Secure Multi-User Content Sharing for Augmented Reality Applications

Kimberly Ruth, Tadayoshi Kohno, and Franziska Roesner, ***University of Washington***

https://www.usenix.org/conference/usenixsecurity19/presentation/ruth

This paper is included in the Proceedings of the 28th USENIX Security Symposium.

August 14–16, 2019 • Santa Clara, CA, USA

978-1-939133-06-9

Open access to the Proceedings of the 28th USENIX Security Symposium is sponsored by USENIX.

Secure Multi-User Content Sharing for Augmented Reality Applications

Kimberly Ruth
kcr32@cs.washington.edu

Tadayoshi Kohno
yoshi@cs.washington.edu

Franziska Roesner
franzi@cs.washington.edu

Paul G. Allen School of Computer Science & Engineering, University of Washington

https://ar-sec.cs.washington.edu/

Abstract

Augmented reality (AR), which overlays virtual content on top of the user's perception of the real world, has now begun to enter the consumer market. Besides smartphone platforms, early-stage head-mounted displays such as the Microsoft HoloLens are under active development. Many compelling uses of these technologies are multi-user: e.g., in-person collaborative tools, multiplayer gaming, and telepresence. While prior work on AR security and privacy has studied potential risks from AR applications, new risks will also arise among multiple human users. In this work, we explore the challenges that arise in designing secure and private content sharing for multi-user AR. We analyze representative application case studies and systematize design goals for security and functionality that a multi-user AR platform should support. We design an AR content sharing control module that achieves these goals and build a prototype implementation (ShareAR) for the HoloLens. This work builds foundations for secure and private multi-user AR interactions.

1 Introduction

Augmented reality (AR) technologies, which overlay digitally generated content on a user's view of the physical world, are now becoming commercially available. AR smartphone applications like Pokemon Go and Snapchat, as well as smartphone-based AR platforms from Apple [5], Facebook [6], and Google [4], are already available to billions of consumers. More sophisticated AR headsets are also available in developer or beta editions from companies like Magic Leap [37], Meta [41], and Microsoft [24]. The AR market is growing rapidly, with a market value projected to reach $90 billion by 2022 [15].

The power that AR technologies have to shape users' perceptions of reality — and integrate virtual objects with the physical world — also brings security and privacy risks and challenges. It is important to address these risks early, while AR is still under active development, to achieve more robust security and privacy than would be possible once systemic issues have become entrenched in mainstream technologies. The computer security and privacy community has already taken steps towards identifying and mitigating potential risks from malicious or buggy AR apps. These efforts — e.g., limiting untrusted apps' access to sensor data [28, 49, 54] or restricting the virtual content apps can display [32, 34] — are reminiscent of recent work on access control for untrusted apps on other platforms, such as smartphones [16, 53]. Despite this valuable initial progress, we observe a critical gap in prior work on security and privacy for AR: though past efforts are valuable for protecting *individual* users from untrusted *applications*, prior work has not considered how to address potentially undesirable interactions between *multiple human users* of an AR app or ecosystem.

The need to consider security for multi-user AR. Despite this gap in prior work, we observe that many compelling use cases for AR will involve multiple users, each with their own AR device, who may be physically co-located or collaborating remotely and who may be interacting with shared virtual objects: for instance, in-person collaborative tools [63], multi-player gaming [3], and telepresence [18]. As one concrete example already available to AR users, Ubiquity6 has released a beta version of its smartphone platform in which all users can view and interact with all AR content within the app [67], as shown in Figure 1.

In these contexts, the potential security, privacy, and safety risks for AR users come not only from the apps on their own devices but also from *other users*. For example, one user of a shared AR app might accidentally or intentionally spam other users with annoying or even disturbing virtual objects, or manipulate another person's virtual object (e.g., artwork) without permission. Indeed, even though multi-user AR technologies are not yet ubiquitous in the consumer market, precursors of such issues have already begun to emerge in the wild and in research settings today. In AR specifically, for example, there have been reports of "vandalism" of AR art in Snapchat [38], and a recent study found that pairs of AR users often positioned virtual objects in each other's personal space [35]. Similar findings have been made in virtual reality (VR), where users have blocked each other's vision

Figure 1: Sample screenshots from Ubiquity6 multi-user application (taken from [67]). Users can, for instance feed a virtual cat (left) or tend a virtual garden (right).

with virtual objects [66] and invaded each other's personal space [1]. In earlier work on digital tabletop displays, researchers observed conflicts between users closing or stealing each others' documents [44]. As a final example, Apple's AirDrop scheme for sharing files between physically co-located Apple devices has been misused to send inappropriate content to strangers in public spaces [11].

Thus, we can and should expect conflicts and tensions to arise between multiple AR users, which raises the critical question: how should AR platform and app designers handle these issues? Existing AR platforms provide limited or no support to app developers on this front. For example, though HoloLens supports running an app shared between multiple device users, it surfaces only basic cross-device messaging features, providing no framework for developers to reason about or support complex multi-user interactions.

This work: Sharing control for multi-user AR. In this work, we thus address the challenge of *providing secure and private content sharing capabilities for multi-user augmented reality applications*. Unlike prior AR security work that focused on protecting users from untrusted apps, we aim to limit undesirable interactions between mutually distrusting human users — similar to how traditional file system permissions attempt to separate mutually distrusting human users from each other. By addressing this issue before multi-user AR becomes widespread, we aim to inform the design of future AR systems, thereby preventing such multi-user concerns from manifesting broadly.

In our exploration of this space, however, we find that controlled sharing for AR content raises unique challenges not present in traditional settings such as file systems or shared online documents. The root cause of this complexity is AR's integration with the physical world. Because people share the same physical world, they may have certain expectations about how AR content is shared. Indeed, prior work has found that users often expect co-located users to see the same virtual content [35]. For example, a user might want to have control of their personal physical space, not allowing another user to place too many virtual objects near them. Fulfilling this request requires that either the second user is restricted in augmenting his or her own view of the world, or that the two users see different things. Diverging views of the world can violate expectations, however: consider watching or interacting with an AR object that only you can see while another AR user unknowingly steps into the space between you and your object.

AR's integration with the physical world further complicates many access control design decisions. Consider the seemingly simple notion of Alice sharing an AR object with Bob: for instance, giving him read access to a piece of virtual artwork. When this object is shared, does Bob see the artwork in the same place as Alice (e.g., on a particular wall), or does Bob see his own instance of the object in a different place? The answer may depend on the semantics of the app and whether Alice and Bob are physically co-located, and the answer interacts with many other design choices.

In our work, we thus explore a set of multi-user AR case study apps that represent different points in the possible design space (co-located and remote users, opt-in versus opt-out sharing) to surface functionality and security goals for an AR sharing control module. We then present the design of such a module, which we envision as an app-level library or OS interface that can be leveraged by AR application developers. This module supports app developers in (1) allowing users to *share* AR content with other (co-located or remote) users, (2) allowing users to *control* both inbound and outbound AR content, while (3) addressing fundamental challenges raised by AR's integration with the physical world.

One key challenge is to define and manage different ways that AR content might be mapped into the physical world — we do so by supporting both location-coupled objects (which all users see in the same physical place) and location-decoupled objects (for which users see their own copies in separate locations), and by managing the resulting impact of these types of objects on sharing and access control functionality. Another key challenge is to address potential discontinuities in user expectations around private content in a shared physical world — we do so by introducing "ghost" objects that allow users to share with other AR users *that* they are interacting with a virtual object without sharing sensitive details about that object. Finally, a third key challenge is to respect users' possible sense of ownership of physical space (e.g., personal space) — to that end, our design supports policies for how to handle AR objects that are within a protected region (e.g., making objects closer to a user more transparent to that user). Through our exploration, we find that no single sharing control model will work for all apps, but that our proposed module can provide key features to support app developers in creating multi-user AR apps that meet users' expectations.

Contributions. In summary, our contributions include:

1. We are the first to rigorously explore the design space for secure and private AR content sharing between users. Through an exploration of multi-user AR case study apps, we identify (in Section 2) key design goals, challenges, and features that app developers require to support secure and private multi-user AR experiences.
2. Building on our design space exploration (Section 2), we present the design (in Section 4) of a multi-user AR sharing control module. Our design addresses key challenges and enables app developers to meet our design goals: supporting users in controlling how they share AR content with others and how AR content is shared with them, while taking into account the ways in which AR content might integrate with the physical world.
3. We provide a concrete prototype implementation (ShareAR, in Section 5) and evaluation (in Section 6), iteratively refining our design and demonstrating its feasibility in practice. Our source code will be made available at the project website.[1]

This work lays a foundation for future secure and private multi-user AR apps. Mitigating undesirable interactions between users can facilitate user adoption of AR and help the technology reach its full potential.

2 Problem Formulation and Design Goals

We begin by formulating, for the first time, the problem space and goals for secure and private multi-user AR content sharing. To do so systematically, we consider four case study apps (Section 2.1) that we selected to explore unique points in the multi-user AR design space and that we envisioned might exercise a broad range of functionality and security needs. From these case studies, we then derive our security and functionality goals (Sections 2.2 and 2.3).

In exploring possible apps, we observe that the key aspect of AR that differentiates it from previous technologies is its tight *physical-world integration*: virtual content appears to the user to be situated in 3D space among physical objects (e.g., the examples in Figure 1). Thus, one key axis is (1) co-location: are the users sharing virtual content co-located or not? A second key axis is (2) opt-in versus opt-out sharing: is sharing a deliberate opt-in action between specific people (as the HoloLens developer guidelines prioritize [43]) or are virtual objects public by default, requiring a deliberate opt-out (as the Meta developer guidelines advocate [40])? The example case studies we highlight explore these dimensions.

2.1 Case Study Applications

Paintball: Co-located, opt-in. In this app, users in the same physical space can play a game of paintball with virtual paint. All users can see the game objects (weapons, paint, etc.). Users may also have a dashboard where they can see the game status. This type of AR multiplayer gaming is already emerging in smartphone apps [20].

