TEO: Ephemeral Ownership for IoT Devices to Provide Granular Data Control

Han Zhang
Carnegie Mellon University
hzhang3@cs.cmu.edu

Yuvraj Agarwal
Carnegie Mellon University
yuvraj@cs.cmu.edu

Matt Fredrikson
Carnegie Mellon University
mfredrik@cs.cmu.edu

ABSTRACT

As Internet-of-Things (IoT) devices rapidly gain popularity, they raise significant privacy concerns given the breadth of sensitive data they can capture. These concerns are amplified by the fact that in many situations, IoT devices collect data about people other than their owner or administrator, and these stakeholders have no say in how that data is managed, used, or shared. To address this, we propose a new model of ownership, *IoT Ephemeral Ownership* (TEO). TEO allows stakeholders to quickly register with an IoT device for a limited period, and thus claim co-ownership over the sensitive data that the device generates. Device admins retain the ability to decide who may become an ephemeral owner, but no longer have access or control to the private data generated by the device. The encrypted data in TEO is accessible only by entities after seeking explicit permission from the different co-owners of that data. We verify the key security properties of our protocol underpinning TEO in the symbolic model using ProVerif. We also implement a cross-platform prototype of TEO for mobile phones and embedded devices, and integrate it into three real-world application case studies. Our evaluation shows that the latency and battery impact of TEO is typically small, adding ≤187 ms onto one-time operations, and introducing limited (<25%) overhead on recurring operations like private data storage.

CCS CONCEPTS

• **Security and privacy** → **Security services**; **Access control**; **Authorization**; Formal security models.

KEYWORDS

Internet of Things, ephemeral ownership, protocol verification, stakeholder privacy, access control

ACM Reference Format:
Han Zhang, Yuvraj Agarwal, and Matt Fredrikson. 2022. TEO: Ephemeral Ownership for IoT Devices to Provide Granular Data Control. In *The 20th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '22), June 25-July 1, 2022, Portland, OR, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3498361.3539774

1 INTRODUCTION

Internet-of-Things (IoT) devices are rapidly gaining popularity in both private settings (e.g., homes, offices) and public spaces (e.g., conference rooms) [32, 55]. However, their growing ubiquity has led to privacy concerns that often stem from the breadth of sensitive data (e.g. audio, video, images) that they can sense [11, 28, 41].

To address growing IoT privacy concerns, prior work proposes expressive access control mechanisms for authorization and delegation [10, 31, 46, 75], decentralized storage solutions based on blockchain [8, 27, 73, 76], and cryptographic access control schemes [4, 44, 53, 56, 68, 77, 78]. One key limitation of these approaches is that they lack support for exclusive user control. They focus primarily on the privacy concerns relevant to the device owners, and not other stakeholders who may have legitimate concerns about these devices since the data they capture may relate to their activities. These approaches often grant the sole device owner, or a small group of administrators, full authority over the data that the device generates, assuming these entities are trusted by those who may be impacted by the devices.

As pointed out by a growing body of research, these assumptions are not always consistent with the emerging ubiquity of smart devices in shared spaces. Passive bystanders may nonetheless be impacted by a smart device simply by, for example, visiting someone's home [11, 88], or by incidental exposure in public areas [21]. A related but distinct scenario is illustrated by the presence of IoT devices in short-term rentals (e.g., Airbnb), where guests may wish to use whatever smart devices are in their units, but have concerns about the host's ability to invade their privacy [25, 60]. Both cases highlight the lack of support for *stakeholders' privacy* in existing access management systems. In situations like these where there is an implied expectation of privacy, any user in a device's vicinity who is impacted by its sensors should have a say when it comes to the device's functionality and the data it generates. A key challenge that we seek to address is giving stakeholders control over devices that impact them, so that they can decide how these devices operate, and who has access to any data that emanates from their sensors.

We propose TEO — IoT Ephemeral Ownership — a model of device ownership that splits the traditional fixed IoT owner role into "admin" and one or more "ephemeral owners", and a corresponding authorization protocol that embodies this model. Take as a working example the home rental scenario. As the admin, the host of the rental can install TEO-enabled devices in the rental unit, claiming physical ownership in an initialization phase. When a rental guest arrives, they claim "ephemeral ownership" of the device, assuming exclusive control over the device's operation and any data that it generates during their stay. Incoming commands to the device must be authorized by the ephemeral owner, and data stored by the device will be encrypted so that only the ephemeral owner is able to later access it. To facilitate data accessibility and ease the on-device storage burden, TEO provides a way to leverage untrusted third-party cloud storage providers.

MobiSys '22, June 25-July 1, 2022, Portland, OR, USA

ACM ISBN 978-1-4503-9185-6/22/06.
https://doi.org/10.1145/3498361.3539774

After the ephemeral owner's term expires, new guests can claim the device and become ephemeral owners. Previous guests will no longer have the authorization to control the device, or any *new* data that it generates, but they preserve ownership of the data captured when they were the ephemeral owner. At any point in the future, when someone wants to access the data for a time period (e.g., Airbnb's customer service to resolve a dispute), they must obtain permission from the ephemeral owners when the data was generated.

Smart devices are also being deployed in shared spaces and raise similar privacy concerns. For example, conference rooms have frequently changing lists of occupants, so control over its devices and their data must be flexible enough to support a constantly-changing association between devices and users. Complicating matters further, several people share the space simultaneously, so authorization needs to support ephemeral ownership *groups*. When a future request is made for data owned by such a group, the system needs to establish that its members consent according to an agreed-upon policy.

To achieve this vision, we design a suite of protocols for the essential operations in typical device lifecycle: device initialization, claiming ownership, routine control and data storage, and third-party access and revocation. We tackle a number of challenges to fulfill our design goals. To support frequent and efficient changes in ownership, we group data into small time segments, and use unique session keys to denote ownership of each segment. To support variable-sized groups, we leverage Shamir Secret Sharing [79] to distribute control across stakeholders. However, the smaller the segment duration is, the more data keys are needed for decryption. We introduce another layer of encryption blocks to aggregate data keys and facilitate fine-grained data sharing. Finally, to enable data access revocation with low communication overhead, we incorporate key homomorphic encryption [18, 85] to enable the storage provider to re-encrypt contents on the user's behalf without being able to decrypt it into plaintext.

To mitigate the potential design vulnerabilities, we formalize our security goals and model the TEO protocol specification with a well-known protocol verifier ProVerif [15] to ensure TEO's security and correctness. We address several modeling challenges of TEO, particularly in formalizing group ownership, key splitting, and revocation. This iterative process of modeling, verifying, and revising our protocol uncovered several vulnerabilities in our design along the way. In the end, we were able to prove all specified security properties, including secrecy, mutual authentication, resilience to data spoofing, and revocation, for variable group sizes.

We implement an open-source TEO prototype[1] as a cross-platform library and develop client applications for mobile devices, single-board computers, and traditional Linux machines. We integrate three real-world IoT applications with TEO to show how they can preserve owners' private data and enable direct control over their devices. We also show that integrating these applications using TEO's API requires minimal changes to their existing codebases. Our evaluation results on both micro-benchmarks and real-world apps show that TEO is indeed practical and incur insignificant battery overhead on mobile devices. Specifically, one-time operations such as initializing devices and claiming ownership take ≤187 ms. Meanwhile, repetitive operations such as data storage depend on the data size and their overhead is dominated by the upload speed to the storage provider. For smaller files, TEO's overall latency is 101-308 ms, while for larger files, TEO incurs 7-25% extra overhead when compared to the baseline latency of just uploading the data.

In summary, we make the following contributions:

- We motivate the need for *stakeholders' privacy* in smart device deployments and identify the challenges arising from the conflicts between the power of device administrators and the lack of protection for users.
- We propose TEO to enable *ephemeral ownership* for IoT devices by splitting the fixed IoT owner role into different entities. We design a set of protocols to enable TEO, covering device operations, access management, revocations, and group ownership.
- We encode all TEO operations as models and verify that they satisfy goals of confidentiality, authentication, and correctness using the symbolic verifier.
- We implement the TEO protocol as an open-source, cross-platform library and integrate three real-world IoT applications with TEO with minimal changes. Our evaluation shows TEO incurs low overhead with ≤187 ms for one-time operations and 7-25% added latency for repetitive operations.

2 RELATED WORK

Stakeholder Privacy. Recent work has identified the challenges due to the discrepancy between decision makers and device users [22, 41, 91], partially motivating the need for stakeholder privacy. Some of this work examines privacy issues from the perspective of bystanders [11, 88] and incidental users [21]. TEO further expands the set of stakeholders and, more importantly, designs and implements a novel system to address several practical challenges such as flexible group ownership, preserving data ownership, and access control and revocation. Moreover, research on preventing intimate partner violence (IPV) for smart devices [33, 34, 40, 70, 84] also echoes our goals for stakeholders' privacy. TEO-enabled devices should protect all users' security and privacy and prevent over-privileged admins in the status quo. However, addressing IPV issues is more difficult since tech-savvy abusers can just avoid using TEO-enabled devices.

Smart Device Access Control. Motivated by real-world security incidents and research results that highlight the risks stemming from mismanaged IoT delegation chains [89], researchers have proposed many improvements and novel access control systems for smart devices, as surveyed by He et al. [42]. Prior work have proposed improving access control policy language with expressive authorization logic [10] and contextual information about the home environment [31, 46, 75]. Specifically, Kratos [80] designs an access control system over multi-user multi-device-aware smart homes. Complementary to providing granular access control from a temporal aspect, I-Pic [1] proposes a novel approach to enforce privacy policies for pictures owned by groups, while Hivemind [51] introduces mechanisms to enable collectively control of shared public IoT actuators. Moreover, many decentralized authorization

[1] Our code and formal security models are available at https://github.com/synergylabs/TEO-release.

Table 1: Related work on smart home access controls compared to TEO as we will elaborate in Section 2.

