokens represent a way of letting others know where to access data one is willing to share. Both tokens and the public key are sent out only when the device senses the presence of a co-located peer in order to prevent a malicious person from tracking its location by passively sniffing announced data (see section *"Smart memory beaconing based on peer detection"* for details). When another peer observers such announcements, it will trigger a passive key-exchange with the peer (see section *"Key exchange protocol"*). To perform such a P2P key exchange in a multi-user environment, all peers broadcast both access tokens and their public keys. A device grants access to its self-captured data that it is willing to share only to co-located peers. It does so by encrypting such shared data with all peers' public keys. As a consequence, only the peers who possess the valid private key will be able to get the shared data (see section *"Access control to shared data"*).

We will now describe each of these steps in more detail.

| Variable | Description |
|---|---|
| $K_{hash}$ | Key for hashing tokens to storage location. |
| $Pub_{ij}$ / $Priv_{ij}$ | Temporary session key-pair; $i$ identifies device, $j$ identifies session. |
| $token_{it}$ | Access token, where $i$ is the device that issued it and $t$ is the time period it was issued in. |
| $t_{dwell}$ | Minimum dwell time before adding or removing a peer $Pub_{ij}$ key to the list of registered peers. |

Table 1: List of variables used in this work.

### Advertise sharing availability using memory beacons

A device advertises its capability (and willingness) to let others access its captured data by simply announcing an online location (URL) of where the data will eventually be located (again, real-time upload of captured data is not a key requirement). This also means that a device does not exchange the actual data itself over the wireless channel, which minimizes both bandwidth and energy consumption.

The actual announcements (*memory beacons*) are send using the Bluetooth Low Energy (BLE) short-range wireless protocol.

The advertised online location is not fixed, but is based on an implied fixed system-wide base URL (not send). The beacon therefore only provides a continuously changing *access token*, under which peers can find captured data for the short period over which this token was used. Our current prototype updates tokens every few seconds. Access tokens are simply long random numbers – large enough so that accidental overlap of tokens becomes unlikely. Due to the limited size available in BLE announcements (see section *"Implementation"* below) we use 200 bits in our prototype, which does not scale well in the long term[4] but is probably OK for immediate deployments. Note that if a client notices that a particular token is already "occupied" during upload, it could simply drop it or overwrite the data at that address with its own encrypted URL (both

[4] A few billion people, each sending 86′400 individual tokens per day (one new token per second), would create $8.6410^{13} \approx 10^{14}$ tokens per day. If we want these to be around for some 3 years (1000 days), we have $10^{17}$ tokens "in use" at any time. Given the birthday paradox, this would leave a $10^{17}/2^{200/2} \approx 10^{17}/10^{30} \approx 10^{-13}$ chance of overlap.