Multi-Team Whiteboards: Not (necessarily) co-located, opt-in. We envision a collaborative AR whiteboard app in which a user, possibly in a co-located group, may choose to share a whiteboard with other users who may be in the same or different physical locations. Although each co-located group of users sees the same whiteboard in the same location, different groups may see the whiteboard instantiated in different locations; furthermore, a user in a group may split off an individual copy of the whiteboard in order to leave the room and still collaborate from another remote space. The contents of all users' copies are synchronized in real-time. Since different whiteboards may have different levels of sensitivity, access control must be at least at whiteboard-level granularity. Unlike in *Paintball*, where a shared game state is core to app function, users of this app may encounter users with whom they don't want to share a sensitive whiteboard. This case study, also, is grounded in existing work: a pending patent application by Apple [29] describes a GUI for AR document editing, though it does not mention access control.

Community Art: Co-located, opt-out. We now consider an example in which co-located users automatically see each other's objects by default. We consider a virtual art app, where users can create and view sculptures, free-drawn markup, and other artistic artifacts made by other, potentially unknown AR users in the same physical (and virtual) space. Variants of *Community Art* might be used to decorate for a celebration so that guests or passersby will see the content, or to place advertisements outside one's shop. Though we consider *Community Art* as an example of a public-by-default app, some use cases may necessitate more fine-grained access control. For instance, artists may choose to keep their art private while constructing it or allow the public to view but not edit their sculptures. This case study is similar to Ubiquity6's smartphone app [67], in which all content is public.

Soccer Arena: Not co-located, opt-out. Finally, we consider an app that lets the user watch a virtual replica — e.g., on the user's living room table — of the soccer game that it is currently broadcasting. By default, all users of this app see all aspects of the playing field, commentator annotations, and ads. Some users may watch the game together in the same physical space, while others may be in separate physical spaces. While using the app, a user may wish to block a distracting ad or turn off annotations. The ability to form AR reconstructions of soccer games from monocular video footage, demonstrated in [51], shows that this app is within reach of today's technology. We find that *Soccer Arena* does not surface new security, privacy, or functionality requirements not covered by the other case studies. In particular, it raises the same spam-related concerns as *Community Art* does and the same non-colocation challenges as *Multi-Team Whiteboards* does. However, we include it for completeness.

[1] `arsharing.cs.washington.edu` or `arsharingtoolkit.com`

2.2 Functionality Goals

From the above case studies, we now derive a set of functionality design goals for multi-user AR apps and platforms. Any sharing control solution must coexist with these functionality goals — while one could trivially meet the security and privacy goals outlined in the next section by allowing *no* shared content, supporting sharing functionality is critical to the success of emerging multi-user apps.

- **Support physically-situated sharing.** For both *Paintball* and *Community Art*, physically co-located users will want to see the same virtual objects. The multi-user AR platform must support a way of sharing virtual state, and a mapping between virtual objects and the physical world, among the collaborating users.
- **Support physically-decoupled sharing.** *Multi-Team Whiteboards* requires that AR content be synchronized for each person's copy, regardless of the users' relative location — when they're in the same room, or adjacent rooms, or halfway across the world. Thus, the platform must support sharing virtual content decoupled from the physical world as well.
- **Support real-time sharing.** Users of *Paintball* will expect for their interactions with other players to occur in real time. Real-time state changes are also desirable for the other case studies. Thus, the platform must support low latency updates of shared state among multiple users, and any sharing control solution should not impose an undue performance burden. (Note that real-time performance also confers a security benefit, since access control changes can propagate quickly.)

2.3 Security Goals

A trivial solution that provides all of the above functionality would make all AR content public by default. However, interaction between multiple users may not always be positive. For example, a malicious or careless user may attempt to:

1. *Share unwanted or undesirable AR content* with another user. For example, in *Multi-Team Whiteboards*, a user may plaster a wall with offensive messages, or in *Community Art*, violate another user's personal space by attaching virtual objects to them as a practical joke. Such behavior has already manifested in shared VR settings [1, 66].
2. *See private AR content* belonging to another user. For example, in *Multi-Team Whiteboards*, a user may attempt to read another user's private whiteboard.
3. *Perform unwanted manipulations on AR content* created by or belonging to another user. For example, in *Community Art* or *Multi-Team Whiteboards*, a user may delete or vandalize another user's virtual creations. Such behavior has already appeared in the wild, with vandalism of AR art in Snapchat [38].

In response to such multi-user threats, we develop the following security and privacy goals for an AR sharing module.

Control of outbound content. Sharing of AR content involves two parties: the originator of the content and the recipient. We decompose our security goals along this dimension, beginning with control of outbound content, i.e., managing the permissions of other users to access shared content.

Three canonical access control rights are "read," "write," and "execute." Extending "read" and "write" to the AR domain (and deferring "execute" to Section 7):

- **Support granting/revoking per-user permissions.** The multi-user AR platform should support setting edit and view permissions for different users. A user of *Paintball* may wish to share a game session only among a specified friend group instead of allowing any nearby user to join, and may wish to retain full control of game administration even among the set of players.
- **Support granting/revoking per-object permissions.** A user of *Community Art* may wish to leave one piece of art publicly visible while working privately on another. Thus, regulating permissions at the granularity of the app is not sufficient to cover all use cases; object-level permissions must be supported as well.

The consequence of the above goals is that users in the same physical space may not share the same view of the virtual space. This is in sharp contrast to current technologies, where the physical presence of a device — e.g., a smartphone or a television set — enables the user of that device both (1) to signal to others that they are busy with that device, and (2) to establish a dedicated spatial region upon which their use of the device depends. The physicality of the device, then, serves as a scaffold around which interpersonal norms have developed. For instance, a person might avoid standing in front of a television set when a user is watching it, and might refrain from blocking the line of sight from a user to the smartphone they are holding.

AR content has no such physicality. Consider, for instance, *Multi-Team Whiteboards*: as thus far stated, a user looking at or interacting with a private whiteboard will appear to a nearby user as staring into or manipulating empty space. There is no mechanism by which the user can exercise agency over their working space in the presence of other users, and no mechanism by which other users passing by can determine whether the object is five feet away from its owner or up against a more distant wall. As a result, one user may inadvertently stand in front of content that a second user is interacting with. Further adding to this issue, prior work has also shown that people can be uncomfortable with the presence of AR users due to not knowing what the AR user is doing [14,35], and private content causes this rift even between users of the same AR platform. The Meta developer guidelines [40] thus recommend that developers build public-by-default content in accordance with human intuition about a shared physical world. Indeed, novice users in the same *physical* space may expect to also see the same *virtual* content [35]. It is possible that social behaviors

will adapt to this physicality disconnect over time, particularly around the current social discomfort of bystanders. But although social norms may change, and although mitigating these issues for bystanders — non-AR users, or AR users of a different and non-compatible platform — is difficult and beyond the scope of this work, we still seek to address this physical-world disconnect at least in the near term for multiple AR users of compatible platforms. Specifically, we wish to achieve the above content privacy goals while at least partially supporting a shared-world physical intuition:

- **Support physically intuitive access control.** An app may wish to signal to a nearby user that another user is (for example) drawing on a whiteboard, without revealing the content being drawn.

Control of inbound content. We next consider security properties from the perspective of the recipients of shared content. Since shared content can have serious implications for the receiver, such as spam that obscures important real-world information [34], we derive the following goals:

- **Support user control of incoming virtual content.** For instance, users of *Community Art* may wish to filter content to only that which is age-appropriate or that does not contain foul language.
- **Support user control of owned physical space.** In the case of *Community Art*, a user may not want arbitrary other users to attach content to their heads without consent, a homeowner may wish to prevent house guests from placing virtual content inside private rooms, and the keepers of a public monument may not want the monument to be vandalized with virtual graffiti. We note that users may want control over their physical space even when they cannot see the object in question: for instance, an app may wish to prevent a virtual "kick me" sign from being attached to a user's back such that the user cannot see and cannot control the sign. We consider the question of determining who controls a particular physical space to be out of scope for the design we present in Section 4 (see Section 4.4 for further discussion), and instead focus on enforcing owned physical space; however, we urge future work to also address this complementary issue.

2.4 Supporting Flexibility

Stepping back, in defining the above functionality and security goals, we observe that not all multi-user AR apps will have the same needs. For example, AR content that is shared with all users by default is suitable for some apps (e.g., *Community Art*) but not others (e.g., *Multi-Team Whiteboards*). Likewise, not all security and privacy goals are relevant in all cases: for instance, enforcing personal space for shared AR content may conflict with the functionality needs of *Paintball*, which requires that virtual paint stick to players upon a hit. Even in an app that is otherwise simple from a sharing control perspective, user needs may warrant a degree of added sharing control complexity: for instance, an AR assistive technology object that transcribes spoken words for deaf users may be exempt from the app's general rules for the enforcement of owned physical space so that it always remains visible to the deaf user who needs it.

Because the right sharing control model is app-specific, AR app developers will need the ability to implement multi-user features with their chosen model. To that end, we identify the need for a flexible AR sharing control module that can be leveraged by app developers. We envision this module as either an app-level library or an OS interface (i.e., set of APIs) that provides sharing control features. The advantage of an app-level library is that it does not require explicit changes to the platform. That is, an app developer could create an app that runs on different AR platforms and, by including the platform-appropriate version of a library, support interactions among users with different types of AR devices. For example, although we prototype our design as an app-level library for HoloLens, in principle it could be adapted for compatibility with Meta or Magic Leap apps.

3 Threat Model and Non-Goals

We aim to design a flexible module that helps app developers create multi-user AR apps that incorporate shared AR content while mitigating the risk of undesirable interactions between multiple users. We focus on the case of a developer building a single app and mediating the interactions of its multiple users, deferring to future work the problem of cross-app communication. We now present the threat model under which we develop our design in Section 4, as well as specify non-goals of this work.

Threat Model. Our primary focus in this work is on *untrustworthy users*. That is, we aim to help app developers create multi-user AR apps that are resilient to security and privacy threats between multiple users of the *same* app. In that context, we assume that two or more users are using the same AR app, written by the same developer and incorporating our sharing module. We thus assume that users trust both the developers of the apps that they install as well as their AR operating system, but that users may *not* trust each other. This trust dynamic is akin to traditional desktop environments — e.g., where two users of a Unix operating system might both trust the underlying system and installed apps, but might not trust each other and hence might not make their home directories world readable or world writable. A key difference, as noted earlier, is that in our model we only consider sharing of content between users of the same app.