Work	Temporally granular access control	Frequent changes in device-user association	Exclusive user control	No trusted central storage	Formally security guarantee
Expressive policy languages [10, 31, 46, 75, 80]	Yes	No	No	Partial	No
Decentralized authorization [4, 27, 73, 76]	Yes	No	No	Yes	No
Cryptographic access control [53, 77]	Yes	No	No	Yes	Yes
TEO	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

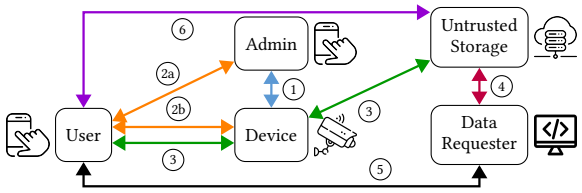


Figure 1: Overall TEO workflow. An admin initializes the device (①). Next, the user claim device ownership with the admin's pre-approval (②a and ②b). During normal operation (③), the device encrypts users' data and uploads it to storage. A requester can download the data (④), but needs the owner's approval to decrypt it (⑤). To revoke access, the user can directly issue a request to the storage provider (⑥).

frameworks, including Wave [4] and several blockchain-based approaches [8, 27, 73, 76], further eliminate the need for central trusted entities. Finally, the use of novel cryptographic constructions to facilitate access control and delegation has been present in several instances of recent work [44, 53, 56, 68, 77, 78].

Table 1 compares TEO with some of the closely-related smart device access control mechanisms. In summary, TEO aims to provide users with exclusive control over IoT devices and their data, while previous approaches using delegation chains cannot provide such support since the root and upstream nodes in the chain (admins and floor managers) inherently retain control as well. Moreover, TEO provides mechanisms for frequent, flexible device-user associations, enables dynamic group ownership for shared devices, and does not rely on trusted third-party services. Furthermore, by formally verifying the protocol underlying TEO's we can provide strong security guarantees that many previous approaches are unable to.

Protocol Verification. Protocol verification is widely used by many research projects, as surveyed by prior work [9, 16]. Verifiers are very helpful in ensuring the security properties of standardized protocols with real-world deployments such as TLS [12, 23], Bluetooth [87], and IoT messaging protocols [86]. In addition, projects like Koi [37] and SST [50] also leverage protocol verifiers to assist the development of novel protocols for mobile and IoT platforms. Taking a similar approach, we integrate verification deeply with TEO's design process to iteratively refine and correct uncovered security bugs while addressing several modeling challenges.

3 SYSTEM OVERVIEW

To address stakeholders' privacy and security concerns, we envision a new model of device ownership that protects the interests of both the device users and its administrators. We propose TEO – IoT Ephemeral Ownership – that grant users, as ephemeral owners, full control over the device's operations. Historical data collected by the device will always belong to the ephemeral owners. When someone wants to access the data, they have to get permission from all the data owners (stakeholders). While the device administrators decide who can claim the device to become an ephemeral owner, they cannot interfere with the device's operation or access private data captured by it without the owners' approval. To help contextualize TEO's workflow, we use a running example of a group of friends renting an Airbnb house for the rest of this section.

The high-level TEO workflow is illustrated in Figure 1. The admin (e.g., Airbnb host) first installs the IoT device and initializes it. (Step ①). Afterward, the device is ready to be claimed by new owners only if they are authorized by the device admin. The potential users (e.g., Airbnb guests) all have a user agent program running on their phones. After booking their reservation, they need to ask the host to issue "pre-auth tokens" with everyone's public key (Step ②a). Pre-auth tokens prevent unauthorized people such as malicious neighbors from accessing the device. When users arrive at the Airbnb rental, the user agent on their phones initiates a process to claim the device (Step ②b). As the group membership changes (users join and leave), the device dynamically adjusts the set of owners. During normal operation, the device protects its stakeholders in two ways. First, if the device receives commands to perform actions (e.g., open the door, adjust the temperature), it needs to ensure the command is authorized by the current owners (omitted from the figure). Every command includes a certificate with user-generated signatures, which the device can verify using the current owners' public keys. Second, the device preserves users' data ownership with a series of encryption operations and distributes individual data keys to the set of owners (Step ③). To facilitate future data access and reduce users' storage overhead, TEO-enabled devices directly upload the encrypted data to any untrusted cloud storage provider. When any entity wants to request access to this encrypted data (Step ④), they need to seek the permissions of the original owner(s) to decrypt it (Step ⑤). Later on, if a user decides to revoke the access, they can directly contact the storage service provider and provide re-key tokens (Step ⑥) generated with key homomorphic encryption. This special cryptography primitive allows the storage provider to switch encryption keys of the ciphertext by directly

applying user-provided tokens. In other words, key revocation can be performed by the storage provider on the encrypted data itself without having to decrypt it first.

3.1 Target Use Cases

TEO aims to provide ephemeral owners with full control over who can access their data and when regardless of data types. Therefore, TEO is well suited for applications and IoT devices that store operational data that could contain sensitive information about their users, such as camera and speaker recordings or sensor readings. For data accesses, we primarily focus on scenarios where data requesters want to use historical data for analysis, such as to train machine learning models or to recall past events.

Since TEO devices maintain an up-to-date list of current owners, we can extend TEO to enforce real-time access control of the device as well. Consider, for example, a smart door lock. After being claimed by a new user, this lock should reject commands issued by previous owners. To achieve this, the device can require that all incoming commands include an authorization certificate signed by the owner, and we implemented a simple application as part of our case studies. Although not the current focus of TEO, we believe it would be a useful future direction to enrich certificate designs with proof-carrying authorizations [6, 10, 13, 54] for more expressive policy specification languages.

TEO targets a variety of deployment scenarios such as rental homes and shared offices. Specifically, they have different design requirements and considerations. In rental homes (e.g., Airbnb), the host sets up smart devices and lets guests use them. Guests have lower churn rates (stay a few days at least) and smaller group sizes. Sometimes, a single owner would suffice as the group implicitly trusts each other if they stay at the same place. In contrast, smart devices in shared offices and conference rooms have more frequent changes in owners and group members would prefer an equal role in decision making. These devices would be managed by the building managers, and they may have more insight on users' daily routine (e.g., which floors and rooms they are likely to occupy). Building managers can potentially use this information to improve the practicality of TEO. For example, they can selectively issue pre-auth tokens to conference rooms based on a user's calendar events.

3.2 Design Goals

Flexible Association of Devices and Users. We expect frequent ownership changes in physical spaces with smart devices. An Airbnb may see ownership changes that span days, while shared spaces in smart buildings (e.g. conference rooms) may see ownership changes even hourly. Moreover, multiple users can share an office and hence be collectively impacted by the devices. Ideally, all stakeholders should have a say in controlling the device and accessing the data it collects. Unfortunately, existing smart home access control systems often assume a static group of user(s) make all these decisions.

Preserving Data Ownership. Data collected by smart devices should always belong to the group of users present at the time of capture. Anyone trying to access the data should request the data owners' permission. Most importantly, dynamically changing ownership of the device's users and administrators should not affect historical data ownership. This requirement ensures that users preserve their control over the private data even if they no longer own the original device in the future.

Decentralized Trust. Users should be able to manage access requests without relying on third parties. Centralized access control systems, managed by individual companies and building owners, require complete trust in these entities and in their ability to protect users' data and enforce access policies. In return, centralized systems provide an efficient solution for processing access requests and sharing data. On the contrary, we want to empower users to decide who should have access to their data themselves while benefiting from the performance and availability of cloud services.

Formally Verified Security. Our goal is to provide formally verified security guarantees for our proposed TEO system. The main components of TEO are a series of complex communication protocols designed for multiple entities in IoT deployments. Therefore, we encode our protocol specifications into models and verify their security and correctness under our stated threat model in Section 5. After several rounds of refinement, our streamlined TEO design provides assurances of security and correctness.

3.3 Threat Model

We designed TEO under the assumption of a powerful attacker, who can monitor all communications between users, devices, and the third-party storage provider, attempts to undermine its goals by either *(1)* controlling the device without the active consent of the ephemeral owner, *(2)* accessing the data generated by the device, which should only be accessible by the ephemeral owner, or *(3)* impersonating one of these parties. This follows the Dolev-Yao network attacker model [26] used in our formal analysis (Section 5). Concretely, such an attacker might correspond to someone in the vicinity of the device, a malicious admin who wishes to violate the privacy of an ephemeral owner, or a previous ephemeral owner who aims to extend their control of the device and its data past the agreed-upon terms.

We assume that local devices are trusted to correctly execute the protocol, i.e. the TEO-enabled device has not been backdoored or rooted, and will not leak data, encryption keys, or bypass authorization checks. We assume that the third-party storage providers may be *passively* malicious (i.e. honest-but-curious): they might attempt to extract private information from the data they receive, but will faithfully execute the TEO protocol as specified. This is consistent with using reputable cloud services, with whom users may not trust storing cleartext data, but for whom the reputational risk stemming from actively-malicious behavior is too great.

Our formal analysis (Section 5) is in the symbolic model, so we must also assume that the cryptographic primitives used by the protocol are secure against computational attacks. Likewise, our analysis does not consider information that might be leaked from metadata (e.g., the identity of users who participate in the protocol, from their public key certificates), nor semantic information that might leak through traffic analysis (e.g., inferring user behaviors by observing the timing and sizes of encrypted network packets). While these may provide opportunities for attackers to

learn unwanted information in certain settings, we leave careful consideration of these risks to future work.

4 TEO PROTOCOL

We now describe the TEO protocol and the workflow between device administrators, users, storage providers, and devices. We start by introducing the key challenges that we sought to address in our design and describing our notation.