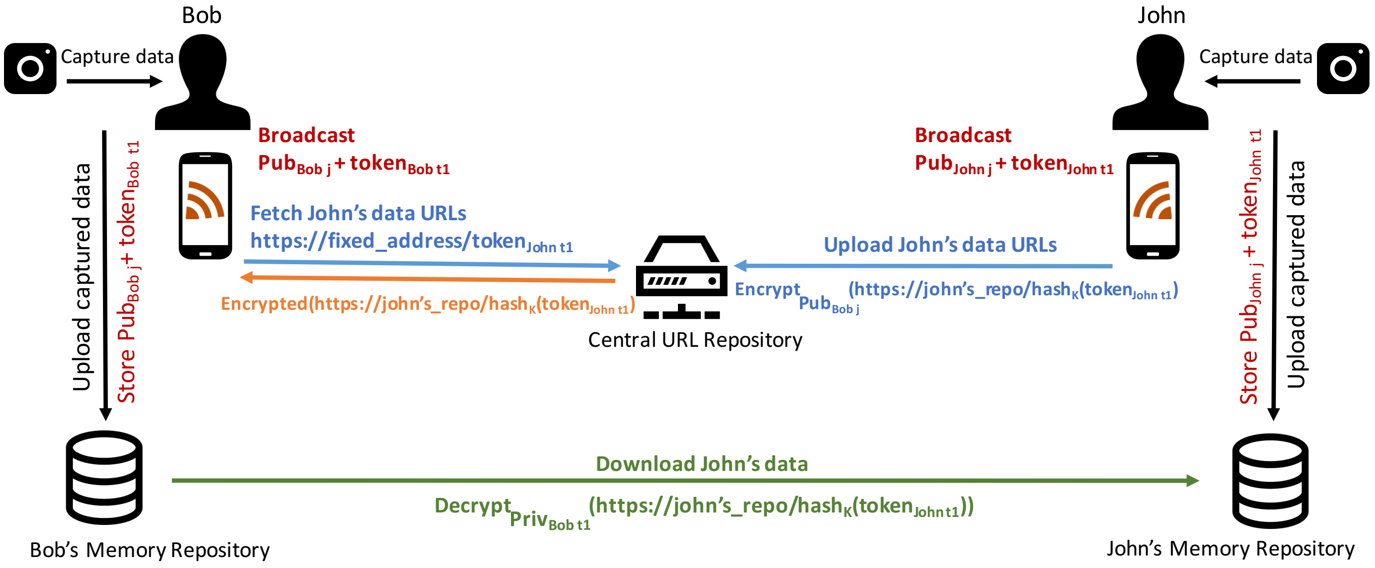

Bob
Capture data
Broadcast
Pub Bob j + token Bob t1
Upload captured data
Store Pub Bob j + token Bob t1
Fetch John's data URLs
https://fixed_address/token John t1
Encrypted(https://john's_repo/hash K(token John t1)
Central URL Repository
Upload John's data URLs
Encrypt Pub Bob j (https://john's_repo/hash K(token John t1)
John
Capture data
Broadcast
Pub John j + token John t1
Store Pub John j + token John t1
Upload captured data
Download John's data
Decrypt Priv Bob t1 (https://john's_repo/hash K(token John t1))
Bob's Memory Repository
John's Memory Repository

Figure 3: Data flow for capturing and sharing lifelog data between two co-located people. Bob's repository downloads John's shared capture data, using decrypted URLs available from a central URL repository.

options result in a single lost second of data, either the current period or the previous occupying period).

Peers use captured tokens to find the shared data at the known system-wide URL. In the following we denote tokens as $token_{it}$, where $i$ is the issuing device and $t$ is the time period for which the token was issued. Their short lifetime means that as soon as a peer leaves the range of the device's beacons, they are unable to access any of the data captured at a later time, as this data will use different access tokens.

While tokens thus provide some access control, as clients need the token to know the address, one might accidentally "stumble" upon uploaded data simply by trying arbitrary tokens. To prevent this, peers also need an access key to decrypt the data present at the token address. As part of the announcements, devices thus also periodically send out public key $Pub_{ij}$, where $i$ again identifies the device and $j$ identifies a "sharing session". Whenever a device stops sharing captured data, i.e., when all peers leave, it starts a new session upon first discovering a new peer. Hence, over the course of a day, several public keys will be generated and used. In our prototype, every $10^{th}$ announcement carries a public key instead of an access token. We describe the use of these public keys to access shared data in section 4.4.

### Smart memory beaconing based on peer detection

Broadcasting a continuous stream of tokens allows a passive attacker to track a device. While tokens change frequently, public keys (which are interleaved with tokens) may not change that often. With few devices around, there may simply not be enough "noise" for a device to "blend-in" with others.

For this reason, devices periodically stop beaconing, moving from *sharing mode* into an *idle mode*. Mode transition is triggered by the the absence of other proximate peers; the device will switch from idle mode to sharing mode if it detects the presence of a peer in range, and vice versa (see Figure 4). Presence detection cannot therefore rely on the reception of tokens from other peers, as these may be in idle mode. Instead, we propose use of a *social detector* – a sensor designed to detect social engagement with other peers. Such a sensor may be audio based (detecting if a device owner is engaged in conversation [14] or counting the number of speakers [18]) or visual (using image recognition to detect faces in captured photographs – a face lingering in front of the camera may well indicate social interaction [7]).

