usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Toward Strong, Usable Access Control for Shared Distributed Data

Michelle L. Mazurek, Yuan Liang, William Melicher, Manya Sleeper, Lujo Bauer, Gregory R. Ganger, and Nitin Gupta, *Carnegie Mellon University;* Michael K. Reiter, *University of North Carolina at Chapel Hill*

https://www.usenix.org/conference/fast14/technical-sessions/presentation/mazurek

This paper is included in the Proceedings of the
12th USENIX Conference on File and Storage Technologies (FAST '14).

February 17–20, 2014 • Santa Clara, CA USA

ISBN 978-1-931971-08-9

Open access to the Proceedings of the
12th USENIX Conference on File and Storage Technologies (FAST '14)
is sponsored by

Symantec

Toward strong, usable access control for shared distributed data

Michelle L. Mazurek, Yuan Liang, William Melicher, Manya Sleeper,
Lujo Bauer, Gregory R. Ganger, Nitin Gupta, and Michael K. Reiter*
*Carnegie Mellon University, *University of North Carolina at Chapel Hill*

Abstract

As non-expert users produce increasing amounts of personal digital data, usable access control becomes critical. Current approaches often fail, because they insufficiently protect data or confuse users about policy specification. This paper presents Penumbra, a distributed file system with access control designed to match users' mental models while providing principled security. Penumbra's design combines semantic, tag-based policy specification with logic-based access control, flexibly supporting intuitive policies while providing high assurance of correctness. It supports private tags, tag disagreement between users, decentralized policy enforcement, and unforgeable audit records. Penumbra's logic can express a variety of policies that map well to real users' needs. To evaluate Penumbra's design, we develop a set of detailed, realistic case studies drawn from prior research into users' access-control preferences. Using microbenchmarks and traces generated from the case studies, we demonstrate that Penumbra can enforce users' policies with overhead less than 5% for most system calls.

1 Introduction

Non-expert computer users produce increasing amounts of personal digital data, distributed across devices (laptops, tablets, phones, etc.) and the cloud (Gmail, Facebook, Flickr, etc.). These users are interested in accessing content seamlessly from any device, as well as sharing it with others. Thus, systems and services designed to meet these needs are proliferating [6,37,42,43,46,52].

In this environment, access control is critical. News headlines repeatedly feature access-control failures with consequences ranging from embarrassing (e.g., students accessing explicit photos of their teacher on a classroom iPad [24]) to serious (e.g., a fugitive's location being revealed by geolocation data attached to a photo [56]). The potential for such problems will only grow. Yet, at the same time, access-control configuration is a secondary task most users do not want to spend much time on.

Access-control failures generally have two sources: ad-hoc security mechanisms that lead to unforeseen behavior, and policy authoring that does not match users' mental models. Commercial data-sharing services sometimes fail to guard resources entirely [15]; often they manage access in ad-hoc ways that lead to holes [33]. Numerous studies report that users do not understand privacy settings or cannot use them to create desired policies (e.g., [14,25]). Popular websites abound with advice for these confused users [38,48].

Many attempts to reduce user confusion focus only on improving the user interface (e.g., [26,45,54]). While this is important, it is insufficient—a full solution also needs the underlying access-control infrastructure to provide principled security while aligning with users' understanding [18]. Prior work investigating access-control infrastructure typically either does not support the flexible policies appropriate for personal data (e.g., [20]) or lacks an efficient implementation with system-call-level file-system integration (e.g., [31]).

Recent work (including ours) has identified features that are important for meeting users' needs but largely missing in deployed access-control systems: for example, support for semantic policies, private metadata, and interactive policy creation [4,28,44]. In this paper, we present Penumbra, a distributed file system with access control designed to support users' policy needs while providing principled security. Penumbra provides for flexible policy specification meant to support real access-control policies, which are complex, frequently include exceptions, and change over time [8,34,35,44,53]. Because Penumbra operates below the user interface, we do not evaluate it directly with a user study; instead, we develop a set of realistic case studies drawn from prior work and use them for evaluation. We define "usability" for this kind of non-user-facing system as supporting specific policy needs and mental models that have been previously identified as important.