Granular Ownership & Data-Sharing. First, we envision TEO's use in settings where device ownership changes frequently, and the duration of ownership varies drastically (e.g., using a conference room for tens of minutes, or renting a house for several days). We accommodate these in TEO by partitioning the ownership period into relatively small segments, with each segment tracking its set of owners (potentially distinct) from adjacent segments. A new user wishing to claim (potentially shared) ownership of a device takes effect at the start of the next available segment. This approach facilitates fine-grained data sharing of selected time windows of data rather than all-or-nothing sharing. However, to do so requires generating fresh keys even when ownership remains the same across segments. Configuring a relatively small segment interval affords flexible and responsive transitive ownership, at the cost of the corresponding overhead of managing cryptographic state for each segment, and additional rounds of TEO communication.

The storage overhead introduced by this scheme may be significant for mobile user agents. However, for many of the envisioned use scenarios for TEO, data sharing requests are likely to access several contiguous segments, e.g., in the case of streaming video or sensor readings. Thus, to mitigate the storage overhead, we designed TEO to securely store groups of keys from a single session on the untrusted cloud storage, so that the user agent is only responsible for maintaining a single key for the entire session. This mechanism also enables efficient group ownership, using Shamir Secret Sharing [79] to split data block keys across group members.

Efficient Revocation. Finally, efficient key revocation is challenging, as it is infeasible for resource-constrained user agents to download, locally re-encrypt, and re-upload ciphertexts to the storage. We leverage key-homomorphic encryption [18, 85] to facilitate re-encryption directly on the untrusted storage provider, requiring users to only generate fresh rekey-tokens.

4.1 Notation

We use three main encryption primitives in our protocol. Each is denoted by $\text{Enc}(\cdot)$ for encryption and $\text{Dec}(\cdot)$ for decryption, but vary by the set of keys that they take. Symmetric-key encryption is represented as $\text{Enc}_k(n, m)$ with key k, nonce n, and message m. Long messages are concatenated with multiple parts $(m_1|\cdots|m_n)$. Encrypting a message with a recipient's public key is denoted by $\text{Enc}_{pk_R}(m)$ where pk_R is the receiver's public key. We use public-key authenticated encryption to protect the message's confidentiality and integrity when the public keys of both parties are known, denoted by $\text{Enc}_{\langle sk_S, pk_R\rangle}(n, m)$ where sk_S is the sender's secret key. Finally, we use key homomorphic encryption from Sieve [85] to support revocation, and denote this operation by $\text{SieveEnc}_{K_s}(n_s, m)$, where K_s is the Sieve key.

4.2 Device Initialization

New devices need to go through a one-time initialization process and obtain a valid device proof (*DP*), as illustrated in Figure 2a, before they can start regular operation and accept new owners. The device proof is necessary for the device to demonstrate its authenticity to potential owners in later steps. To facilitate mutual authentication, devices and administrators need to establish an out-of-band communication channel to share setup keys. This is common with smart devices using existing solutions including QR codes printed on devices, physically-printed passkeys packaged with the device, and short-range wireless communication [69]. Our prototype implementation uses QR codes to share setup keys in this phase of the protocol. If the device needs to change its administrator, it must be reset, and the initialization process needs to be completed again with fresh out-of-band key material. Note that this is different from changing *ephemeral* owners, which we describe next.

4.3 Device Ownership Management

Device Discovery. When users enter a new environment, they can discover TEO-enabled devices to obtain public information about the device, such as its public key and admin information. This can be achieved in several ways. First, they can locate the device and scan the QR code dynamically generated and displayed on the device. However, many IoT devices do not have displays to provide such functionalities. Alternatively, the device can advertise its presence and communicate this information over a local network, similar to service discovery functions used in existing protocols [7, 67, 81, 92]. Users broadcast discovery requests to all local hosts and TEO devices will respond with their information. Since this request is broadcast, users may receive responses from devices located in nearby rooms and offices. To reduce false positives, we envision several potential extensions to improve TEO's usefulness and practicality in future deployments. First, the reply messages can include additional information about the device's location, so users can choose the correct device based on their own location. In addition, admins can decide who are capable of claiming certain devices by restricting pre-auth tokens to users. For example, Airbnb hosts can give tokens based on guests' emails, and building managers can provision tokens based on users' locations (buildings, floors, etc.). Finally, the device can ask its current occupants to moderate new users' join requests since they can physically confirm whether the new users are in the room.

Proximity Detection and Duration Setting. We propose a Bluetooth Low Energy (BLE) based proximity detection mechanism for the TEO enabled device to detect when ephemeral owners leave its vicinity. For simpler scenarios such as home rentals, this might be unnecessary since the owners' stay has a fixed duration. However, this mechanism would alleviate challenges in shared spaces such as offices and conference rooms where users frequently enter and leave. Figure 2b shows the workflow of TEO's BLE proximity detection. The device periodically generates new proximity nonce n_p and broadcasts it through BLE. We do not require a high frequency of new proximity nonce generation, since the device only needs to know if the owner is part of the current data block recording (e.g. one data block per minute). Since multiple users can collectively own the same device, we set the device in BLE beacon mode so it

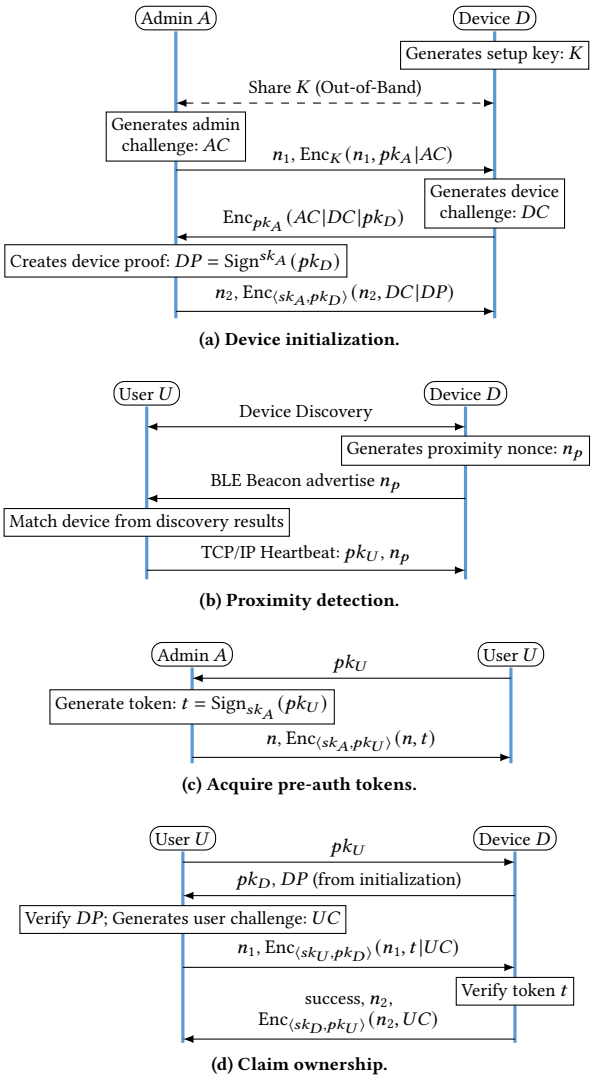


Figure 2: Protocol workflow for device initialization and ownership management.

continuously advertises information without requiring an explicit connection from the user's phone. On the other side, the TEO app on the user's phone periodically sends out heartbeat messages with the latest nonce sent within the device advertisements over BLE. Once the device stops receiving the correct nonce from a particular user's phone for an extended period, it can infer that the user must no longer be in BLE range and thus remove them from the list of ephemeral owners. The device should be configured with proper transmission power so only nearby users likely in the same space can receive the BLE advertisement while reducing false positives. This automated process is executed by the user's phone app to minimize user burden. However, as a fallback mechanism, users can manually specify the duration of their occupancy in a space (or use default values) to be included in the ownership claiming process.

This mechanism is designed to ensure current ephemeral owners are still within the device's vicinity. Consequently, the device can quickly remove stale owners that have already left the room. On the other hand, this mechanism is not intended to limit who *can* claim devices (the responsibility of pre-auth tokens) because BLE could have high false positive rates (i.e., everyone within the range will receive this message).

Claiming Ownership. Figures 2c and 2d illustrate the steps to claim ephemeral ownership. First, the user acquires a "pre authorization token" from the administrator (Figure 2c). The protocol uses these tokens to prevent unauthorized access to a device, and not to enforce more granular, device-specific usage policies. For example, rental hosts can generate pre-auth tokens for guests with upcoming reservations, and building administrators can give pre-auth tokens for the devices in a specific office to those who are allowed to use them.

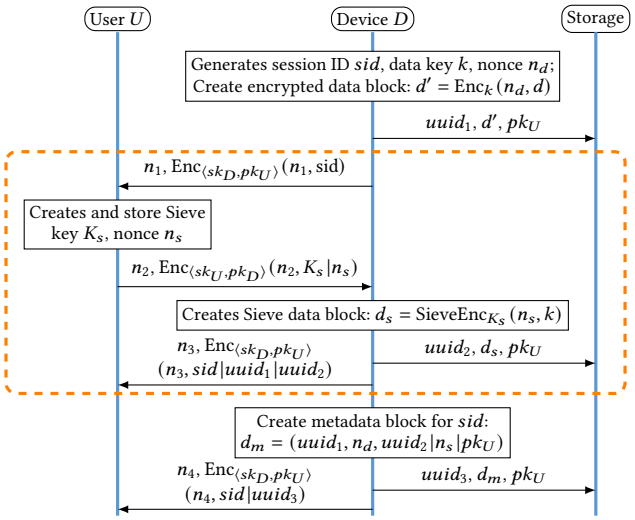
Figure 2d illustrates the next step in establishing ownership. Just as the pre-auth token establishes the authenticity of the ownership claim to the device, the device proves its authenticity to the new owner by issuing its device proof (DP) obtained during initialization. The user also generates a fresh random challenge (UC) to prevent replay-enabled device spoofing. Extending this phase to support group ownership is straightforward: each user performs the steps in Figure 2d independently, and the device tracks the list of ephemeral owners accordingly. The device is then responsible for synchronizing control between its current owners.