Note that infrastructure sensors are basically peers who do not move. No privacy issues prevent them from simply always broadcasting beacon announcements.

### Key exchange protocol

Upon encountering a memory beacon from another device, or upon detecting social engagement, a device will trigger beacon announcement. As part of this process, it will run a passive key exchange protocol with other co-located devices to exchange public keys for the session ($Pub_{ij}$). Since announcements are broadcast, the exchange is passive in the sense that there is no direct communication or a handshake between the devices whatsoever. Once a peer's public key is received, a device subsequently encrypts uploaded data using that public key. If multiple peers are detected, a broadcast encryption scheme [6] is used to encrypt the data using each peer's public key (see Section 4.4 below).

In order to prevent passers-by from receiving a set of access tokens "by accident" and thus having access to captured data, we enforce a *dwell time* ($t_{dwell}$) during which the peer's public key must be seen repeatedly. Only if a peer has been present for long enough, will we retroactively mark all captured data staring from its first detection as being also shared with this particular peer. As long as the peer stays in range (i.e., as long

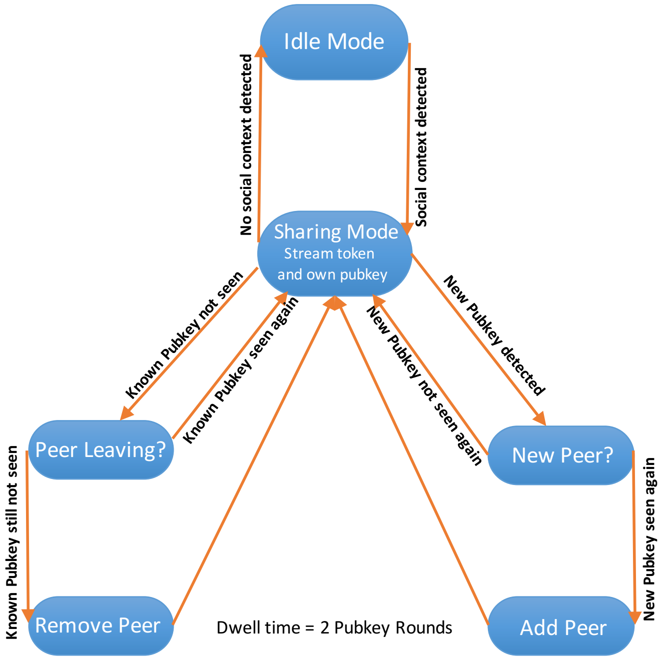


Figure 4: System state diagram showing the transition between Idle Mode and Sharing Mode and the peer presence detection, using an example dwell time $t_{dwell}$ of two rounds.

as its public key is periodically received), the device will add the peer's $Pub_{ij}$ key to the list of authorized clients that can access its captured data (see Figure 4).

### Access control to shared data

A device controls access to its captured data using both the access tokens sent during "sharing mode", and the public keys received from other peers. Co-located peers can then query that device's database at a *known base address* and exchange tokens they have collected for actual data:

$$\longrightarrow https://fixed\_address/token_{it}$$

In order to facilitate the use of one's own repository to host actual captured data, the known base address does not directly provide data but instead offers a "redirect" to the data host. In order to prevent trivial tracking through resolution of collected tokens, the URL redirect is encrypted with the public keys of all authorized peers:

$$\longleftarrow E(https://user\_repo/data\_address)$$

A peer accessing the token URL thus receives an encrypted value that, when decrypted using the peer's private session key at the time (which the peer will need to keep track of), will result in another URL for access to actual captured data.

To achieve a multi-user encryption setup, we use *broadcast encryption* [6]: instead of re-encrypting the data URL for each peer, we use a (random) *symmetric key* to encrypt the URL with a symmetric cipher, and then encrypt that symmetric key with each recipients $Pub_{ij}$ key. While URLs are not long, encrypting only a 128-bit symmetric key instead of a 30-40 byte long URL is clearly more economical.