Under this threat model, we do not consider malicious apps that omit or misuse our sharing module. We explicitly trust app developers to incorporate our module (e.g., as an app-level library) into their apps; a malicious app developer might choose to simply not use our sharing module, implementing their own adversarially-motivated sharing functionality, or use our module but violate security or pri-

vacy properties through out-of-band means. Though a user may install malicious apps alongside legitimate ones that use our module, these malicious apps cannot interfere via our module: we consider (and our module supports) AR content sharing only among multiple users of the *same* app, rather than also considering sharing *across* apps. This is consistent with the capabilities of current AR technologies, which are either single-app or do not allow multiple concurrently running apps to communicate [33]. We also assume that all users are running legitimate, uncompromised versions of the app; strategies for verifying [36,75] or enforcing [73,74] this assumption are significant research challenges of their own.

Finally, we assume that communication between devices is secured with today's best practices, e.g., end-to-end encrypted. Thus, we rule content eavesdropping and content modification attacks as out of scope. Current network best practices still suffer from denial-of-service attacks and traffic analysis attacks, but we do not aim to protect against such attacks in this work, focusing instead on the app-level security and privacy issues.

Non-Goals. We consider the following design questions to be non-goals of our present effort:

- **Non-goal: UI/UX design.** Although we propose underlying mechanisms for the sharing control needs of app logic, and although those mechanisms in some instances have implications for what developers are empowered to surface at the UI level, we do not aim to define exactly how those mechanisms should manifest to users in the specific interaction modality or look-and-feel of an app. Thus, we leave the design of specific interfaces — including how much of our module's control should be surfaced directly to users versus shouldered by the app — to future efforts by researchers and app developers. Our work is similar in spiritspirit to work on user interface toolkits (e.g., [25, 27]) in that our goal is to enable app developers to easily create and innovate on a range of user interfaces and experiences, rather than to design and iterate on these interfaces directly.
- **Non-goal: Network architecture design.** It remains an open question whether multi-user AR will ultimately be enabled by client-server, peer-to-peer, or other network architectures; we thus design our platform to be agnostic to network architecture. Additionally, we do not consider how two AR devices initially bootstrap communication; prior, complementary work considers how to securely pair two HoloLens devices [60].
- **Non-goal: App-specific choices about sharing control properties.** We do not aim to recommend to apps which sharing control properties and functionalities might make sense in the context of the app, instead enabling app developers to choose the appropriate subset of properties for their specific use cases.

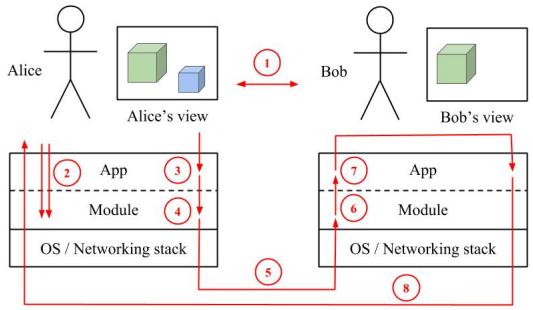

Figure 2: Basic object sharing flow: Alice creates the blue and green boxes and then chooses to share the green box with Bob. See Section 4.1 for details.

- **Non-goal: Accurate spatial localization of AR users and content.** We do not aim to design a system by which the location of an AR user can be accurately and securely determined. Prior work has studied how to localize devices accurately [23, 31], how to verify location claims [10, 69], and how to verify co-location claims [21, 55]. We note that even without further sharing controls, future location-based AR apps will benefit from secure location and co-location verification methods. Thus, we consider this topic to be orthogonal and of independent in.

4 Design

We now present the design of a module that AR developers can use to support secure and private sharing of AR content among multiple users. Compatible with our threat model of untrusted users but trusted developers, we envision this module as an app-level library or an OS interface.

4.1 Module Design Overview

To illustrate the relationships between the OS, the sharing control module, the app, and multiple users, we begin by walking through a simple case of Alice creating two objects and sharing one with Bob (Figure 2).

1. *Precondition:* Alice and Bob are both running an app that incorporates the sharing module and, as such, already have an open communications channel between their devices.
2. *Object creation:* Alice creates two AR objects, a small blue box and a large green box. Her app calls the sharing module's `InstantiateShared()` API for both objects, allowing the module to track permissions at the granularity of those objects (in this case, beginning with view and edit permissions for only Alice).
3. *Outbound sharing (app-level):* Through some user interface provided by the app, Alice chooses to share the green box with Bob.
4. *Outbound sharing (module-level):* On Alice's device, the app calls the sharing module's `SetPermission()`

API. As a result, the module updates its internal permission map, adding Bob to the list of users with view permissions for the green box.

5. *Communication:* The sharing module sends a message (via the device's OS and networking stack) containing object content and metadata to Bob's device, whose OS and networking stack dispatch it to the sharing module in Bob's instance of the app.
6. *Inbound sharing (module-level):* The sharing module surfaces a `SetPermission` event to Bob's app.
7. *Inbound sharing (app-level):* On Bob's device, the app shows some user interface to allow Bob to accept or deny the shared object. (Other apps may skip this step and show the object to Bob automatically, and/or respect Bob's previously-set preferences for shared objects from Alice.) Bob chooses to accept the shared object from Alice; the app updates his view of the world to include the green box.
8. *State update and communication:* The app calls the sharing module's `AcceptObject()` API, which in turn transmits that message back to Alice's device.

Following this transaction, Bob can now see a shared copy of Alice's green box and, depending on the sharing settings, can manipulate that box in ways that are also visible to Alice.

This sharing flow might seem simple: the sharing control module provides APIs that help an app keep track of which users can access which AR objects — i.e., view and edit permissions — and syncs this information across the devices of all users of the app. However, as surfaced in Section 2, sharing in the AR context requires thoughtful consideration — particularly in the face of users' expectations of and interactions within the physical world.

Key design challenges. While striving to achieve the functionality and security goals identified in Section 2, our design space exploration surfaced several key questions which do not arise for sharing and access control in traditional systems (e.g., file systems). These challenges are deeply connected with AR's integration with the physical world, and although they do not on the surface appear to be security-centered questions, they affect the security and privacy mechanisms we design, and so we must address them:

- *Integration of shared AR objects with the physical world (Section 4.2):* How is a shared object integrated into the physical world? In the above example, do Alice and Bob see the green box in the same physical location or in different physical locations? Are Alice and Bob themselves in the same physical location, and what happens when their co-location status changes?
- *Private content in a shared physical world (Section 4.3):* How should the sharing module handle or help shape users' expectations of private AR content, such as Alice's blue box, when they interact in a shared physical world?

	Outbound sharing controls	**Inbound sharing controls**
What and with whom	Permission management	Two-party sharing consent
Where	Location coupling (§4.2)	Personal space (§4.4)
How much	Ghosting (§4.3)	Clutter management

Table 1: Summary of the components of our design for controlling the outbound and inbound sharing of AR content.

- *Ownership of physical-world spaces (Section 4.4):* How can a sharing module help apps respect people's existing ownership of physical spaces? For example, users may wish to control AR content that they or others see in front of their homes or on their own bodies.

Effective solutions to these challenges must integrate with the system design components that have more direct analogues in current technologies. In particular, we incorporate the following established control structures into our design:

- *Permission management.* We leverage classic access control work [30] to track and enforce per-object and per-user permissions. Although we aim to be compatible with whichever access control model a particular app chooses to layer atop our module — e.g., a model akin to Google Docs for the *Multi-Team Whiteboards* case study — we note that this alone is not enough to support the 3D experience of AR, and that the above key design challenges must also be addressed.
- *Two-party sharing consent.* Some existing sharing models require that both the sharer and the receiver of digital content authorize a sharing event before its completion (e.g., Google Drive, Apple AirDrop). We use this principle in our design, with one twist: to help developers avoid decision fatigue in apps with high-volume content sharing, we allow the app to authorize a sharing event without the user in the loop. For instance, an app might automatically authorize content under some contexts but not others [70], use a notification UI that minimally disrupts the user's workflow, or allow users to always trust content from a specific other user. We advise developers to be conscious of habituation and interruption in their app designs.
- *Clutter management.* Our design supports temporarily or permanently removing an object from the user's field of view, as we discuss further in Section 4.2.

We summarize these aspects of sharing control, both new and precedented, in Table 1. We categorize the design points along two axes: (1) where in the above sharing flow the control occurs (*outbound* on the sharer's end, or *inbound* on the receiver's end), and (2) what type of control is enforced (*what* object is shared and *with whom*, *where* a shared object can be, or *how much* information from that object is shared).

4.2 Physical World Integration

The sharing flow in Section 4.1 demonstrates the basic building blocks of a sharing module, with view and edit permis-

sions for users at the granularity of AR objects (e.g., a virtual cat or virtual browser window). We now explore how these notions become significantly more complicated when shared AR objects are integrated into the physical world.

Location-coupled and -decoupled sharing. Recall from Section 2 that we aim to support both physically co-located sharing (i.e., two users in the same physical place and seeing the same AR objects in the same physical locations) and remote sharing (i.e., two users physically separated but seeing the same AR objects in their own physical spaces).

Accordingly, our design supports two notions for how an AR object can be shared with respect to the physical world: (1) *Location-coupled* objects, which all users see in the same physical location, and (2) *Location-decoupled* objects, where all users see the same object but in different physical locations. In the coupled case, if one user moves the object, other users also see the object's location update; in the decoupled case, the two instantiations of the object can be moved independently.

We intend for these notions not to be mutually exclusive for an object but rather to apply between sets of users. For example, an AR object (say, a virtual whiteboard) may be shared (1) in a location-coupled way between Alice and Bob co-located in New York, and (2) in a location-coupled way between Guanyou and Huijia in Beijing, and simultaneously (3) in a location-decoupled way between the two groups.

Handling people moving around the physical world. A challenge for location coupling and decoupling of shared objects arises when we consider that users' co-location can change as they move around the physical world.

For example, suppose that Alice and Bob share a location-coupled AR object — say, a whiteboard — both seeing it in the same physical location. Alice may *also* share it in a location-decoupled way with collaborator Carol working in another room — i.e., Carol will see an instantiation of the whiteboard in her own physical space. Initially, this appears to meet our goals: all users see the whiteboard in their own physical space in the same location as other co-located users.