After a user becomes an ephemeral owner, the device can enforce access checks for incoming commands and instructions from a cloud back-end or home automation platform (e.g., IFTTT) to prevent previous owners from retaining controls. To support this check, every automation applet or cloud service should obtain authorization in the form of signed certificates from current owners.

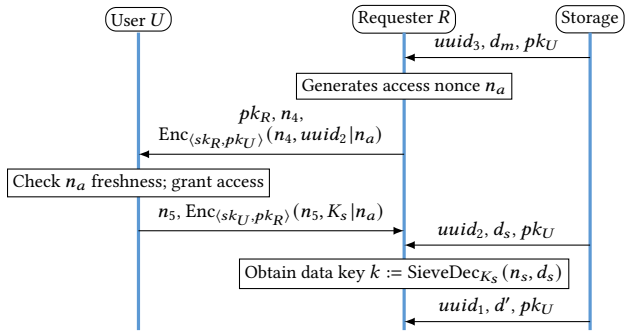
4.4 Data Storage and Access

To preserve users' ownership of data generated by the device, one approach would be to transmit any such data directly to the user, and let them manage it independently. This is too demanding for mobile user agents, and even if it were not, would impose an unnecessary burden on them. Our protocol instead uses a third-party cloud storage provider that is honest but curious: we expect it to correctly store data from the device and respond faithfully to users' requests, but do not trust it to refrain from attempting to inspect the confidential data. To protect the confidentiality of the device data, which may contain sensor readings or video recordings that users consider private, the device could encrypt the data before sending it to the cloud provider, and provide the user with all of the keys necessary to access it in the future. This poses several challenges, including how to support group ownership while managing data-sharing and subsequent revocation requests.

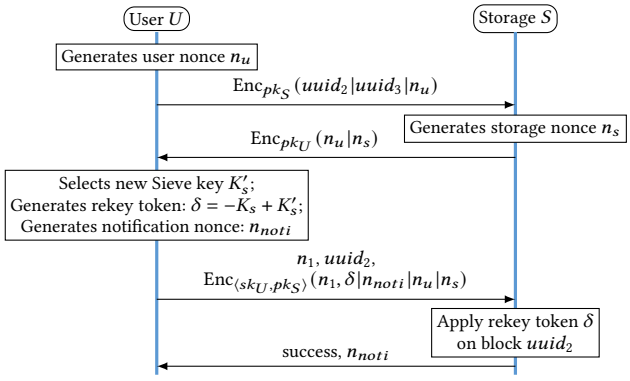
Sieve Encryption. In addition to standard cryptography operations, we incorporate a key homomorphic cipher proposed in Sieve [85] in TEO's protocol design. Key homomorphism [18] allows an entity to change the encryption key of a ciphertext without seeing the underlying plaintext. This characteristic is well-suited for untrusted TEO storage providers to assist in the process of



(a) Example data storage workflow. Nonces with numerical subscripts are local variables, only used within the corresponding protocol flow. The orange box indicates user-specific actions in group ownership.



(b) Example data access workflow. The requester wants to access the data associated with $uuid_3$ from the previous case. For brevity, the steps involved with sending download requests for UUID to the storage are omitted.



(c) Revocation workflow.

Figure 3: Data encryption and access workflow.

revoking data access without ever being able to decrypt the data itself.

Storing Data. Here we explain the data storage process for a single owner, as we will discuss group modes later. Figure 3a illustrates how TEO addresses these challenges and the steps taken to store a user's data in the cloud. First, the device encrypts the data with a freshly generated session key k for the current time segment. We use symmetric encryption for computational efficiency. The device uploads the encrypted data block to the storage provider, with an identifier $uuid_1$. Next, the device obtains Sieve credentials from current owners (Sieve keys K_s and Sieve nonce n_s). Afterwards, the device constructs the Sieve data block for this session and encrypts the value of session key k with Sieve cipher. This Sieve data block is then uploaded to the storage provider with an identifier $uuid_2$. Finally, the device uploads a metadata block to the storage provider, containing all the information needed to locate the encrypted data, as well as the Sieve blocks and the nonces used for the Sieve cipher. Meanwhile, the owner can collect bookkeeping information (such as session IDs and block UUIDs) from the device asynchronously.

To grant access to an encrypted block, the user can distribute the Sieve key to the requester. The symmetric key used to decrypt the data is stored in the Sieve block on the storage provider. Figure 3b shows the process for someone to request data access. After receiving the Sieve key, the requester can gather all the information necessary to decrypt the requested data.

Threshold Encryption. One building block to enable group ownership is the well-established threshold encryption; specifically, we use Shamir Secret Sharing [79]. At a high level, the t-of-n threshold encryption allows protecting a secret message with n key shares. To decrypt the message, someone only needs to collect t shares ($t \leq n$). The values of t and n must be set statically before the encryption process begins.

Group Ownership. To extend the protocol to support group ownership, the device needs to collect Sieve key information from each owner in the group by repeating the steps in the orange box of Figure 3a. Assuming a group of owners with public keys $pk_G = [pk_{U_1}, \cdots, pk_{U_N}]$, the device still encrypts the data with the session key k the same way as before, and then uploads the result to the storage provider as $(uuid_1, d', pk_G)$. The device then splits the data key k into N key shares $k_1, \ldots, k_N$. It constructs one Sieve data block for each owner, $d_{s_i} = \text{SieveEnc}^{K_{s_i}}(n_{s_i}, k_i)$, and sends them to the storage provider as $(uuid_{2_i}, d_{s_i}, pk_{U_i})$. Finally, the device constructs a metadata block $uuid_3$ that refers to each of the owners and their corresponding Sieve blocks, and stores it on the storage provider:

$$d_m = (uuid_1, n_d, uuid_{2_1} | d_{s_1} | pk_{U_1} | \cdots | uuid_{2_N} | d_{s_N} | pk_{U_N})$$

To access a shared data block, the requester needs to seek permission of each owner and obtain the Sieve key for their share of the data key.

We conclude by noting that threshold encryption can support several data access policies by adjusting the threshold value t. Currently, TEO requires that accessors have the approval of all group members (by setting $t == n$) because we want to give everyone the right to veto. It is straightforward to extend TEO with alternative policies (e.g., requiring majority approval by choosing $t > n/2$). In addition, a future extension of TEO can include another layer of threshold encryption for individual users. Each user can save

multiple key shares on different agents (laptops, phones, backup codes) and have a threshold value $t == 1$ in case they lose devices.

4.5 Revocation

TEO uses three blocks to encrypt each segment of data uploaded by the device and to support efficient revocation of access to a given block — the encrypted payload, a Sieve block, and the metadata block in plaintext. If the revocation was not needed, then the design could be simplified and storage overhead mitigated by dropping the Sieve block, and granting access by sharing the data key directly. Instead, our approach manages access by treating the Sieve key as a credential so that revocation can be accomplished by having the storage provider re-encrypt only the Sieve block, which is small in size relative to the actual ciphertext.

Figure 3c illustrates revocation in TEO. The user generates a rekey-token δ, and sends it to the storage provider along with identifiers for the appropriate Sieve blocks. The storage provider re-encrypts these blocks using the rekey-token. A benefit of this design is the consolidation of multiple encrypted data blocks, each with its own key, into a single Sieve block while maintaining low overhead on the client's side for revocation. The client can generate a fixed size rekey token to change the Sieve data block, thereby avoiding the need to download and re-encrypt arbitrary sized Sieve data blocks containing multiple data keys. Unifying encrypted data blocks in this way is especially helpful when a session contains a series of smaller data chunks, for example, an hour's worth of video recording may be stored as one-minute chunks to accommodate frequent membership changes and granular sharing, but the user is not burdened with managing credentials for each of these chunks individually. Additionally, each owner in a group can make access control decisions independently by rekeying their corresponding Sieve block.

4.6 Partial Availability

To process access requests, data owners (users in Figure 3b) need to be online. This requirement in TEO is intentional to give users direct control over their data, as all access requests must seek their direct approval. However, the limitation is that even if a single user is unreachable, no one can access the original content even if they have everyone else's permission already. To strike a balance between data availability and users' access control, the group of users can modify the access policies at the time of data recording to choose different values of t for the threshold encryption (as discussed in Section 4.4).

Moreover, individual users may lose their mobile devices and thus lose their key shares. Aside from periodic backups, one popular solution for implementing recovery mechanisms is to leverage threshold encryption (as mentioned in Section 4.4 and demonstrated by prior work [85]). We acknowledge that such a mechanism is important for future TEO deployment in the real world.

Finally, users may be temporarily unreachable when the device executes the data storage operation. Our protocol design ensures that unresponsive users will not block the main data encryption and upload functions (first step in Figure 3a). If the device cannot reach a user, it can store the user's key shares locally and retry later. Eventually, the device deletes its local copy once the user is back online. Meanwhile, since the private data have already been uploaded, the device does not need to keep the data while waiting for the unresponsive users.

5 SECURITY ANALYSIS

We formally model TEO using a well-known protocol verifier, ProVerif [15], and encode several key security properties to verify TEO's security and correctness. In modeling TEO we address several challenges, particularly in formalizing group ownership, key splitting, and revocation.

5.1 Security Goals

We aim to achieve the following security goals with our TEO protocol design.

- **Secrecy.** A user's private data, once encrypted by a TEO supported device, should not be accessible by anyone without the explicit authorization of the user. For group ownership, the policy requires that only entities with the consent of all owners are able to access the data.
- **Mutual Authentication.** After the device is initialized and claimed (Section 4.3), all parties must mutually authenticate and agree on each other's roles (i.e., device and admin, device and owner must acknowledge each other).
- **Prevent Data Spoofing.** Attackers should not be able to spoof data, potentially overwhelming users' local storage space with keys for non-existent data blocks. If the user concludes a data store operation, then the device must have indeed stored the user's private data for the corresponding session.
- **Effectiveness of Revocation.** If the data requester's access is revoked by the owner, the requester should not be able to decrypt the data block if they download it again from the storage provider. Conversely, revocation should only happen when the owner requests it, and the new key should be able to decrypt the data in the future. For groups, an individual's decision should not affect others (i.e., others' keys should still work since they did not revoke their keys). Note that TEO does not preclude a requester from storing the already decrypted data offline perpetually.