Given the above, captured data can then be made available at a repository of a user's choice and does not have to be encrypted, as only those who both know the token $token_{it}$ and have had their session public key $Pub_{ij}$ captured by the device (and subsequently used to encrypt the used symmetric key) will be able to find URLs of shared images.

In order to come up with the final URL on the user's repository, the user can simply use a keyed hash, together with a long-term secret key $K_{hash}$, to convert each access token into a storage URL on their own repository. A non-keyed hash function would not be sufficient, as it would allow an attacker who knows the base address of the user's repository (e.g., from a previous peer exchange) to simply "fish" for images using only captured access tokens.

A device can revoke access to (unretrieved) data at any time. To invalidate access for a peer, the device simply removes the public key $Pub_{ij}$ of the peer from the set of authorized devices and re-runs the broadcast encryption scheme to re-encrypt URLs with a new key. This, of course, only makes sense if the peer has not already downloaded the shared data.

### ENVISIONED THREATS

One of the goals of the proposed architecture is to protect the *captured memories* of a user from unauthorized access as well as prevent tracking the *user's location*. As a result, we primarily try to minimize "oversharing", i.e., the accidental tuning-in of a peer's device to our captured data stream, without actually interacting with us, as well as enlarging our tracking envelope.

Tracking is a prominent risk in our system since a device advertises its willingness to share captured data by sending out an announcement over a non-secure channel. An attacker could simply listen passively for such advertised information and thereby track a user's location. To counter this threat, a device does not send out announcements all the time. Instead, it announces itself only when it detects the *presence* of an appropriate peer. Also, announcements should not be linkable.

The system should avoid oversharing (i.e., allowing non-authorized parties to access captured data) by granting access only to peers who were present and engaged with the user at the time of capture. Figure 5 depicts the different ways an unauthorized person can try to get access to a user's captured data. An attacker can try to construct a data URL (one that leads to actual data of a user) by guessing a valid access token or passively sniffing for tokens being sent out. Either way, the attacker then has to also guess a valid decryption key (i.e. the user's private ECC key) in order to successfully get a valid data URL. To some extent, this sort of attack is similar to [10] where they show that most of the existing URL shorteners (which are mainly used for data exchange) have serious security consequences due to their low token space (typically 5-6 characters). In our case, guessing a randomly generated access token of 25 bytes (200 bits) and a valid decryption key (the private part of a 256 bits ECC key-pair; token and key generation are explained in the following section) is extremely hard to happen. Even if an attacker manages that, this would only allow her to get a small set of data points and not compromise the whole system, as other data are protected by different