What happens, however, when Carol moves into the same physical space as Alice and Bob? Since the whiteboard is shared among all of them, they will likely assume that all three of them can now see the same AR whiteboard object on the same wall [35]. This is not the case, however: Alice and Bob see one instantiation of the whiteboard, and Carol sees a separate instantiation in a slightly different location.

To resolve this potential inconsistency, our design keeps track of all copies of a shared object, allowing the app to show *all* of these copies to all users. Thus, when Carol joins Alice and Bob in the same room, all users see *both* versions of the whiteboard. All users then share the same view of the augmented physical world. Note that this location information may have privacy implications, though none in scope for this work; we discuss this point further in Section 7.

Moreover, since users' co-location may change over time, their desired location-(de)coupling of objects may change over time too. For example, in the above scenario, Alice and Bob may wish to merge their whiteboard object with Carol's instantiation, so that all three indeed see the same, single whiteboard object in the same place. Conversely, users may wish to collaborate on a location-coupled object while they are physically co-located but to both take their work with them when they physically separate. Thus, we also provide mechanisms to *merge* two location-decoupled instances into a location-coupled object and to *separate* a location-coupled object into two location-decoupled instances.

Another way to think about shared AR objects, then, is that there is one conceptual object and potentially multiple views of it. Each user sees a view in the same location as do all other users, and users in different physical spaces will see different views. If a user is in the same space as multiple views, that user will see all present views. The object's views may be manipulated separately in space; a single view may be split into two, or two may be merged into one, but the underlying object that all views represent remains the same.

Implications of location coupling for object deletion. A shared object's location coupling or decoupling has design implications for other features as well. For example, our design lets users delete AR content that they have created; it is not clear, however, that this decision should propagate to other users with whom the object has been shared, and location coupling or decoupling affects how deletion is handled.

We design the module to support three cases, which can be chosen by the app developer as appropriate:

1. *Case 1: Local Deletion: Affect user's local view of object only.* This option allows Alice to delete her object without affecting other users. If, in *Multi-Team Whiteboards*, Alice and Bob share a whiteboard in a location-coupled manner, Alice can delete her instance of the object while Bob keeps working.
2. *Case 2: Global Location-Coupled Deletion: Affect all users' views of location-coupled object.* Here, a deleted object is also deleted for all other users with whom that object was shared in a location-coupled way. That is, when Alice deletes her document, Bob also sees that document disappear. However, if Alice has also shared the document in a location-decoupled way with remote collaborator Carol, this option allows Alice to delete her and Bob's location-coupled instantiation without affecting Carol's remote, location-decoupled instantiation.
3. *Case 3: Global, Location-Independent Deletion: Affect all users' view of object, independent of location coupling.* In this case, *all* instantiations of the object for all users are deleted. Continuing the previous example, Alice, Bob, *and* Carol will all see the object deleted.

Which of these cases is most appropriate depends on the semantics of an app and each use case within that app.

Location coupling and decoupling have other design and implementation implications as well. For instance, hiding content with which the user does not currently wish to interact requires considering the same set of options as deletion.

4.3 Private Content in Shared Physical World

As raised in the *Multi-Team Whiteboards* case study in Section 2 and in prior work [35], the fact that AR supports per-user private content can have benefits, but it can also fail to provide a signal about the use of physical space (e.g., leading to one user inadvertently standing in front of another's virtual content, or causing social tension due to one user not knowing what another user is doing).

Thus, in this section we propose a design that allows users to socially signal to other AR users that they are busy interacting with private content without sharing the details of their activities. Further, we aim for this design to align with users' intuitions about a shared physical world.

Strawman designs. Consider two incomplete solutions:
Status quo. A solution with no further intervention would cause private content to be completely invisible to other AR users. A user interacting with private content thus appears to others as if the user were staring off into and manipulating an undefined region of empty space, giving no cue to other users about how far away the object is if they want to walk around it as well as no sense for what the user might be doing.
Occlusion by virtual barrier. Meta's developer guidelines recommend that sensitive content be shared publicly but occluded by a virtual barrier such as a curtain [40]. Although this does provide a shared-world physical intuition and social cue, it is not a robust privacy-preserving mechanism. Consider a user who places a virtual curtain around sensitive content so that the content is visible only from the user's point of view. A curious other user can surreptitiously look over the user's shoulder and observe the sensitive content, similar to shoulder-surfing with current mobile devices [56, 64].

Our approach: "Ghosts". We propose an alternate design that achieves our goal while avoiding the above drawbacks. The key idea is to allow users to share *that* they are interacting with an AR object, without sharing the *details* of that object. This idea is analogous to how a user interacting with a smartphone implicitly signals to bystanders that they are engaged in another activity located in the palm of their hand, without the contents of that activity being directly revealed, or to how users may share free/busy information about specific time blocks on their calendar to avoid double-booking without sharing the details of their calendar events.

To support this interaction, we introduce a new partial-visibility state for shared AR objects that we call *ghosting*. Ghost objects show only their location in space, not the sensitive content they contain, no matter at which angle they are viewed. As such, unlike the above smartphone analogy, they are not susceptible to shoulder-surfing. Furthermore, a user with whom a ghost object is shared receives from the sharer only the data needed to instantiate the ghost, rather than the full object data; this further insulates the private content.

Ghost shape granularity. For non-planar objects, we encounter the following question: How does the sharing module determine the appropriate level of granularity to expose in the ghosted object, given that object shapes may contain app-specific information content?

For instance, in *Community Art*, the ghost of a sculpture that is private during its development should not mimic the original sculpture's shape too closely. However, a shape with too coarse of a granularity — e.g., a large fixed-size cube regardless of the sculpture — no longer gives meaningful physical-world information to nearby users: e.g., by marking an unreasonably large physical space as occupied.

To balance this dilemma,we allow app developers to specify what the ghosted view of an object should look like. This approach allows for non-planar objects to assume an obscured shape appropriate for the app-specific information they carry. For instance, in the *Community Art* case study, a sculpture that is private during its development might be displayed to others as an appropriately sized cylinder.

4.4 Respecting Ownership of Physical Spaces

Finally, we turn to the question of how the sharing control module can help apps respect people's existing ownership of physical-world space. We refer to both personal space (near one's body) and static owned space (e.g., one's home) as *owned physical space* in this section.

Helping users protect their owned physical spaces requires several components: (1) Determining who owns a region in space, (2) Determining what the boundaries of that region are, and (3) Enforcing some kind of policy on shared AR objects in that region.

We defer to future work (1), how to determine *who owns* a region in space. This in itself is complex; for instance, Google has analyzed abuse of location ownership in its Maps app [26], and prior work has considered physical world ownership for restricting continuous sensing apps (including AR) [54]. Accounting for different types of space ownership is also nontrivial: in particular, we identify (a) fixed-location physical spaces (e.g., a house, a room, a storefront, or a public park), (b) person-relative spaces (e.g., within 5 feet of a user), and (c) object-relative spaces (e.g., within 35 feet of a virtual art object). A complete solution to this issue should also consider non-AR users, and we offer the following suggestion as a starting point for future work: locally on the AR user's device, employ computer vision techniques to identify the spatial positions of bystanders visible by the AR user, estimate the rough pose for each bystander, and use that information to mark bystanders' forms as protected regions of space (e.g., using techniques from [2, 39]). However, we do not pursue this topic further in this work, since it is a complex area of investigation in its own right; instead, we assume a prior definition of owned physical space, and we focus on

the challenges of enforcing that space.

For (2), we observe that defining the boundaries of a protected region of physical space — e.g., around a person or house — is not a simple binary determination. A user might perceive an object two feet away to be too close, for instance, but may consider an object nine inches away to be even more so; this definition may also vary across different users [22]. Building on this observation, our solution is to model owned physical space as a *continuum*, where violations become more severe — and thus policies could become stricter — as virtual content approaches the protected region.

The key question, then, is (3): what should an app do when one user's shared AR content overlaps another user's personal physical space, or a physical region (e.g., a house) that another user owns?

Policies for AR content violating owned spaces. To answer this question, we propose that the sharing control module can provide a variety of policies that an app's developer could choose to apply in such a case. These policies can be enforced either such that the result of enforcement is only visible to the user whose personal space is in question, or such that *all* users with access to the shared object see the result of enforcement. App developers may choose which policies to enable based on the needs of the app.

A simple, binary policy might make objects *invisible inside a fixed radius*: for instance, define a three-foot radius around a person within which AR content should not be visible (at least to that user). Such a policy can be exploited by surrounding the person with AR content just outside the boundary; thus, there is a tension between a large radius to minimize the effects of such an attack and a small radius to enable legitimate functionality.

To help balance this tension, and to take advantage of the continuum in the boundary described above, our design provides a *transparency gradient* policy, by which the module makes shared objects more transparent the closer the objects are to the boundary of any protected region (e.g., around a person). Under this policy, objects would start becoming transparent much farther than three feet away, avoiding blocking the person's vision but still being useful.

By having owned-space policies be enforced by apps themselves (via the sharing module), rather than by users, they can be applied without changing the underlying permissions on the shared object. This avoids two pitfalls. First, it prevents malicious users from exploiting the policy, e.g., by gaining control of an object simply by walking up to it. Second, it enables policy enforcement even on objects that the physical space owner cannot see (protecting even non-AR user bystanders from the "kick me" sign from Section 2).

5 Implementation

We now describe our prototype, which we implement as an app-level library for the Microsoft HoloLens, demonstrating the feasibility of our design for a currently available head-mounted AR platform. Our prototype, called ShareAR, is implemented in C# and uses the HoloLens Unity development kit. We implemented the concept of an AR object using the Unity GameObject primitive, which is a virtual entity comprising shape, texture, location, physics properties, script-controlled behavior, etc. Our implementation consists of a core module (1888 lines of code), a network shim layer (1137 lines of code), and a short supplementary script to accompany any object shared using the toolkit (45 lines of code), totalling 3070 lines of code.[2]

The ShareAR core comprises:

- Data and meta-data, including an access control matrix [30] and options for how objects are shared (e.g., location coupled or decoupled).
- Methods to instantiate objects, manually or automatically accept shared objects, change permissions on objects, and sync data between users. Table 2 summarizes sample corresponding message types in our prototype.
- Simple fixed-radius personal space controls in the form of Unity's *plane clipping*, where the portion of an object closer to the user than the fixed plane-clipping distance is not rendered. We did not implement more nuanced controls, like our proposed transparency gradient.