5.2 Modeling Protocol Workflow

We aim to identify protocol design-level bugs that may compromise TEO's security and privacy protections. We assume that the cryptographic primitives (e.g., encryption algorithms) are secure. Hence, we choose a *symbolic* verifier (ProVerif [15]) since it requires lower human guidance and is better suited for automated analysis compared to *computational* ones. Interested readers can refer to prior work for a more detailed discussion of different types of protocol verifiers [9, 14].

In ProVerif, protocols are modeled as sets of *processes*. Each process can generate fresh internal variables and local secrets, such as private keys that are hidden from the attacker. We represent each entity (e.g., admin, user, device) as its own process that can spawn and execute repeatedly to conduct multiple rounds of communication.

We symbolically encode all cryptographic operations so that incorrect credentials (e.g., decryption keys and invalid signatures) will terminate the process' execution. Processes communicate over *channels*. Attackers can intercept, drop, or fabricate any message over the channel. Since attackers can obtain a complete history of all network messages, we can conservatively model the untrusted storage provider with this general network attacker.

5.3 Modeling Security Goals

We encode every security goal from Section 5.1 with concrete ProVerif queries to ensure TEO's protocol design satisfies all security properties. We create unique *events* as checkpoints for the execution of each process. We construct *correspondence* queries to encode properties such as "*if A happens, B must have already happened*". Correspondence queries can be *injective*, which means that the verifier will check that there is a strong one-to-one mapping between events. To ensure correctness, we also add *reachability* queries to verify that all events are reachable during process execution; unreachable events can vacuously satisfy correspondence queries, leading to a false conclusion of TEO's correctness.

Secrecy. We create a private variable userPrivateData to represent the confidential information. This variable can be shared across different processes but remains hidden from the network attacker. The device encrypts this variable and uploads the ciphertext to the storage. We construct a secrecy query to verify that this variable remains secret from the network attacker and untrusted storage provider.

To verify that a data requester (requesterPK) can only decrypt the data with approval from all owners (owner_1, owner_2, ...), we construct a query that the event AccessData(requesterPK, userPrivateData) from the requester is preceded by GrantAccess(owner_i_PK, requesterPK, dataUUID) events from all data owners $i \in [1, N]$.

Mutual Authentication. We encode several injective correspondence queries to verify this goal. For initialization, the query states that whenever the event DeviceAcceptAdmin(devicePK, adminPK) happens on the device, the predecessor AdminAcquireDevice(adminPK, devicePK) must already have occurred for the admin.

For claiming device ownership, the device needs to verify that the user has a valid pre-auth token before accepting new owners. Therefore, the final event UserFinishDevice(userPK, devicePK, adminPK) on the user process should be preceded by the device-generated DeviceAcceptUser(userPK, devicePK, adminPK, preAuthToken), which itself should be preceded by the event that admin marks this token as valid, AdminGrant(adminPK, userPK, preAuthToken).

Prevent Data Spoofing. When the user finishes the Sieve key negotiation in data store (Figure 3a), it produces the event UserStoreFinish(userPK, devicePK, sessionID). This event should be preceded, injectively, by the device side event DeviceFinishSieve(userPK, devicePK, sessionID) issued after it finishes uploading Sieve data block to storage. Violating this query will cause users to store information for non-existed sessions.

Effectiveness of Revocation. We encode the revocation process in different *phases*, a cross-process synchronization primitive provided by ProVerif. Operations in one phase will be inactive when the model moves into a new phase. All processes start in phase 0. The data requester obtains the owner's authorization and successfully access the data. When the user revokes access, the system transfers into "phase 1" and Sieve data blocks in the storage are updated with new keys. In "phase 2", the requester downloads the data blocks from storage but attempts to decrypt with the previously cached Sieve key. We verify that SucceedDecryptOldKey(dataBlockUUID) should be unreachable. As group size increases, we add more phases and pick one owner to revoke access at each phase. By the end of every round, the requester's cache of other owners' Sieve keys is still valid, thus we also verify that one owner's decision to revoke will not interfere with other users' keys.

Finally, we implement an injective query to ensure that the storage provider only applies to rekey tokens upon the data owner's request. This query led us to identify a bug in an earlier protocol draft that an attacker can replay rekey requests, rendering users' data inaccessible by anyone. This finding prompted us to add additional nonces to our protocol.

5.4 Modeling Group Ownership

To support group ownership, we use Shamir Secret Sharing to distribute data keys among co-owners. However, ProVerif currently lacks language support for this type of threshold encryption [62], particularly for encoding variable-sized sets of co-owners. To address this, we encode the size of the owner set statically in the model. All parameters of cryptographic operations must also be set statically, including the number of users and their positions. We have to create unique processes for every user in the group to handle different keys and internal states. As the group size grows, we have to expand these arguments and processes accordingly. Note that the implementation of different user processes is nearly identical, except for minor differences in user indices. Therefore, we developed a preprocessor language that automates the construction of static models with specified group sizes. We express the protocol flow with template functions and parameterized values. In this way, we can implement a common user process and, during compilation, expand this template into a variable number of concrete user processes.

6 IMPLEMENTATION

We provide an open-source repository[2] that includes the source code of TEO and the details of the security protocol modeling. We implement the core TEO protocol as a shared cross-platform library, `libteo`, with public-facing APIs. The library is written in 8945 lines of C++, excluding third-party libraries and evaluation tests. We use libsodium [59] as the main cryptography library, containing implementation for standard secret-key and public-key cryptography operations, with X25519 key exchange, XSalsa20 stream cipher encryption, Poly1306 MAC authentication, and Ed25519 signatures. In addition, we leverage the Crypto++ [24] library to implement Shamir Secret Sharing and the key splitting functionality for group ownership. Since the authors of Sieve [85] did not release their code, we re-implement Sieve operations using the Ed448-Goldilocks elliptic curve library [39] and consulted with

[2] https://github.com/synergylabs/TEO-release

them over email with our implementation details to ensure correctness. We use FlatBuffers [35] to serialize TEO's protocol message in a cross-platform format.

Our TEO prototype consists of client applications for multiple platforms. We developed a prototype Android app with support for the users' and admins' functionalities in 3350 lines of Java code. We use Android Beacon Library [65] for BLE scanning. It includes `libteo` as a native C++ library and uses Java Native Interface to execute API calls. We also implement test clients for different roles on x86 Linux desktops and popular single-board computers with ARM SoCs (Rasberry Pi 4 and Pi Zero W). Moreover, we develop a storage provider daemon as a key-value store for encrypted data contents using LevelDB [36] and with support for TEO revocation. In total, we implement these agents in 1206 lines of C++ in addition to the `libteo` library. TEO's protocol model contains 940 lines of ProVerif code with the group templates. After compilation, these models include 917, 1258, and 1599 lines of ProVerif code for group of size 1–3. We observed an exponential growth in verification time and memory consumption with larger group sizes. For example, on a 16-core machine with 64 GB RAM group size=1 took 3.62s, 258 MB memory while group size=3 took 17+ hours and consumed 50 GB. While we did not verify higher group sizes, our model generalizes to any group size.

7 EVALUATION

We evaluate TEO's design and our prototype implementation, with a suite of microbenchmarks and by integrating TEO with several real-world IoT device applications. Our evaluation results demonstrate that TEO introduces nominal communication and power consumption overhead over a baseline system without TEO's security primitives. One-time operations, such as device initiation and ownership claims, add an additional latency of up to 187 ms. Meanwhile, devices that continuously upload TEO encrypted data experience a performance overhead mostly dominated by the network communication speed: for larger files, TEO incurs 7–25% extra latency compared to a baseline of just uploading the same size data; for smaller files (10KB – 1MB), TEO's storage latency is 101-308 ms.

To characterize the overhead of TEO's primitives, we select representative IoT devices and client platforms, with different computational capabilities. We developed a TEO mobile app and installed it on Android phones (Nexus 5X), serving as TEO client agents for users and admins. For operations requiring human interaction (e.g., deciding whether to grant access or issue pre-auth tokens), we skip the user confirmation step to automate the tests so as to only measure the overhead of TEO's protocol communication and not the user reaction time. We chose off-the-shelf single-board computers, namely Raspberry Pi 4 (1.5 GHz 4-core, 4GB RAM,$35–$55) and Raspberry Pi Zero W (1 GHz single-core, 512MB RAM, $10), as TEO-endabled IoT devices. Both Android phones and IoT devices connect to our campus WiFi infrastructure as other devices in the building. On the other hand, we launched our prototype storage providers and agents for requesting data accesses on Linux machines (8-core, 16GB RAM) with wired connections to the same infrastructure.

Table 2: Average latency (in ms) for TEO operations, with a performance comparison of different IoT device hardware. We also measure battery usage for the TEO phone app (in μAh). Data access and revocation operations do not involve devices' participation.

Operation	User App Battery (μAh)	Average Latency ± Standard Deviation (ms)	
		RPi 4	RPi Zero
Initialize Device	20.18	44 ± 9	65 ± 35
Acquire Pre-Auth Token + Claim Device	34.43	187 ± 52	258 ± 128
Claim Device	21.19	67 ± 10	94 ± 31
Store Data, 1MB	43.03	308 ± 57	684 ± 155
Access Data, 1MB	22.25	170 ± 54	
Revocation and re-encrypt	25.07	62 ± 15	

7.1 Microbenchmarks

Latency. We first measure the overhead of each TEO operation in terms of the end-to-end latency from initiating the operation to the time it completes (Table 2). We repeat every operation 100 times and report the mean and standard deviation latency. Several operations such as device initialization (~44ms) and revocation (~62ms) are lightweight, while the initial claiming of the device has higher latency (~187ms). We analyze recurring TEO operations (data store) in further detail. Switching the device from Raspberry Pi 4 to Pi Zero, we observe modest slowdown (up to 2.2x for data store) but all operations finish within 65–258 ms. This is understandable since they have drastically different compute capabilities. Overall, most TEO operations are only needed once or very infrequently, so a ≤187 ms latency increase has a modest impact on end users.