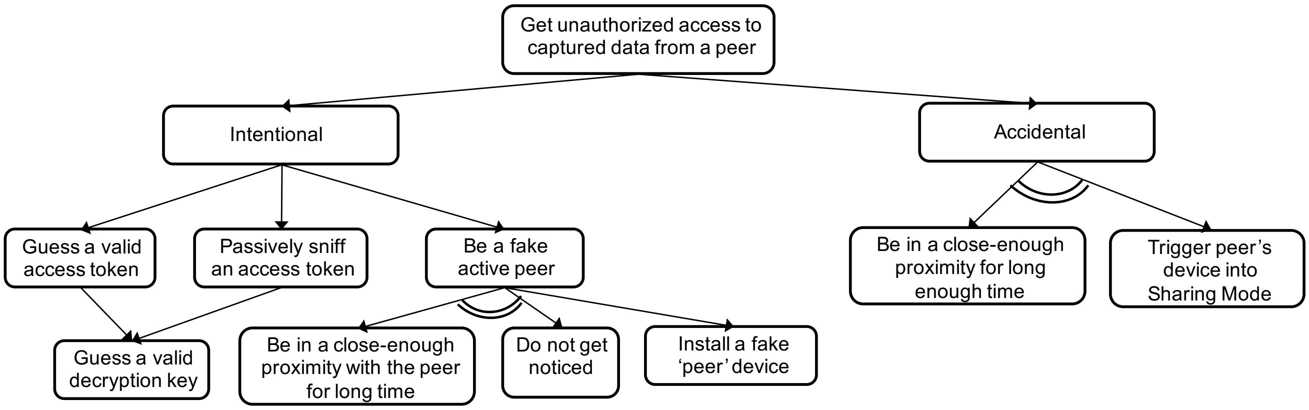

Get unauthorized access to captured data from a peer
Intentional
Accidental
Guess a valid access token
Passively sniff an access token
Be a fake active peer
Be in a close-enough proximity for long enough time
Trigger peer's device into Sharing Mode
Guess a valid decryption key
Be in a close-enough proximity with the peer for long time
Do not get noticed
Install a fake 'peer' device

Figure 5: Attack tree visualizing the different ways for obtaining unauthorized access to a peer's captured data. Tree nodes represent sub-actions that must be accomplished for the attack to succeed. Double half-arcs represent an AND relationship among nodes indicating that all such node actions have to be achieved in order for the parent action to succeed.

access tokens and eventually different encryption keys. Oversharing might also happen due to accidental data spills, e.g., by erroneously identifying passers-by as peers. All in all, the shared data should only cover co-located periods.

Note that we do not guard against secret recordings – either out-of-band (e.g., a hidden camera or microphone) or through a "hidden" (fake) peer device that exploits the system by pretending to be a peer capture device, but in reality is simply passing on captured data to an attacker. Countering hidden cameras or microphones is beyond the scope of this work. However, to counter fake peers, one could imagine equipping capture devices with low-powered OLED displays to show the number of connected peers, coupled with a short audible beep if a new peer "connects". This would, e.g., allow meeting participants to detect data over-sharing through a quick head count mismatch.

We also do not consider control for re-sharing, i.e., preventing peers from passing data on to others. However, one could consider watermarking to help identify the source of a leak should it, for example, surface on the Internet.

## IMPLEMENTATION

We have implemented a prototype smartphone app to enable the secure sharing of pictures captured by a worn camera such as the Narrative Clip. As the current generation of the Narrative Clip does not provide programmatic access, the following procedure must currently be followed for achieving our envisioned functionality:

- A user wears a lifelogging camera and also carries a smartphone running our app.
- The camera captures the user's activities while the phone exchanges keys and tokens with other peer devices.
- Later, the user manually connects the wearable camera and uploads captured images to their repository. The app wirelessly syncs with the repository, uploading a set of timestamps and distributed tokens, as well as any captured public keys from peers.
- A process in the repository then binds each image to distributed tokens $token_{it}$ based on timestamps. Each image is made available at a URL based on the keyed hash of the token used to advertise it.
- Finally, the repository process generates a set of encrypted URLs for each image URL, based on received $Pub_{ij}$ keys from other peers, and uploads these to a central repository under the respective tokens.

Clearly, a next-generation wearable camera could integrate all of these steps into a single worn device, handling, e.g., all upload and processing during its charging at night.

### Smartphone based beacon App

Our prototype app consists of two main services: a *beacon transmitter* and a *beacon scanner*. The transmitter service generates a public/private key-pair every time it starts and then broadcasts the public key; both keys are generated with Elliptic Curve Cryptography (ECC) using a curve over a 256 bit prime field. The transmitter also generates and broadcasts a new random access token at a specific frequency. Tokens are generated using a hash based pseudo-random number generator. The scanner service listens for nearby beacons of the same type. Once it detects a beacon, it will store both access tokens and captured public keys from other peers in internal storage.

The app uses Bluetooth Low Energy (BLE) as the underlying technology for sending and receiving both keys and access tokens. We use the Android Beacon Library[5] to create a BLE beacon-like device. While most phones can scan for nearby BLE beacon devices, there are currently few smartphones that can also act as a beacon transmitter. For this prototype we use the Nexus 5X, which does support this functionality.