Though network architecture is out of scope of our design, in practice we must choose some way to connect between HoloLens devices. In our prototype, we used the MixedRealityToolkit Sharing toolkit, an open-source library from Microsoft.[3] MixedRealityToolkit does not provide any sharing control or access control functionality; we use it only as a basic tunnel to send messages between HoloLens devices.

We build a network shim layer that serializes and deserializes ShareAR messages and uses MixedRealityToolkit Sharing to send them between devices. A developer who wishes to use a different networking solution — e.g., one relying further on a central server for data storage, or one implementing a more rigorous consensus protocol — may write a replacement network shim layer satisfying the same interface with the ShareAR core.

Users may join, leave, and re-join the network. To be robust to access control changes occurring while a user is offline, we include in our implementation a means for a newly reconnected user to receive a "digest" version of an object containing only the information needed for consistency with the other online users. Since consensus is best done with network architecture in mind, we provide a means to create this object "digest" as a higher-level functionality but relegate consensus operations to the networking shim layer.

[2] To calculate lines of code, we use the CLoC tool version 1.80 available at `https://github.com/AlDanial/cloc/releases/tag/v1.80`. We omit lines of code solely related to our performance evaluation.

[3] `https://github.com/Microsoft/MixedRealityToolkit-Unity`

Message name	Sent when	Bytes
`InstantiateShared`	A new shared object is created	104
`AcceptObj`	A newly-shared object is accepted	22
`SetPermissionNew`	A newly-instantated public object is accepted and there are more than 2 users present	38
`SetPermissionObject`	A permission change is made or offered on an existing object	92
`SetPermission`	A permission change on an existing object is accepted and there are more than 2 users present	54
`UpdateLocation`	A shared object's location in space is updated	62
`DeleteShared`	A shared object is deleted	22

Table 2: Example message types and sizes in our prototype. Messages are relatively small because they do not include full AR object meshes but rather an ID corresponding to the type of object in question and a string of object data that fully specifies the particular object of that type. Sizes are for basic objects with no additional object data.

Feature	*Paintball*	*Cubist Art*	*Doc Edit*
Location-coupled sharing	✓	✓	
Location-decoupled sharing			✓
Public permission settings	✓	✓	✓
Ghost-only permission settings			✓
Private permission settings		✓	✓
Auto-accepting content	✓	✓	
Accepting content ad hoc			✓
Local deletion		✓	✓
Global location-coupled deletion		✓	
Global location-indep. deletion			✓
Updating object location		✓	✓
Updating object data		✓	✓

Table 3: ShareAR sharing control features in case study apps.

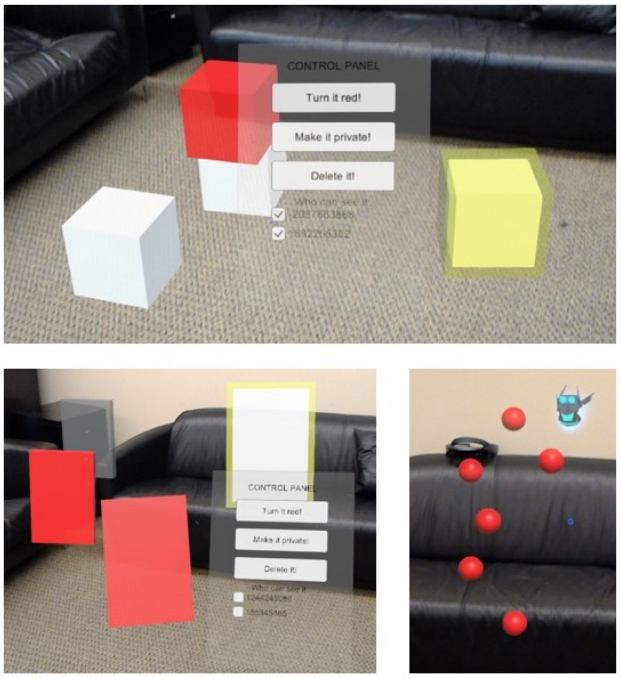

Figure 3: Screenshots of prototype apps: Cubist Art (top), Doc Edit (bottom left), and Paintball (bottom right). In the Doc Edit app, the semitransparent gray box in front of the file cabinet in the upper left is a ghost view of another user's document, and the two red boxes are two users' separate instantiations of the same shared document.

6 Evaluation

We now evaluate our prototype's functionality (Section 6.1), security (Section 6.2), and performance (Section 6.3).

6.1 Functionality Evaluation

We evaluate the functionality of our prototype by implementing case study apps and by comparing against existing AR design guidelines. We find that our prototype is flexible enough to support a range of app sharing control needs and is compatible with all considered existing design guidelines.

Case study applications. To evaluate the flexibility of our design to support our functionality goals, and the associated developer effort, we built bare-bones prototype versions of our case studies from Section 2.1: Paintball, Doc Edit (a variant of *Multi-Team Whiteboards*), and Cubist Art. (We did not implement *Soccer Arena*, since it does not surface new security, privacy, or functionality requirements not covered by the other case studies. Section 2 provides further analysis.) Our prototypes are intended to cover a broad spectrum of sharing control functionalities; see Appendix A for detailed descriptions of the apps. Screenshots of the apps are in Figure 3, the range of sharing control features each exercises is in Table 3, and the sharing-related lines of code for each is in Table 4.

We use lines of code as a proxy measure for developer effort (see Table 4). For each app, we count the total lines of code in the app and the lines of code specific to sharing functionality. The low number of sharing-related lines of code suggests that the burden on app developers to use our toolkit in practice is reasonable. Furthermore, we conjecture that fully fledged apps are likely to contain many more lines of code unrelated to sharing, further reducing the comparative proportion of sharing-related lines of code in the app. We note also that the repetition of some prototyped features across multiple apps (such as location-coupled sharing in both the Paintball and Shared Blocks apps) suggests that ShareAR's features are composable, and that developers can choose an app-appropriate subset of functionality.

Compatibility with existing guidelines. We also consider the compatibility of ShareAR with existing design guidelines from AR headset manufacturers. We focus on guidelines related to multi-user interactions, asking: Does ShareAR allow an app developer to meet these guidelines? We in-

App	Sharing LoC	Total LoC
Paintball	13	240
Doc Edit	173	1236
Cubist Art	153	1131

Table 4: Lines-of-code counts for the three prototype applications. We report both the total lines of code for the application and the lines of code dedicated to interfacing with the ShareAR toolkit.

vestigate the Microsoft HoloLens guidelines [43] and Meta guidelines [40]; we find that ShareAR is compatible with all of them. The results are summarized in Table 5; see Appendix B for additional information.

6.2 Security Evaluation

We examine the security and privacy of our ShareAR-enabled apps under our threat model of untrusted users (but trusted OS and apps). As described in Section 3, we rely on app developers to use the ShareAR APIs that are appropriate for their use case. For the sake of exposition, we focus on the Doc Edit app since it invokes all of the restrictions our ShareAR prototype supports; our observations also extend to Paintball and Cubist Art where applicable.

We find that ShareAR's security and privacy restrictions function as intended and meet the security goals in Section 2.3. Considering first the outbound security goals:

- *Support granting/revoking per-user permissions*: The Doc Edit app includes a menu that allows a user to grant and revoke per-user permissions. A user who never received or no longer has view permissions on a document cannot see it.
- *Support granting/revoking per-object permissions*: The aforementioned menu controls permissions on a per-document basis: only the currently selected document is affected by a permission change.
- *Support physically intuitive access control restrictions*: Doc Edit provides a ghost version of a document object (as a flat gray box). A user with permission only to view the ghost cannot tell if the original document is red or not; but the user sees the ghost in the same location as the document's owner sees the original document, and this location remains synchronized when the document's owner moves the document in physical space.

Considering the inbound security goals from Section 2.3:

- *Support user control of incoming virtual content*: The Doc Edit app surfaces an incoming permission-granting message to the user via a small menu, through which the user can choose to accept or decline. If the user accepts, a new (location-decoupled) instantiation of the document appears in front of the user, and the user can also see the sharer's instantiation of the document in its full (rather than ghost) form. If the user declines the document, no such change occurs.
- *Support user control of owned physical space*: As described in Section 5, our prototype leverages a Unity plane-clipping feature to implement simple owned physical space enforcement. We clip parts of any object closer to the user than 0.85 m (with the distance chosen to match a HoloLens recommended setting [42]).

6.3 Performance Evaluation

We now evaluate ShareAR's performance, measuring its operations (and comparing them to baseline operations where possible) and studying how it scales with numbers of virtual objects and users. We find that ShareAR imposes only a modest overhead on interdevice communication even as the number of objects and users increases.

Experimental setup. We build an app (1506 lines of C# code) to exercise ShareAR's components and measure its performance. In our test app, a test device creates objects that are location-coupled or location-decoupled, sharing them publicly, as ghosts, or keeping them private. One or more other test devices auto-accept or manually accept objects. The first device changes the objects' permissions, updates the objects' location, and finally deletes the objects.

Our experimental setup consists of five HoloLens devices communicating on the same local network, in two experimental scenarios: (1) for each $n \in \{1,2,3,4,5\}$, we select n devices and fix $h = 1$ shared AR object; (2) for each $h \in \{2^0, 2^1, 2^2, 2^3, 2^4, 2^5\}$, we set h to be the number of objects present and fix $n = 2$ devices. All devices run our evaluation app with the same n and h parameters; all devices join the network sequentially, and then the last device to join the network triggers the evaluation app.

The operations we measure are `Create`, `Accept Create`, `Change Permission`, `Accept Change`, `Update Location`, and `Delete`. Each operation involves work done on User A's device to initiate the operation, a message sent across the network from User A to User B, and work done on User B's device to process the operation. Note that in some cases (for `Create` and `Change Permission`) User B's device reacts by initiating an operation that we also measure (`Accept Create` and `Accept Change`).