Phone Battery Impact. Table 2 reports the battery consumption of TEO operations involving the user's phone app, as the average μAh over 100 iterations. We currently use the battery levels reported by Android since they seemed sufficient for our use case [5], leaving more precise energy measurements for future work [57, 58]. Our test Nexus 5x has a rated battery capacity of 2700 mAh. Operations such as providing Sieve key share for the device to store data (43.03 μAh) once a minute consumes just 2.2% of the battery life over a 24 hour period. Proximity detection (Section 4.3) for group membership supported by BLE scans also affects battery life depending on the frequency. We measure the battery drain speed over a period of 5 minutes and calculate the difference when the phone is idle. Continuous scanning quickly drains the battery at a speed of 2090 μAh per minute. However, a simple optimization (increasing the BLE scan interval to once every 10 seconds) reduces the drain to 66.2 μAh per minute, consuming 3.5% of the battery over a 24 hour period. With additional optimizations (iBeacon-based BLE entry/exit detection, reducing scan intervals), the energy impact of proximity detection can be reduced further.

Data Store and Sizes. The latency overhead of recurring operations, such as encrypting the data on the device and then transmitting it to be stored on a storage provider, depends on the size of the data, as reported in Table 3. We omit other overheads in the

Table 3: Data store operation overhead breakdown for Raspberry Pi 4, reported as mean values in *ms*.

Data Size	Data Encryption	Data Upload	Total Time (vs. Upload Time)
10KB	< 1	19	101 (5x)
100KB	2	29	116 (4x)
1MB	25	127	308 (2.42x)
10MB	168	1429	1791 (1.25x)
100MB	1577	15256	16293 (1.07x)

Table 4: Average latency and standard deviation for storing 1MB data for different group sizes. We emulate multiple owners as different processes on a PC and have the device repeatedly store data 100 times. We also include a single user running a TEO phone agent.

Group Size	Phone	Emulation			
	1	1	5	25	50
Average Latency (ms)	308 ±57	234 ±156	316 ±30	634 ±58	1085 ±85

breakdown table since they do not scale significantly with data sizes. For example, Sieve data blocks (containing the data keys) are independent of the size of the data in this experiment and have the same size since we use a single key to encrypt the data.

As the data size increases, the overheads associated with data encryption and upload scale proportionally, dominating the latency for using TEO for large data sizes. For example, the total latency is just 1.25x compared to the time spent on uploading the 10MB files (since TEO's symmetric encryption produces ciphertext the same length as the plaintext). For larger files (100MB), the relative latency of TEO decreases further to 1.07x upload time, showing that TEO's overhead amortizes as the data size increases. For smaller data sizes (e.g. 10KB - 1MB), TEO's protocol overhead is still relatively small ($\leq$ 308 ms) given the asynchronous nature of data storage operations.

To help reduce TEO overhead for large files, we implemented an optimization to *pipeline* data encryption and upload. We split large data into fixed chunks (1MB by default) and start uploading them as soon as the encryption of that chunk completes. Therefore, for large files such as 100MB, the sum of encryption and upload time exceeds the total elapsed time.

Group Ownership. We measure the performance impact of variable group sizes on the device's data store operations. We emulate a large number of users in a group using a standalone Linux desktop with a wired connection to the campus network. Each user in this scenario is a separate process on this desktop. The device still performs normal TEO operations, but it needs to communicate with all users. Table 4 shows the average latency for such operations for groups of up to 50 members. To provide a comparison between our emulated users forming a group and a real phone client, we also include the latency for a single user on an Android phone (first column). We observe a small performance discrepancy between the emulation platform (Linux) and the real Android phones (308 ms vs. 234 ms). As the size of the group increases, the main cause of the additional latency is the resource contention on the device. For every owner, the IoT device spins off a new thread to encrypt their key share with Sieve and upload the Sieve data block to the storage. Because the Raspberry Pi 4 only has 4 cores, threads for different owners cause CPU contention. Even with a large group size of 50 users, the slowdown is just 5x compared to the single-owner case.

Table 5: Total changes required (lines of code) to integrate existing applications with TEO. These changes mostly focus on redirecting data storage to the co-located TEO device driver program. See more details in Section 7.2.

Applications	Motion [63]	Mycroft [64]	Doorlock [38]
Language	C++/Python	Python	Node.js
Lines Changed	31	73	121

7.2 Case Studies

We integrate three real-world smart IoT applications into TEO-enabled devices (Raspberry Pi 4). We searched for popular open-source smart apps on GitHub and tutorial websites and finalized one for each of the interesting categories. In all three cases, we extend the original apps with new functionality using TEO operations and primitives. Table 5 shows the total changes in terms of lines of code we made to each application. In general, the integration process incurs minimal changes. We develop a TEO driver (as part of our TEO prototype) that manages the TEO runtime on the device. It opens API interfaces as REST endpoints exposed only to *localhost* so that the application can leverage the driver to store data and verify command certificates.

Motion Camera. Motion [63] is a smart camera app that records video clips whenever it detects motions. The users can later review these recorded events. To preserve privacy, all data are saved locally. We edit Motion's configuration file and implement a post-recording *hook* program that uses TEO to encrypt and store the video recording. This integration not only protects user data but also increases the limited local storage space.

For evaluation, we set the length of event recording to be 1 minute, repeatedly triggering event detection over 100 times to measure runtime latency. We set the group size to be a single user. On average, the 1-minute clip is around 22MB and the TEO driver takes around 2879 ms to process the storage request. These performance numbers are consistent with our microbenchmark results (Table 3). Since it only takes $\sim$ 3 seconds to store a one-minute long video recording, we believe that TEO integration would be a useful and practical extension for the Motion app.

Speaker with Voice Assistant. Mycroft AI [64] is a smart speaker app similar to Amazon Echo and Google Home. Users trigger it with a "wake" word, followed by their instructions. A critical privacy concern with smart speakers is that they can record user interactions and upload audio clips to train better machine learning models and for internal analysis [19, 20]. We extend Mycroft so that users' private audio recording data are protected with TEO encryption and access control. Every time a wake word is detected, Mycroft starts recording the following user commands and uploads them to a TEO storage provider. Compared to video data, audio clips are

much smaller and highlight the extra overhead in TEO communication. On average, the audio clips are around 72KB, and it takes the app 235 ms to finish storing these clips. This result is slightly higher than our microbenchmark, highlighting that applications storing smaller pieces of data are more sensitive to TEO's overhead and cross-app communications (between Mycroft and the TEO driver). However, we consider this overhead still acceptable since storing clips is done asynchronously and does not block the users' normal interaction with the device.

Smart Doorlock. Complementary to the previous two cases, we implement a smart doorlock application that shows how TEO can protect owners in real time and prevent unauthorized control of the device from non-stakeholders. We develop this app based on an open source door lock project [38] and utilize the Blynk IoT Library [17]. With TEO integration, the app can check for access authorizations accompanying every incoming command, and only executes valid commands from current owners. Our evaluation measures an average increase of 11 ms in latency due to certificate check on the TEO runtime. Low overhead is important in this case, since the process is now on the critical path of the device's functionality and user interactions.

8 DISCUSSION AND LIMITATIONS

Support for Less Capable, Lower-Power Devices. Currently, our TEO prototype supports mobile phones, Linux machines, and Raspberry Pis. These platforms are equipped with modern ARM or x86 processors. Unfortunately, porting TEO to other low-power devices (e.g. using the popular ESP32 series or the ARM Cortex M3 series) is more challenging for two reasons. First, TEO makes extensive use of several heavy-weight cryptography libraries, which are not yet supported by microcontroller-based architectures in these devices. Second, assuming that all of TEO's dependencies have been ported, TEO could encounter high performance overhead due to limited processing power and resources (e.g., ESP8266 only has 160Mhz CPU and 50KB memory [30]). It would be an interesting future research direction to extend TEO to these lower-power class of devices and, indeed, recent works have proposed novel cryptography protocols to enable public-key cryptography on them [2].

Reducing the Trusted Computing Base. TEO assumes that IoT devices and user mobile phones are trusted to protect user data. Compromised devices can bypass TEO and directly leak users' private data, or impersonate other devices through Cuckoo attacks by relaying network traffic [71]. Malware-infected phones can steal credentials to users' data. Furthermore, TEO assumes that the cloud storage services correctly perform the computation for re-encryption to work. To reduce the trusted computing base, one promising future direction is to expand TEO with a secure hardware infrastructure. For example, we can perform security critical operations inside the Trusted Execution Environment (e.g., Intel SGX, ARM TrustZone) as inspired by many prior works [8, 61, 74, 82, 94] and leverage Trusted Platform Modules and remote attestations [3, 45, 52, 66, 72] to ensure the integrity of TEO programs and operating systems.

Specifically, to mitigate the threats of compromised smart devices, we can borrow insights from many recent works and develop a trusted TEO hub. The hub can act as a network access point for local devices and require all egress network traffic to be encrypted with TEO [29, 43, 83, 93], or redesign the IoT application programming architecture and have the trusted hub to process all user's private data [47, 49, 90].

Deployment Challenges. There are some practical challenges with large-scale use of TEO. In addition to partial availability (Section 4.6), identity management can be complex. Admins need to associate a user's public key with their real identity. This could be facilitated by conventional PKIs, third-party services like Keybase [48] or a trusted mediator (e.g., Airbnb holds public keys and identities for hosts and guests). Moreover, storage providers should be compensated since they will host all encrypted data with high availability. To encourage competitive pricing and avoid vendor lock-in, TEO's design does not require strong trust in the storage provider, so this role can be filled by many entities (e.g., building manager self-hosts, public cloud services, or centralized servers in Airbnb). We also provide a reference storage provider implementation in our TEO prototype that uses a simple key value store database.