The Bluetooth specification enables data broadcast through its *advertisement packet*. BLE version 4 allows up to 28 bytes (excluding the BLE header bytes) for manufacturer specific data in the advertisement packet. Three bytes are consumed

[5] see https://altbeacon.github.io/android-beacon-library/

by the beacon library (2 for specifying a mandatory beacon identifier, and one for a mandatory power reference value[6]) leaving 25 bytes for actual data (see Fig 6 for the beacon layout). This size limnitation restricts our choice of key and token length. As a result, the app generates tokens $token_{it}$ of exactly 25 bytes (200 bits). Since 200 bits is not sufficient for public key transmission $Pub_{ij}$, we instead broadcast a 25 byte identifier for a key uploaded to a known *key server*; peers can use the identifier to retrieve the key from this known server.

| BLE Header<br>(10 bytes) | Beacon Header<br>(2 bytes) | TX Power Reference<br>(1 byte) | Actual Data<br>(25 bytes) |
|---|---|---|---|

Figure 6: The beacon protocol data unit.

The app alternates between sending its session public key $Pub_{ij}$ and the changing access tokens $token_{it}$. We use the 2 reserved bytes of the beacon header (see Figure 6) to differentiate between key and token data. Since tokens are refreshed much more frequently than public keys, the app is configured to broadcast token beacons with a higher frequency than keys. However, a too low frequency of key distribution will result in delayed "registration" of peers. We experimented with different transmission schedules in order to find the best ratio between public key and token transmission frequencies and report results in the following section.

The app features an energy-efficient social sensor for detecting the presence of other co-located people with whom a user might be engaged. It is a modified version of the speaker count algorithm as described in [18] which uses microphone audio analysis to count the number of different people engaged in a discussion.

### Prototype performance measurements

In order to *a)* see whether such an architecture would work using existing smartphone technology and to *b)* understand its performance capabilities especially of the underlying BLE packet transmission, we used three Nexus 5X phones to simulate a three-party social encounter, measuring how long it took for the phones to "register" each other once they get in proximity and how reliably a phone can pick up other peers' access tokens.

Our performance measures compare three different token-pubkey ratios (2.0:1.0; 3.0:1.0; and 5.0:1.5; all given in seconds) over the three BLE announcement rates supported by Android 6 (1 Hz, 3 Hz, and 10 Hz). For example, in the first case the system would thus send out tokens for, say, 2 seconds, followed by the device's public key (session) for 1 second; at a rate of 3 Hz, this would mean 6 token packets followed by 3 public key packets.

Figure 7 shows the public key and token reception rates per second, averaged over the results of all 3 phones running with each configuration for a period of 5 minutes. We see that increasing the transmission frequency from 1 Hz to 3 Hz improves reception rates, but a further increase to 10 Hz adds delay when simultaneously transmitting and scanning for packets. We hence use 3 Hz for the transmission rate.

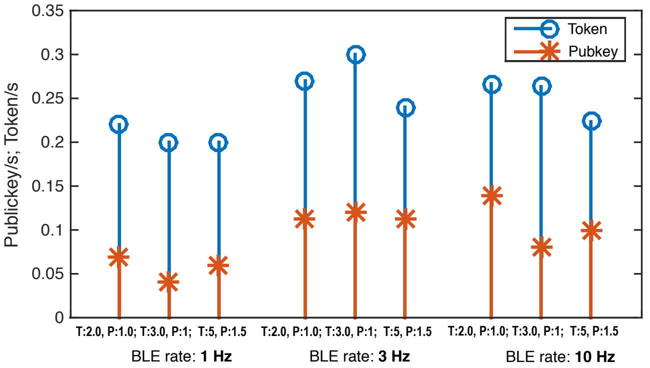


Figure 7: Average beacon reception rates for three token-pubkey ratios (all given in seconds) 2.0:1.0; 3.0:1.0; and 5.0:1.5, using three BLE announcement frequencies 1 Hz; 3 Hz; and 10 Hz.