In addition to measuring the within-module time for A's initiating action and B's receiving action, we measure and report on *operation completion time*, i.e., the time it takes from A's initiating action until B has finished processing. To correct for clock skew between the two devices, we add into the evaluation script a message from B to A containing B's timestamps (note that this is not part of our module's protocol, but exists solely for the purposes of the evaluation). Device A then combines this information with its own timestamps to compute the final timing numbers.

Finally, for the sharing module operations that clearly correspond to a primitive Unity operation (`Create`, `Update Location`, and `Delete`), we also measure as a baseline the timing of the Unity operation.

We repeat each evaluation point — defined by operation, configuration (e.g., location coupled or decoupled), number

Guideline	Short description	Source	Support?
How are they sharing?	Design for app's purpose of sharing: presentation, collaboration, guidance	HoloLens Developer Guidelines [43]	✓*
What is the group size?	Accommodate as many users as the app expects to need	HoloLens Developer Guidelines [43]	✓*
Where is everyone?	Support users in the same or different physical spaces as needed	HoloLens Developer Guidelines [43]	✓
When are they sharing?	Design for asynchronous or real-time sharing as appropriate	HoloLens Developer Guidelines [43]	✓*
How similar are their physical environments?	Place objects appropriately in non-co-located users' environments	HoloLens Developer Guidelines [43]	✓
What devices are they using?	Integrate with VR as needed	HoloLens Developer Guidelines [43]	✓*
Clip planes near user	Set minimum visible distance for object to 0.85 m	HoloLens Hologram Stability Guidelines [42]	✓*
Do not disturb	Avoid incessant notifications to user	Meta Developer Guidelines [40]	✓*
The holographic campfire	Allow users to see each other	Meta Developer Guidelines [40]	✓
Public by default	Support shared-world intuition by making content publicly visible	Meta Developer Guidelines [40]	✓*

*Table 5: Summary of ShareAR's compatibility with existing multi-user AR design guidelines. For the check marks with a * appended, see Appendix B for additional details.*

of users, and number of objects — for 2 warm-up trials and 5 measured trials, reporting the mean and standard deviation.

Basic profile of operations. We begin by considering ShareAR's operations with a single pair of users ($n = 2$) sharing a single object ($h = 1$).

The overall operation completion time is between approximately 70 ms and 250 ms, depending on the operation and configuration. This overall time is significantly dominated by factors external to the ShareAR module, and that *any* multi-user sharing solution would encounter: i.e., the network latency and the HoloToolkit Sharing library on either end of it. The overhead specific to ShareAR is minimal: we find that `Create` and `Change Permissions` operations are most expensive on average, still taking less than 5 ms in the worst case for the computation on each device; all other operations take less than 1 ms on each device. For the operations that we can compare directly with Unity baselines, we also find that ShareAR's overhead is minimal: the operations stay within 2.5 ms of the baseline in the worst case.

Scaling with the number of users. Next, we consider how ShareAR scales as the number of users increases.

In terms of network traffic, a user sharing an object needs to send object create and update messages to $n-1$ others; additionally, once a user accepts a sharing offer, their device sends an acceptance message back to the sharer and an informational message to all other $n-2$ users to stay in sync. The total number of messages in the interaction thus scales quadratically. For updates to already-shared objects (location change, deletion), the sharing user sends one message per other user, and no replies or additional messages are sent (overall scaling linearly with users).

In terms of timing, all operations under all test conditions took less than 5 ms. For all but `Create` and `Change Permission`, the operations on average remained under 1 ms. These overheads are reasonable, especially given their small additional overhead beyond to the corresponding baseline operations where present (shown with a dashed line for `Create`, `Update Location`, `Delete`). More detailed performance data is in Appendix C.

In terms of scaling, `Create` and `Change Permission` scale approximately linearly with the number of users; all other operations remain approximately constant. Different configurations for an operation (e.g., location coupled versus decoupled sharing, or different object deletion modes) may slightly affect performance (as reflected in the differently colored lines in the graphs), typically taking longer for location-decoupled objects due to the overhead of processing multiple instantiations of the same object.

Scaling with the number of objects. Finally, we measure ShareAR with increasing numbers of AR objects.

In terms of network traffic, we observe that it scales linearly with the number of objects, as each operation and associated message is independent per object.

In terms of timing, all operations took less than 3 ms (and often less than 1 ms). These overheads are reasonable, especially given their relation to the corresponding baseline operations where present. (For the module operations for which a baseline Unity operation is plotted — `Create`, `Update Location` and `Delete` — the relevant module operation timing is very close to that of the baseline Unity operation.) Additional details are in Appendix C.

In terms of scaling, for all operations, the time taken is approximately constant per object as the number of objects scales: in other words, an operation on one object registered with the module is independent of how many other objects are also registered with the module. Some operations exhibit a slight slope downward, suggesting caching benefits.

Performance evaluation summary. From our measurements, we see that object creation and permission changes are the most computationally expensive operations. However, we anticipate that in practice these operations will only occur during a small fraction of the frame updates in an app.

Even so, the greatest observed time taken for an operation was under 5 ms, and most measurements remained under 1 ms. Furthermore, since these measurements were of our unoptimized research prototype, continued code optimization may bring the performance overhead down even further.

7 Discussion

This work presents the first systematic investigation of multi-user sharing control for AR apps. We propose a module that is flexible enough to support many different decisions by app developers. Below we discuss several examples of future directions enabled by our work.

Execute permissions. Although multi-user AR systems are still primitive, we envision that future systems will support not only read and write but also execute permissions. One possible manifestation may be to allow a user to execute *predefined actions* on another user's object without having full edit control. For instance, an app may allow other users to make a virtual dog wag its tail without allowing them to make the dog arbitrarily large. Our module can be extended to include additional permissions, including this one.

Asynchronous sharing. Our design exploration assumes that both users are online when a sharing action occurs; extensions of our work could explore removing this assumption. For example, consider a user who places publicly visible virtual decorations outside their home. We may want (1) the objects to still be visible to a passerby when the user is not home, but (2) the passerby's device to only become notified of the objects' existence and public visibility when the passerby is physically proximate to the home. Such a design may require an alternate network architecture than peer-to-peer; our module's network agnosticism would support this.

Minimizing developer errors. We emphasize that one consequence of our module's flexibility is that developers must be cautious to use it in a way that supports their app use case. Some potential user-to-user threats may be subtle: for example, if app developers chose to share ghost objects automatically with no way to refuse or delete them, one user might intentionally or accidentally clutter another user's view with ghost objects (an example of a denial-of-service attack). Or, if an app developer implements a personal space policy that makes AR objects invisible to all users but does not provide a way to interact with or retrieve an invisible object, then a malicious user could walk up to others' objects to force them to become invisible and non-interactible. Still other pitfalls may depend on app semantics: for instance, if the developer of an app such as *Community Art* does not put limits on users, a user could monopolize a common space and prevent other users from placing objects there. Future work, then, may explore ways to support app developers in using the features from our system that are most suitable for their overall goals.

Analysis in the wild. More broadly, our work lays a foundation for future empirical studies on how developers use our module's components in practice and how users respond to concrete usage of these components. Such an evaluation is nontrivial since evaluating the usability of a single app does not generalize well to the usability of others [45], for the same reason described in Section 2 that a sharing control module cannot be one-size-fits-all. However, we note the importance of follow-up studies considering user perceptions when making specific design decisions, and we encourage future work to leverage our technical foundation to examine under which circumstances certain sharing mechanisms are appropriate.

Location privacy. Much multi-user app functionality, including our design, requires that users share their location with the app: sharing at least where one is within a physical space is necessary for location-synchronized virtual content. Some users may not anticipate or agree with such location sharing, even for trustworthy apps, though such sharing may be fundamental to the design of location-based AR apps. Additional location privacy concerns could be introduced by app developers, if app developers mishandle and accidentally or intentionally expose a user's location to *other users*. This threat, however, is dependent upon app-level semantics, and is neither unique to nor preventable by the underlying sharing framework. We encourage future work to explore this point further.

Inherently conflicting goals. Finally, we conjecture that there may be fundamental tensions in some aspects of secure and private content sharing between users. For example, consider the case of a shared space in which one user owns a publicly visible ball object and another user owns a private wall object. When the public ball is thrown at the private wall, it is not obvious which user(s) should see the ball rebound. If the ball rebounds for both users, then the ball owner gains information about the presence of the wall; if the ball does not rebound for either user, then the wall owner sees the ball go through the wall, defying physics; if only the wall owner sees the ball rebound, then the two users no longer have a synchronized view of the shared space. Determining how physics-obeying virtual objects should interact to minimize information leakage via this side channel while maximizing physical intuition is a subtle area for future work, and we conjecture that *no* content sharing solution can simultaneously achieve both goals perfectly.

8 Related Work

Although AR has a long history (e.g., [61]), the computer security community has only recently begun examining the space [13, 52]. Prior efforts on AR security and privacy include filtering raw real-world input [12, 28, 49, 54, 65] and regulating untrusted AR output [32, 34]. These efforts focus on the case of a single user interacting with an AR device.

Literature on multi-user AR security and privacy is just beginning to emerge. Some prior work has proposed methods

for secure device pairing via out-of-band channels [19, 60]; our work is complementary. Other prior work has proposed specific multi-user interaction modalities, such as location-based interfaces for making virtual content private and auditing content visibility [8, 9], mediating shared experiences with remote collaborators [50], and using personal tablets in shared spaces to separate private and public content [62, 72]. While these works present specific multi-user AR systems, our work is the first to systematically and broadly consider the design space for AR sharing control and our module could be leveraged when implementing these prior ideas.

There is a rich literature on access control (see, e.g., [7] for an overview). Our work does not assume what access control model is best for a particular app. Our implementation leverages an access control matrix [30] as a simple and flexible model for per-user and per-object permissions; we intend for other established access control models in specific app contexts (e.g., [17, 68]) to be layered on top of our toolkit, and we instead focus on the challenges of managing the implications of access control in the 3D physical AR setting.