Monitoring Device Ownership. To ensure their data are always protected by TEO, users should keep monitoring the list of devices they currently own. Otherwise, they might mistakenly think they still are ephemeral owners when the device is re-claimed by someone else. We envision extending TEO mobile apps with monitoring functionality for future deployment to alleviate user burdens. The app can send notifications when the user loses device ownership to help them stay informed. It can also analyze the latest data stored by the device to verify that the user is one of the owners (since metadata is publicly accessible).

9 CONCLUSION

In this paper, we identify an emerging challenge in smart device deployments – mismatched device and data ownerships during ownership. To protect all stakeholders' security and control over their data, we propose TEO – IoT Ephemeral Ownership – and design a complete protocol specification to achieve this idea for smart devices. We conduct formal security analysis and prove the correctness and security of TEO's protocol design. Finally, we implement a prototype of TEO and integrate several real-world smart devices on top of it to demonstrate its practicality and low performance overhead in real-world settings.

ACKNOWLEDGEMENTS

We thank Haojian Jin, Frank Wang, and Bruno Blanchet for their helpful correspondences and valuable insights influencing the design of TEO's core components. Furthermore, we greatly appreciate the anonymous reviewers, CMU colleagues (Bryan Parno, Wenting Zheng, Eunsuk Kang, Travis Hance, and Christopher Canel), and our shepherd, Diego Perino, for their feedback and comments on our paper drafts. This work was partially supported by NSF Awards CNS-1704542, CNS-1943016, TWC-1564009 and SaTC-1801472, ONR Award N000141812618, and the Carnegie Mellon CyLab Security and Privacy Institute.

REFERENCES

[1] Paarijaat Aditya, Rijurekha Sen, Peter Druschel, Seong Joon Oh, Rodrigo Benenson, Mario Fritz, Bernt Schiele, Bobby Bhattacharjee, and Tong Tong Wu. 2016. I-pic: A platform for privacy-compliant image capture. In *Proceedings of the 14th annual international conference on mobile systems, applications, and services*. 235–248.

[2] Fatemah Alharbi, Arwa Alrawais, Abdulrahman Bin Rabiah, Silas Richelson, and Nael Abu-Ghazaleh. 2021. CSProp: Ciphertext and Signature Propagation Low-Overhead Public-Key Cryptosystem for IoT Environments. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*. 609–626.

[3] Mahmoud Ammar, Bruno Crispo, and Gene Tsudik. 2020. SIMPLE: A remote attestation approach for resource-constrained IoT devices. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 247–258.

[4] Michael P Andersen, Sam Kumar, Moustafa AbdelBaky, Gabe Fierro, John Kolb, Hyung-Sin Kim, David E Culler, and Raluca Ada Popa. 2019. WAVE: A Decentralized Authorization Framework with Transitive Delegation. In *28th USENIX Security Symposium (USENIX Security 19)*. 1375–1392.

[5] Android. 2021. BatteryManager. https://developer.android.com/reference/android/os/BatteryManager.

[6] Andrew W Appel and Edward W Felten. 1999. Proof-carrying authentication. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*. 52–62.

[7] Apple. 2021. Bonjour. https://developer.apple.com/bonjour/.

[8] Gbadebo Ayoade, Vishal Karande, Latifur Khan, and Kevin Hamlen. 2018. Decentralized IoT data management using blockchain and trusted execution environment. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*. IEEE, 15–22.

[9] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. 2021. SoK: Computer-Aided Cryptography. In *IEEE Symposium on Security and Privacy*.

[10] Lujo Bauer, Scott Garriss, Jonathan M McCune, Michael K Reiter, Jason Rouse, and Peter Rutenbar. 2005. Device-enabled authorization in the Grey system. In *International Conference on Information Security*. Springer, 431–445.

[11] Julia Bernd, Ruba Abu-Salma, and Alisa Frik. 2020. Bystanders' Privacy: The Perspectives of Nannies on Smart Home Surveillance. In *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*.

[12] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. 2015. A messy state of the union: Taming the composite state machines of TLS. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 535–552.

[13] Arnar Birgisson, Joe Gibbs Politz, Úlfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner. 2014. Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud. In *Network and Distributed System Security Symposium*.

[14] Bruno Blanchet. 2012. Security protocol verification: Symbolic and computational models. In *International Conference on Principles of Security and Trust*. Springer, 3–29.

[15] Bruno Blanchet. 2016. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. *Found. Trends Priv. Secur.* (2016).

[16] Bruno Blanchet. 2021. ProVerif users. https://prosecco.gforge.inria.fr/personal/bblanche/proverif/proverif-users.html.

[17] Blynk.io. 2022. Blynk Library. https://github.com/blynkkk/blynk-library.

[18] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. 2013. Key homomorphic PRFs and their applications. In *Annual Cryptology Conference*. Springer, 410–428.

[19] CNBC. 2021. Amazon Alexa records you every time you ask it something — here's how to delete those recordings. https://www.cnbc.com/2021/02/18/how-to-delete-amazon-alexa-recordings-for-privacy.html.

[20] CNET. 2021. Amazon's Astro may be cute, but security experts warn of privacy concerns. https://www.cnet.com/tech/amazons-astro-may-be-cute-but-security-experts-warn-of-privacy-concerns/.

[21] Camille Cobb, Sruti Bhagavatula, Kalil Anderson Garrett, Alison Hoffman, Varun Rao, and Lujo Bauer. 2021. "I would have to evaluate their objections": Privacy tensions between smart home device owners and incidental users. *Proceedings on Privacy Enhancing Technologies* 4 (2021), 54–75.

[22] Jessica Colnago, Yuanyuan Feng, Tharangini Palanivel, Sarah Pearman, Megan Ung, Alessandro Acquisti, Lorrie Faith Cranor, and Norman Sadeh. 2020. Informing the design of a personalized privacy assistant for the internet of things. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*.

[23] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. 2017. A comprehensive symbolic analysis of TLS 1.3. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1773–1788.

[24] Crypto++. 2021. Crypto++. https://www.cryptopp.com/.

[25] Rajib Dey, Sayma Sultana, Afsaneh Razi, and Pamela J Wisniewski. 2020. Exploring Smart Home Device Use by Airbnb Hosts. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–8.

[26] Danny Dolev and Andrew Yao. 1983. On the security of public key protocols. *IEEE Transactions on information theory* 29, 2 (1983), 198–208.

[27] Chethana Dukkipati, Yunpeng Zhang, and Liang Chieh Cheng. 2018. Decentralized, blockchain based access control framework for the heterogeneous internet of things. In *Proceedings of the Third ACM Workshop on Attribute-Based Access Control*. 61–69.

[28] Pardis Emami-Naeini, Janarth Dheenadhayalan, Yuvraj Agarwal, and Lorrie Faith Cranor. 2021. Which Privacy and Security Attributes Most Impact Consumers' Risk Perception and Willingness to Purchase IoT Devices?. In *2021 IEEE Symposium on Security and Privacy (SP)*. 1937–1954.

[29] Jeremy Erickson, Qi Alfred Chen, Xiaochen Yu, Erinjen Lin, Robert Levy, and Z Morley Mao. 2018. No one in the middle: Enabling network access control via transparent attribution. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. 651–658.

[30] Espressif. 2020. ESP8266 Datasheet. https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.

[31] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. 2016. FlowFence: Practical Data Protection for Emerging IoT Application Frameworks. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 531–548. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/fernandes

[32] Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, and Rajesh K Gupta. 2020. ACES: Automatic configuration of energy harvesting sensors with reinforcement learning. *ACM Transactions on Sensor Networks (TOSN)* 16, 4 (2020), 1–31.

[33] Diana Freed, Sam Havron, Emily Tseng, Andrea Gallardo, Rahul Chatterjee, Thomas Ristenpart, and Nicola Dell. 2019. " Is my phone hacked?" Analyzing Clinical Computer Security Interventions with Survivors of Intimate Partner Violence. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–24.

[34] Diana Freed, Jackeline Palmer, Diana Elizabeth Minchala, Karen Levy, Thomas Ristenpart, and Nicola Dell. 2017. Digital technologies and intimate partner violence: A qualitative analysis with multiple stakeholders. *Proceedings of the ACM on Human-Computer Interaction* 1, CSCW (2017), 1–22.

[35] Google. 2021. FlatBuffers. https://google.github.io/flatbuffers/.

[36] Google. 2021. LevelDB. https://github.com/google/leveldb.

[37] Saikat Guha, Mudit Jain, and Venkata N Padmanabhan. 2012. Koi: A location-privacy platform for smartphone apps. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 183–196.

[38] Hacker Shack. 2017. Smartphone Connected Home Door Lock. https://www.hackster.io/hackershack/smartphone-connected-home-door-lock-69944f.

[39] Mike Hamburg. 2021. Ed448-Goldilocks. https://sourceforge.net/p/ed448goldilocks/wiki/Home/.

[40] Sam Havron, Diana Freed, Rahul Chatterjee, Damon McCoy, Nicola Dell, and Thomas Ristenpart. 2019. Clinical computer security for victims of intimate partner violence. In *28th USENIX Security Symposium (USENIX Security 19)*.

[41] Weijia He, Maximilian Golla, Roshni Padhi, Jordan Ofek, Markus Dürmuth, Earlence Fernandes, and Blase Ur. 2018. Rethinking access control and authentication for the home internet of things (IoT). In *27th USENIX Security Symposium (USENIX Security 18)*.

[42] Weijia He, Valerie Zhao, Olivia Morkved, Sabeeka Siddiqui, Earlence Fernandes, Josiah D. Hester, and Blase Ur. 2021. SoK: Context Sensing for Access Control in the Adversarial Home IoT. In *Proceedings of the 6th IEEE European Symposium on Security and Privacy*.