With a 3 Hz packet transmission, the highest beacon reception rates (0.3 token/s and 0.12 keys/s) are achieved with 3 seconds of token transmission and 1 second of key transmission. This means that it may take up to 3 seconds before a device reliably receives a new token from another device; public key beacons will be picked up in less than 9 seconds (8.3 seconds). In practice, this means that once two devices are in range, it should take no more than 9 seconds (average 4.15 seconds) for them to "register" each other and, after sufficient dwell time $t_{dwell}$ has elapsed, to add each other's public keys to their respective encrypted URL uploads. Tokens should therefore not be updated more frequently than once every 3 seconds, otherwise a peer may miss a token (and thus be unable to access captured data for this period). These results were more or less dependent on the performance of the smartphones that we had, and clearly better results may be achieved with future hardware.

## LIMITATIONS

The presented architecture provides a secure and an easy way of exchanging captured data with other peers. Nevertheless, it also has some limitations related to the choice of using BLE for underlying beacon implementation and to the level of actors' commitment and fairness when it comes to uploading actual data to the shared URLs.

The pitfall when using BLE solely for proximity detection is that the its signal can be "heard" even by devices which are not in a close enough proximity to be considered as co-located peers. Another alternative that can reduce the proximity range of beacon announcements and improve the co-presence detection is to combine BLE with an audio sensing solution [16, 14, 17].

The architecture clearly does not guarantee a successful acquisition of self-captured data of other co-located peers, as users may not upload their data to their repositories in the first place. One has to rely on others willingness to perform the actual data uploading on a timely manner. At the end, other peers

[6] The TX Power Reference Value is a pre-measured signal strength at 1m distance from a beacon, which allows a recipient to estimate the actual distance of the signal sender.

are users that one meets and socially interacts with and one can always try to reach back other users and request them to upload the missing data.

Last but not least, we have not yet addressed the issue of IoT interoperability in our system, e.g., with established IoT frameworks (see, e.g., [11, 13]) or standards (see, e.g., [12, 15]).

## CONCLUSION

Lifeloging has the potential to offer significant benefits for applications to support human memory. However, due to limitations of worn devices, visual lifelogging data is often too constrained to fully capture personal experience (e.g., due to a poor point-of-view or visual obstruction). By enabling a seamless and secure exchange of captured data with both co-located peers and any available capture infrastructure (e.g. surveillance cameras), we aim to compensate such issues and allow for much richer representations of a previous experience.

Our initial prototype demonstrates how, within a few seconds, we can reliably pick up a set of identifiers (tokens and public keys) that allow for the secure exchange of captured data. We are currently working on a study design in order to conduct comprehensive "in the wild" user testing and performance evaluation.