Work in AR user experience has surfaced security- and privacy-relevant themes for multi-user contexts. Lebeck et al. [35] surface multi-user concerns such as physiological attacks, virtual clutter, and the obscurity of other users' actions. Poretski et al. [48] examine normative tensions in AR, emphasizing enforcing personal space and designing for user control. Olsson et al. [46,47] identify user needs such as control over privacy, socially appropriate ways to interact with devices, and solutions for abuse of public content by other users. These studies shed light on desired system properties and user concerns but do not directly address system design; our work builds concretely on these findings.

Multi-user digital interactions that take place in a physical space have also been studied in the context of tabletop interfaces and large computerized displays [44, 57–59, 71]. Our work addresses similar needs for and tensions around public versus private content arising in the AR setting, where immersive 3D content can be situated anywhere in the physical world rather than constrained to a shared display.

9 Conclusion

Multi-user AR technologies hold much promise, but also raise security and privacy risks in the potentially undesirable interactions between human users. These risks should be addressed while AR ecosystems are being actively developed rather than after sub-optimal ad hoc conventions have taken root. To that end, we are the first to systematically develop a set of security and functionality goals for multi-user AR. We present the design of a sharing control module for AR content, which we envision as an app-level library or OS interface that can be leveraged by app developers. Our work identifies and addresses key challenges that stem from AR's tight integration into the physical world. Our prototype, ShareAR,[4] for the Microsoft HoloLens demonstrates the feasibility of our design, and our evaluation suggests that it meets our design goals and imposes minimal performance overhead. By addressing multi-user AR sharing control systematically now, we are taking steps toward securing the fully fledged multi-user AR applications of the future.

Acknowledgments

We thank Ivan Evtimov, Earlence Fernandes, Kiron Lebeck, Lucy Simko, and Anna Kornfeld Simpson for valuable discussions and feedback on previous drafts; we thank James Fogarty for his advice on tabletop interface related work. This work was supported in part by the National Science Foundation under awards CNS-1513584, CNS-1565252, and CNS-1651230, and by the Washington Research Foundation.

References

[1] J. Alexander. 'Ugandan Knuckles' is overtaking VRChat, Jan. 2018. https://www.polygon.com/2018/1/8/16863932/ugandan-knuckles-meme-vrchat.

[2] R. Alp Güler, N. Neverova, and I. Kokkinos. DensePose: Dense human pose estimation in the wild. In *CVPR*, 2018.

[3] E. Alvarez. Facebook's next big augmented reality push is multiplayer games, Sept. 2018. https://www.engadget.com/2018/09/07/facebook-ar-games-multiplayer-first-look/.

[4] ARCore. https://developers.google.com/ar/.

[5] ARKit. https://developer.apple.com/arkit/.

[6] AR Studio. https://developers.facebook.com/products/camera-effects/ar-studio/.

[7] M. Bishop. *Computer Security: Art and Science*. Addison-Wesley Professional, 2nd edition, 2018.

[8] A. Butz, C. Beshers, and S. Feiner. Of vampire mirrors and privacy lamps: Privacy management in multi-user augmented environments. In *ACM UIST*, 1998.

[9] A. Butz, T. Höllerer, S. Feiner, B. MacIntyre, and C. Beshers. Enveloping users and computers in a collaborative 3D augmented reality. In *IEEE/ACM International Workshop on Augmented Reality*, 1999.

[10] J. T. Chiang, J. J. Haas, and Y.-C. Hu. Secure and precise location verification using distance bounding and simultaneous multilateration. In *WiSec*, 2009.

[11] S. Curtis. Sex pests are using Apple AirDrop to send explicit pictures to unsuspecting commuters, Aug. 2017. https://www.mirror.co.uk/tech/sex-pests-using-apple-airdrop-10987968.

[12] L. D'Antoni, A. Dunn, S. Jana, T. Kohno, B. Livshits, D. Molnar, A. Moshchuk, E. Ofek, F. Roesner, S. Saponas, et al. Operating system support for augmented reality applications. *HotOS*, 2013.

[13] J. A. de Guzman, K. Thilakarathna, and A. Seneviratne. Security and privacy approaches in mixed reality: A literature survey, 2018. http://arxiv.org/abs/1802.05797.

[4] See arsharing.cs.washington.edu or arsharingtoolkit.com

[14] T. Denning, Z. Dehlawi, and T. Kohno. In situ with bystanders of augmented reality glasses: Perspectives on recording and privacy-mediating technologies. In *CHI*, 2014.

[15] Digi-Capital. Ubiquitous $90 billion AR to dominate focused $15 billion VR by 2022, 2018. https://www.digi-capital.com/news/2018/01/ubiquitous-90-billion-ar-to-dominate-focused-15-billion-vr-by-2022/.

[16] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, and D. Wagner. How to ask for permission. In *HotSec*, 2012.

[17] D. Ferraiolo and R. Kuhn. Role-based access controls. In *NCSC*, 1992.

[18] C. Fink. The trillion dollar 3D telepresence gold mine, Nov. 2017. https://www.forbes.com/sites/charliefink/2017/11/20/the-trillion-dollar-3d-telepresence-gold-mine/#42b8f0a12a72.

[19] E. Gaebel, N. Zhang, W. Lou, and Y. T. Hou. Looks good to me: Authentication for augmented reality. In *TrustED*, 2016.

[20] J. Gallagher. Upcoming game easily shows you how to master paintball, Aug. 2017. https://mobile-ar.reality.news/news/apple-ar-upcoming-game-easily-shows-you-master-paintball-0179651/.

[21] G. Hancke and M. Kuhn. An RFID distance bounding protocol. In *SECURECOMM*, 2005.

[22] L. A. Hayduk. Personal space: Where we now stand. *Psychological Bulletin*, 94(2):293–335, 1983.

[23] T. He, C. Huang, B. M. Blum, J. A. Stankovic, and T. Abdelzaher. Range-free localization schemes for large scale sensor networks. In *MobiCom*, 2003.

[24] Microsoft HoloLens. https://www.microsoft.com/microsoft-hololens/en-us.

[25] S. Houben and N. Marquardt. WatchConnect: A toolkit for prototyping smartwatch-centric cross-device applications. In *CHI*, 2015.

[26] D. Y. Huang, D. Grundman, K. Thomas, A. Kumar, E. Bursztein, K. Levchenko, and A. C. Snoeren. Pinning down abuse on google maps. In *WWW*, 2017.

[27] S. E. Hudson, J. Mankoff, and I. Smith. Extensible input handling in the subArctic toolkit. In *CHI*, 2005.

[28] S. Jana, D. Molnar, A. Moshchuk, A. M. Dunn, B. Livshits, H. J. Wang, and E. Ofek. Enabling fine-grained permissions for augmented reality applications with recognizers. In *USENIX Security*, 2013.

[29] S. W. Kim. 3D document editing system. U.S. Patent Application 20180081519, 2016, https://bit.ly/2N5Dt2S.

[30] B. W. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8(1):18–24, Jan. 1974.

[31] L. Lazos and R. Poovendran. SeRLoc: Secure range-independent localization for wireless sensor networks. In *WiSe*, 2004.

[32] K. Lebeck, T. Kohno, and F. Roesner. How to safely augment reality: Challenges and directions. In *HotMobile*, 2016.

[33] K. Lebeck, T. Kohno, and F. Roesner. Enabling multiple applications to simultaneously augment reality: Challenges and directions. In *HotMobile*, 2019.

[34] K. Lebeck, K. Ruth, T. Kohno, and F. Roesner. Securing augmented reality output. In *IEEE Symposium on Security & Privacy*, 2017.

[35] K. Lebeck, K. Ruth, T. Kohno, and F. Roesner. Towards security and privacy for multi-user augmented reality: Foundations with end users. In *IEEE Symposium on Security & Privacy*, 2018.

[36] L. Li, D. Li, T. F. Bissyandé, J. Klein, Y. Le Traon, D. Lo, and L. Cavallaro. Understanding Android app piggybacking: A systematic study of malicious code grafting. *IEEE TIFS*, 12(6):1269–1284, 2017.

[37] Magic Leap. https://www.magicleap.com/#/home.

[38] L. Matney. Jeff Koons' augmented reality Snapchat artwork gets 'vandalized', Oct 2017. https://techcrunch.com/2017/10/08/jeff-koons-augmented-reality-snapchat-artwork-gets-vandalized/.

[39] D. Mehta, O. Sotnychenko, F. Mueller, W. Xu, S. Sridhar, G. Pons-Moll, and C. Theobalt. Single-shot multi-person 3D pose estimation from monocular RGB. In *3DV*, 2018.

[40] Meta. Spatial interface design: Public by default. https://devcenter.metavision.com/design/spatial-interface-design-principles-public-by-default.

[41] https://www.metavision.com/.

[42] Microsoft. Hologram stability. https://docs.microsoft.com/en-us/windows/mixed-reality/hologram-stability.

[43] Microsoft. Shared experiences in mixed reality. https://developer.microsoft.com/en-us/windows/mixed-reality/shared_experiences_in_mixed_reality.

[44] M. R. Morris, A. Cassanego, A. Paepcke, T. Winograd, A. M. Piper, and A. Huang. Mediating group dynamics through tabletop interface design. *IEEE CGA*, 6(5):65–73, 2006.

[45] D. R. Olsen, Jr. Evaluating user interface systems research. In *UIST*, 2007.

[46] T. Olsson, T. Kärkkäinen, E. Lagerstam, and L. Ventä-Olkkonen. User evaluation of mobile augmented reality scenarios. *Journal of Ambient Intelligence and Smart Environments*, 4:29–47, 2012.

[47] T. Olsson, E. Lagerstam, T. Kärkkäinen, and K. Väänänen-Vainio-Mattila. Expected user experience of mobile augmented reality services: A user study in the context of shopping centres. *Personal and ubiquitous computing*, 17(2):287–304, 2013.

[48] L. Poretski, J. Lanir, and O. Arazy. Normative tensions in shared augmented reality. *CSCW*, 2018.

[49] N. Raval, A. Srivastava, A. Razeen, K. Lebeck, A. Machanavajjhala, and L. P. Cox. What you mark is what apps see. In *MobiSys*, 2016.

[50] D. Reilly, M. Salimian, B. MacKay, N. Mathiasen, W. K. Edwards, and J. Franz. Secspace: Prototyping usable privacy and security for mixed reality collaborative environments. In *ACM SIGCHI EICS*, 2014.