[43] James Hong, Amit Levy, Laurynas Riliskis, and Philip Levis. 2018. Don't Talk Unless I Say So! Securing the Internet of Things with Default-Off Networking. In *3rd ACM/IEEE International Conference on Internet-of-Things Design and Implementation (IoTDI)*.

[44] Qinlong Huang, Yixian Yang, and Licheng Wang. 2017. Secure Data Access Control With Ciphertext Update and Computation Outsourcing in Fog Computing for Internet of Things. *IEEE Access* 5 (2017), 12941–12950. https://doi.org/10.1109/ACCESS.2017.2727054

[45] Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Gene Tsudik. 2018. Us-aid: Unattended scalable attestation of iot devices. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 21–30.

[46] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Z. Morley Mao, and Atul Prakash. 2017. ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms. In *21st Network and Distributed Security Symposium*.

[47] Haojian Jin, Gram Liu, David Hwang, Swarun Kumar, Yuvraj Agarwal, and Jason I Hong. 2022. Peekaboo: A Hub-Based Approach to Enable Transparency in Data Processing within Smart Homes. In *2022 IEEE Symposium on Security and Privacy (SP). IEEE*.

[48] Keybase. 2021. Keybase. https://keybase.io/.

[49] Dohyun Kim, Prasoon Patidar, Han Zhang, Abhijith Anilkumar, and Yuvraj Agarwal. 2022. Self-Serviced IoT: Practical and Private IoT Computation Offloading with Full User Control. *arXiv preprint arXiv:2205.04405* (2022).

[50] Hokeun Kim, Eunsuk Kang, Edward A. Lee, and David Broman. 2017. A Toolkit for Construction of Authorization Service Infrastructure for the Internet of Things.

In *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*. 147–158.
[51] Wonjung Kim, Seungchul Lee, Youngjae Chang, Taegyeong Lee, Inseok Hwang, and Junehwa Song. 2021. Hivemind: social control-and-use of IoT towards democratization of public spaces. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. 467–482.
[52] Boyu Kuang, Anmin Fu, Shui Yu, Guomin Yang, Mang Su, and Yuqing Zhang. 2019. ESDRA: An efficient and secure distributed remote attestation scheme for IoT swarms. *IEEE Internet of Things Journal* 6, 5 (2019), 8372–8383.
[53] Sam Kumar, Yuncong Hu, Michael P Andersen, Raluca Ada Popa, and David E. Culler. 2019. JEDI: Many-to-Many End-to-End Encryption and Key Delegation for IoT. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 1519–1536. https://www.usenix.org/conference/usenixsecurity19/presentation/kumar-sam
[54] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. 1992. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems (TOCS)* 10, 4 (1992), 265–310.
[55] Gierad Laput, Yang Zhang, and Chris Harrison. 2017. Synthetic sensors: Towards general-purpose sensing. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 3986–3999.
[56] Tam Le and Matt W Mutka. 2019. Access control with delegation for smart home applications. In *Proceedings of the International Conference on Internet of Things Design and Implementation*. 142–147.
[57] Ding Li, Shuai Hao, Jiaping Gui, and William GJ Halfond. 2014. An empirical study of the energy consumption of android applications. In *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 121–130.
[58] Ding Li, Shuai Hao, William GJ Halfond, and Ramesh Govindan. 2013. Calculating source line level energy information for android applications. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. 78–89.
[59] libsodium. 2021. A modern, portable, easy to use crypto library. https://libsodium.org/.
[60] Shrirang Mare, Franziska Roesner, and Tadayoshi Kohno. 2020. Smart Devices in Airbnbs: Considering Privacy and Security for both Guests and Hosts. *Proc. Priv. Enhancing Technol.* 2020, 2 (2020), 436–458.
[61] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. 94–108.
[62] Murat Moran and Dan S Wallach. 2017. Verification of STAR-Vote and Evaluation of FDR and ProVerif. In *International Conference on Integrated Formal Methods*. Springer, 422–436.
[63] Motion. 2021. Motion. https://motion-project.github.io/.
[64] Mycroft. 2021. Mycroft – The Open Source Privacy-Focused Voice Assistant. https://mycroft.ai/.
[65] Radius Network. 2021. Android Beacon Library. https://altbeacon.github.io/android-beacon-library/index.html.
[66] Ivan De Oliveira Nunes, Karim Eldefrawy, Norrathep Rattanavipanon, Michael Steiner, and Gene Tsudik. 2019. {VRASED}: A Verified {Hardware/Software} {Co-Design} for Remote Attestation. In *28th USENIX Security Symposium (USENIX Security 19)*. 1429–1446.
[67] Open Connectivity Foundation. 2021. UPnP Standards and Architecture. https://openconnectivity.org/developer/specifications/upnp-resources/upnp.
[68] Nouha Oualha and Kim Thuat Nguyen. 2016. Lightweight Attribute-Based Encryption for the Internet of Things. In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*. 1–6. https://doi.org/10.1109/ICCCN.2016.7568538
[69] Shijia Pan, Carlos Ruiz, Jun Han, Adeola Bannis, Patrick Tague, Hae Young Noh, and Pei Zhang. 2018. Universense: IoT device pairing through heterogeneous sensing signals. In *Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications*. 55–60.
[70] Simon Parkin, Trupti Patel, Isabel Lopez-Neira, and Leonie Tanczer. 2019. Usability analysis of shared device ecosystem security: Informing support for survivors of IoT-facilitated tech-abuse. In *Proceedings of the New Security Paradigms Workshop*.
[71] Bryan Parno. 2008. Bootstrapping Trust in a" Trusted" Platform.. In *HotSec*.
[72] Bryan Parno, Jonathan M McCune, and Adrian Perrig. 2010. Bootstrapping trust in commodity computers. In *2010 IEEE Symposium on Security and Privacy*. IEEE, 414–429.
[73] Otto Julio Ahlert Pinno, Andre Ricardo Abed Gregio, and Luis C. E. De Bona. 2017. ControlChain: Blockchain as a Central Enabler for Access Control Authorizations in the IoT. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. 1–6. https://doi.org/10.1109/GLOCOM.2017.8254521
[74] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *2015 IEEE symposium on security and privacy*. IEEE, 38–54.
[75] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. 2018. Situational access control in the internet of things. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1056–1073.
[76] Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquennoy. 2017. Towards blockchain-based auditable storage and sharing of IoT data. In *Proceedings of the 2017 on Cloud Computing Security Workshop*. 45–50.
[77] Hossein Shafagh, Lukas Burkhalter, Sylvia Ratnasamy, and Anwar Hithnawi. 2020. Droplet: Decentralized Authorization and Access Control for Encrypted Data Streams. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 2469–2486.
[78] Hossein Shafagh, Anwar Hithnawi, Lukas Burkhalter, Pascal Fischli, and Simon Duquennoy. 2017. Secure sharing of partially homomorphic encrypted IoT data. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. 1–14.
[79] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.
[80] Amit Kumar Sikder, Leonardo Babun, Z Berkay Celik, Abbas Acar, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A Selcuk Uluagac. 2020. Kratos: multi-user multi-device-aware access control system for the smart home. In *Prelavantroceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*.
[81] Thread Group. 2021. Thread. https://www.threadgroup.org/.
[82] Florian Tramer and Dan Boneh. 2018. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287* (2018).
[83] Rahmadi Trimananda, Ali Younis, Bojun Wang, Bin Xu, Brian Demsky, and Guoqing Xu. 2018. Vigilia: Securing Smart Home Edge Computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*.
[84] Emily Tseng, Rosanna Bellini, Nora McDonald, Matan Danos, Rachel Greenstadt, Damon McCoy, Nicola Dell, and Thomas Ristenpart. 2020. The Tools and Tactics Used in Intimate Partner Surveillance: An Analysis of Online Infidelity Forums. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 1893–1909.
[85] Frank Wang, James Mickens, Nickolai Zeldovich, and Vinod Vaikuntanathan. 2016. Sieve: Cryptographically enforced access control for user data in untrusted clouds. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 611–626.
[86] Qinying Wang, Shouling Ji, Yuan Tian, Xuhong Zhang, Binbin Zhao, Yuhong Kan, Zhaowei Lin, Changting Lin, Shuiguang Deng, Alex X Liu, et al. 2021. MPInspector: a systematic and automatic approach for evaluating the security of IoT messaging protocols. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*. 4205–4222.
[87] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Dave Jing Tian, Antonio Bianchi, Mathias Payer, and Dongyan Xu. 2020. {BLESA}: Spoofing Attacks against Reconnections in Bluetooth Low Energy. In *14th {USENIX} Workshop on Offensive Technologies ({WOOT} 20)*.
[88] Yaxing Yao, Justin Reed Basdeo, Oriana Rosata Mcdonough, and Yang Wang. 2019. Privacy perceptions and designs of bystanders in smart homes. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–24.
[89] Bin Yuan, Yan Jia, Luyi Xing, Dongfang Zhao, XiaoFeng Wang, and Yuqing Zhang. 2020. Shattered Chain of Trust: Understanding Security Risks in Cross-Cloud IoT Access Delegation. In *29th USENIX Security Symposium (USENIX Security 20)*. 1183–1200.
[90] Gina Yuan, David Mazières, and Matei Zaharia. 2022. Extricating IoT Devices from Vendor Infrastructure with Karl. *arXiv preprint arXiv:2204.13737* (2022).
[91] Eric Zeng and Franziska Roesner. 2019. Understanding and improving security and privacy in multi-user smart homes: a design exploration and in-home user study. In *28th USENIX Security Symposium (USENIX Security 19)*.
[92] Zeroconf. 2021. Zero Configuration Networking. http://www.zeroconf.org/.
[93] Han Zhang, Abhijith Anilkumar, Matt Fredrikson, and Yuvraj Agarwal. 2021. Capture: Centralized Library Management for Heterogeneous IoT Devices. In *USENIX Security Symposium*.
[94] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2017. Opaque: An oblivious and encrypted distributed analytics platform. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 283–298.