## ACKNOWLEDGMENTS

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. Vannevar Bush. 1967. As we may think. *The growth of knowledge: readings on organization and retrieval of information* (1967), 23–35.
2. Daragh Byrne, Aisling Kelliher, and Gareth J.F. Jones. 2011. Life Editing: Third-party Perspectives on Lifelog Content. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 1501–1510. DOI: http://dx.doi.org/10.1145/1978942.1979162
3. Sarah Clinch, Nigel Davies, Mateusz Mikusz, Paul Metzger, Marc Langheinrich, Albrecht Schmidt, and Geoff Ward. The Big Picture: Lessons Learned from Collecting Shared Experiences through Lifelogging. (????). http://www.research.lancs.ac.uk/portal/services/downloadRegister/106871999/PC_PC_2015_02_0011.R1_Clinch.pdf
4. Sarah Clinch, Paul Metzger, and Nigel Davies. 2014. Lifelogging for 'Observer' View Memories: An Infrastructure Approach. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication (UbiComp '14 Adjunct)*. ACM, New York, NY, USA, 1397–1404. DOI:http://dx.doi.org/10.1145/2638728.2641721
5. N. Davies, A. Friday, S. Clinch, C. Sas, M. Langheinrich, G. Ward, and A. Schmidt. 2015. Security and Privacy Implications of Pervasive Memory Augmentation. *IEEE Pervasive Computing* 14, 1 (Jan. 2015), 44–53. DOI: http://dx.doi.org/10.1109/MPRV.2015.13
6. Yevgeniy Dodis and Nelly Fazio. 2002. Public Key Broadcast Encryption for Stateless Receivers. In *Digital Rights Management*, Joan Feigenbaum (Ed.). Number 2696 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, 61–80. http://link.springer.com/chapter/10.1007/978-3-540-44993-5_5 DOI: 10.1007/978-3-540-44993-5_5.
7. A. Fathi, J. K. Hodgins, and J. M. Rehg. 2012. Social interactions: A first-person perspective. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1226–1233. DOI: http://dx.doi.org/10.1109/CVPR.2012.6247805
8. Jim Gemmell, Gordon Bell, and Roger Lueder. 2006. MyLifeBits: A Personal Database for Everything. *Commun. ACM* 49, 1 (Jan. 2006), 88–95. DOI: http://dx.doi.org/10.1145/1107458.1107460
9. Jim Gemmell, Lyndsay Williams, Ken Wood, Roger Lueder, and Gordon Bell. 2004. Passive Capture and Ensuing Issues for a Personal Lifetime Store. In *Proceedings of the the 1st ACM Workshop on Continuous Archival and Retrieval of Personal Experiences (CARPE'04)*. ACM, New York, NY, USA, 48–55. DOI: http://dx.doi.org/10.1145/1026653.1026660
10. Martin Georgiev and Vitaly Shmatikov. 2016. Gone in Six Characters: Short URLs Considered Harmful for Cloud Services. *arXiv:1604.02734 [cs]* (April 2016). http://arxiv.org/abs/1604.02734 arXiv: 1604.02734.
11. Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems* 29, 7 (2013), 1645 – 1660. DOI: http://dx.doi.org/10.1016/j.future.2013.01.010 Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services and Cloud Computing and Scientific Applications – Big Data, Scalable Analytics, and Beyond.
12. Isam Ishaq, David Carels, Girum K. Teklemariam, Jeroen Hoebeke, Floris Van den Abeele, Eli De Poorter, Ingrid Moerman, and Piet Demeester. 2013. IETF Standardization in the Field of the Internet of Things (IoT): A Survey. *Journal of Sensor and Actuator Networks* 2, 2 (2013), 235. DOI: http://dx.doi.org/10.3390/jsan2020235
13. M Victoria Moreno, Miguel A Zamora, and Antonio F Skarmeta. 2015. An IoT based framework for user–centric smart building services. *International Journal of Web and Grid Services* 11, 1 (2015), 78–101.
14. Toshiya Nakakura, Yasuyuki Sumi, and Toyoaki Nishida. 2009. Neary: Conversation Field Detection Based on Similarity of Auditory Situation. In *Proceedings of the 10th Workshop on Mobile Computing Systems and*

*Applications (HotMobile '09)*. ACM, New York, NY, USA, 14:1–14:6. DOI: http://dx.doi.org/10.1145/1514411.1514423

15. Zhengguo Sheng, Shusen Yang, Yifan Yu, Athanasios V Vasilakos, Julie A McCann, and Kin K Leung. 2013. A survey on the IETF protocol suite for the internet of things: Standards, challenges, and opportunities. *IEEE Wireless Communications* 20, 6 (2013), 91–98.
16. Wai-Tian Tan, Mary Baker, Bowon Lee, and Ramin Samadani. 2013. The Sound of Silence. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys '13)*. ACM, New York, NY, USA, 19:1–19:14. DOI: http://dx.doi.org/10.1145/2517351.2517362
17. Hien Thi Thu Truong, Xiang Gao, Babins Shrestha, Nitesh Saxena, N. Asokan, and Petteri Nurmi. 2015. Using contextual co-presence to strengthen Zero-Interaction Authentication: Design, integration and usability. *Selected Papers from the Twelfth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2014)* 16, Part B (Jan. 2015), 187–204. DOI: http://dx.doi.org/10.1016/j.pmcj.2014.10.005
18. Chenren Xu, Sugang Li, Gang Liu, Yanyong Zhang, Emiliano Miluzzo, Yih-Farn Chen, Jun Li, and Bernhard Firner. 2013. Crowd++: Unsupervised Speaker Count with Smartphones. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '13)*. ACM, New York, NY, USA, 43–52. DOI: http://dx.doi.org/10.1145/2493432.2493435