[51] K. Rematas, I. Kemelmacher-Shlizerman, B. Curless, and S. Seitz. Soccer on your tabletop. In *CVPR*, 2018.

[52] F. Roesner, T. Kohno, and D. Molnar. Security and privacy for augmented reality systems. *Communications of the ACM*, 57(4):88–96, 2014.

[53] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan. User-driven access control: Rethinking permission granting in modern operating systems. In *IEEE Symposium on Security and Privacy*, 2012.

[54] F. Roesner, D. Molnar, A. Moshchuk, T. Kohno, and H. J. Wang. World-driven access control for continuous sensing. In *ACM CCS*, 2014.

[55] N. Sastry, U. Shankar, and D. Wagner. Secure verification of location claims. In *WiSe*, pages 1–10, 2003.

[56] F. Schaub, R. Deyhle, and M. Weber. Password entry usability and shoulder surfing susceptibility on different smartphone platforms. In *MUM*, 2012.

[57] S. D. Scott, M. S. T. Carpendale, and K. M. Inkpen. Territoriality in collaborative tabletop workspaces. In *CSCW*, 2004.

[58] S. D. Scott, K. D. Grant, and R. L. Mandryk. System guidelines for co-located, collaborative work on a tabletop display. In *ECSCW*, 2003.

[59] C. Shen, K. Everitt, and K. Ryall. Ubitable: Impromptu face-to-face collaboration on horizontal interactive surfaces. In *UbiComp*, 2003.

[60] I. Sluganovic, M. Serbec, A. Derek, and I. Martinovic. HoloPair: Securing shared augmented reality using Microsoft HoloLens. In *ACSAC 2017*, 2017.

[61] I. E. Sutherland. A head-mounted three-dimensional display. In *Fall Joint Computer Conference, American Federation of Information Processing Societies*, 1968.

[62] Z. Szalavári, E. Eckstein, and M. Gervautz. Collaborative gaming in augmented reality. In *VRST*, 1998.

[63] D. Takahashi. Spatial raises $8 million for augmented reality collaboration platform, Oct. 2018. `https://venturebeat.com/2018/10/24/spatial-raises-8-million-for-augmented-reality-collaboration-platform/`.

[64] F. Tari, A. Ant Ozok, and S. Holden. A comparison of perceived and real shoulder-surfing risks between alphanumeric and graphical passwords. In *SOUPS*, 2006.

[65] R. Templeman, M. Korayem, D. Crandall, and A. Kapadia. PlaceAvoider: Steering first-person cameras away from sensitive spaces. In *NDSS*, 2014.

[66] R. Tilton. Daydream Labs: positive social experiences in VR. Google, Aug. 2016. `https://www.blog.google/products/google-vr/daydream-labs-positive-social/`.

[67] Ubiquity6. `https://ubiquity6.com`.

[68] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *FMSE*, 2004.

[69] X. Wang, A. Pande, J. Zhu, and P. Mohapatra. STAMP: Enabling privacy-preserving location proofs for mobile users. *IEEE/ACM ToN*, 24(6):3276–3289, 2016.

[70] P. Wijesekera, A. Baokar, L. Tsai, J. Reardon, S. Egelman, D. Wagner, and K. Beznosov. The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences. In *IEEE Symposium on Security and Privacy*, 2017.

[71] M. Wu and R. Balakrishnan. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. In *UIST*, 2003.

[72] Y. Xu, M. Gandy, S. Deen, B. Schrank, K. Spreen, M. Gorbsky, T. White, E. Barba, I. Radu, J. Bolter, and B. MacIntyre. Bragfish: Exploring physical and social interaction in co-located handheld augmented reality games. In *ACE*, 2008.

[73] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu. ViewDroid: Towards obfuscation-resilient mobile application repackaging detection. In *WiSec*, 2014.

[74] W. Zhou, X. Zhang, and X. Jiang. AppInk: Watermarking Android apps for repackaging deterrence. In *ASIA CCS*, 2013.

[75] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party Android marketplaces. In *CODASPY*, 2012.

A Prototype Application Descriptions

In Section 6.1, we describe our assessment of our design's functionality by its ability to flexibly support our representative case study apps. Here, we provide further details on our implemented prototype case study apps.

Paintball. This is a minimalist implementation of the *Paintball* case study. Players can launch red spheres that, upon contact with another player, attach to that other player. All users in the game session can see all of the red spheres. For the purposes of the prototype, we leave out scorekeeping and more advanced game features.

Doc Edit. This is a basic version of *Multi-Team Whiteboards* in which each user accessing content has a personal instantiation of it. Users, by interacting with a simple control panel, create flat rectangular boxes as documents. Documents are location-decoupled, and though they are private and ghosted by default, users can choose to share individual documents with individual users. A user can also turn a document red, modifying the document's contents in a way that ghost documents do not display (for the prototype, this emulates arbitrary content entry, which we do not implement); the user can also delete the document in a group-extended way (i.e., all other users' instances of the document are also deleted).

Cubist Art. This is a simplified version of the *Community Art* case study. Rather than making and manipulating arbitrary objects, users create and control cubes and can choose to share them or not. Although many of the user's possible actions via the control panel are similar to those of Doc Edit, there are several key differences: (1) Cubes are public by default rather than private. (2) Cubes are shared in a location-coupled way rather than location-decoupled. (3) Cubes obey real-world physics instead of being entirely script-controlled. (4) Object deletion is global location-coupled rather than

global location-independent (note, though, that the semantics of location-coupled sharing make these two cases visually equivalent for solely location-coupled objects).

We did not implement the *Soccer Arena* case study, since it does not surface new security, privacy, or functionality requirements not covered by the other case studies. Section 2 provides further analysis.

B Interaction with Existing Design Recommendations

Below we include a further analysis of our module's compatibility with existing design recommendations.

How are they sharing? The HoloLens guidelines list possible sharing scenarios as consisting of presentation, collaboration, or guidance. Our design supports all of these: for instance, a developer can use ShareAR to set appropriate controls such as view-only permissions when a presenter shares content with an audience. Besides the opt-in scenarios that the HoloLens guidelines describe, our design also supports opt-out public content sharing, which we argue should be treated as another important use case for AR.

What is the group size? The HoloLens guidelines remind developers to design for as many users as needed. Our design can support an arbitrary number of users. (In practice, our implementation stores both object IDs and user IDs as 32-bit integers, providing a generous upper bound on its capacity. We examine performance questions in Section 6.3.)

When are they sharing? The HoloLens guidelines ask whether sharing is asynchronous or synchronous. Although we explicitly design ShareAR to support real-time sharing, we do not preclude the possibility of asynchronous sharing. A developer could, for instance, write a replacement network shim layer that relies on a central server for data storage and periodically queries the server for updates.

What devices are they using? In particular, the HoloLens guidelines ask whether AR users might share content with VR users. Although this is outside the scope of this work, we note that there is nothing in principle that fundamentally prevents developers from extending our work into VR as well. More broadly, we note that in principle, the ShareAR design is compatible across any AR HMD platforms that satisfy basic assumptions such as a shared notion of 3D space. (Our implementation is built for the HoloLens and has not been ported to other platforms as of this writing.)

Clip planes near user. HoloLens recommends setting a "plane clipping" distance of 0.85 m so that a user does not see any portions of AR objects that are closer than that in the user's field of view [42]. Plane clipping may conceptually be considered a partial way of enforcing personal space; however, it only affects the view of the user whose space is invaded, and other users still see the object as being close to the user. ShareAR's treatment of owned physical space

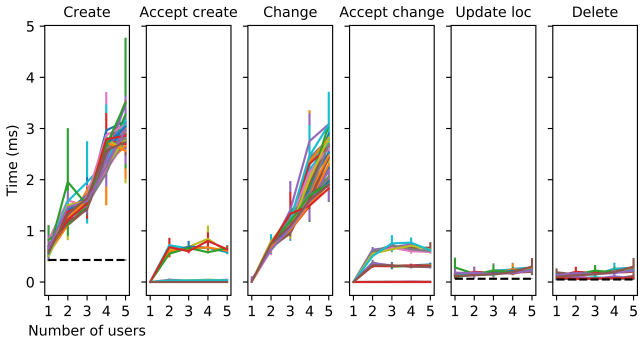

Figure 4: Timing measurements for all steps in the evaluation protocol, each from the perspective of the device initiating the step, as the number of present users scales. Acceptance times are measured on the receiver's device; all other times are measured on the sharer's device. Black dashed lines denote a corresponding baseline Unity operation where one exists.

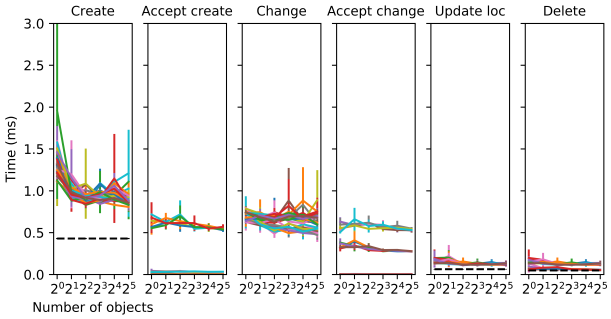

Figure 5: Timing measurements for all steps in the evaluation protocol, each from the perspective of the device initiating the step, on a per-object basis as the number of objects scales. Acceptance times are measured on the receiver's device; all other times are measured on the sharer's device. Black dashed lines denote a corresponding baseline Unity operation where one exists.

encompasses this recommendation and is a more complete solution. (Our implementation does not yet include personal space; however, it does include basic plane clipping.)

Do not disturb. Meta cautions against showing the user incessant notifications. Our design does not specify the user interface: notifications from other devices are passed as an event to the app but not displayed to the user. Thus, ShareAR is flexible enough to support this design choice.

Public by default. This recommendation is similar in spirit to our goal of supporting shared physical-world intuition. Although our design does support a purely public virtual world, we do not recommend it for all circumstances; our ghosting mechanism maintains a basic shared-world intuition while preserving a degree of privacy.

C Detailed Performance Data

Results as the number of users scales are shown in Figure 4; results as the number of objects scales are shown in Figure 5.