Penumbra's design is driven by three important factors. First, users often think of content in terms of its attributes, or *tags*—photos of my sister, budget spreadsheets, G-rated movies—rather than in traditional hierarchies [28,47,49]. In Penumbra, both content and policy are organized using tags, rather than hierarchically. Second, because tags are central to managing content, they must be treated accordingly. In Penumbra, tags are cryptographically signed first-class objects, specific to a single user's namespace. This allows different users to use different attribute values to describe and make policy about the same content. Most importantly, this design ensures tags used for policy specification are resistant to unauthorized changes and forgery. Policy for accessing tags is set independently of policy for files, allowing for private tags. Third, Penumbra is designed to work in a distributed, decentralized, multi-user environment, in which users access files from various devices without a dedicated central server, an increasingly important environment [47]. We support multi-user devices; although these devices are becoming less common [13], they remain important, particularly in the home [27, 34, 61]. Cloud environments are also inherently multi-user.

This paper makes three main contributions. First, it describes Penumbra, the first file-system access-control architecture that combines semantic policy specification with logic-based credentials, providing an intuitive, flexible policy model without sacrificing correctness. Penumbra's design supports distributed file access, private tags, tag disagreement between users, decentralized policy enforcement, and unforgeable audit records that describe who accessed what content and why that access was allowed. Penumbra's logic can express a variety of flexible policies that map well to real users' needs.

Second, we develop a set of realistic access-control case studies, drawn from user studies of non-experts' policy needs and preferences. To our knowledge, these case studies, which are also applicable to other personal-content-sharing systems, are the first realistic policy benchmarks with which to assess such systems. These case studies capture users' desired policy goals in detail; using them, we can validate our infrastructure's efficacy in supporting these policies.

Third, using our case studies and a prototype implementation, we demonstrate that semantic, logic-based policies can be enforced efficiently enough for the interactive uses we target. Our results show enforcement also scales well with policy complexity.

2 Related work

In this section, we discuss four related areas of research.

Access-control policies and preferences. Users' access-control preferences for personal data are nuanced, dynamic, and context-dependent [3, 35, 44]. Many policies require fine-grained rules, and exceptions are frequent and important [34, 40]. Users want to protect personal data from strangers, but are perhaps more concerned about managing access and impressions among family, friends, and acquaintances [4, 12, 25, 32]. Furthermore, when access-control mechanisms are ill-suited to users' policies or capabilities, they fall back on clumsy, ad-hoc coping mechanisms [58]. Penumbra is designed to support personal polices that are complex, dynamic, and drawn from a broad range of sharing preferences.

Tags for access control. Penumbra relies on tags to define access-control policies. Researchers have prototyped tag-based access-control systems for specific contexts, including web photo albums [7], corporate desktops [16], microblogging services [17], and encrypting portions of legal documents [51]. Studies using role-playing [23] and users' own tags [28] have shown that tag-based policies are easy to understand and accurate policies can be created from existing tags.

Tags for personal distributed file systems. Many distributed file systems use tags for file management, an idea introduced by Gifford et al. [22]. Many suggest tags will eclipse hierarchical management [49]. Several systems allow tag-based file management, but do not explicitly provide access control [46, 47, 52]. Homeviews provides capability-based access control, but remote files are read-only and each capability governs files local to one device [21]. In contrast, Penumbra provides more principled policy enforcement and supports policy that applies across devices. Cimbiosys offers partial replication based on tag filtering, governed by fixed hierarchical access-control policies [60]. Research indicates personal policies do not follow this fixed hierarchical model [34]; Penumbra's more flexible logic builds policies around non-hierarchical, editable tags, and does not require a centralized trusted authority.

Logic-based access control. An early example of logic-based access control is Taos, which mapped authentication requests to proofs [59]. Proof-carrying authentication (PCA) [5], in which proofs are submitted together with requests, has been applied in a variety of systems [9, 11, 30]. PCFS applies PCA to a local file system and is evaluated using a case study based on government policy for classified data [20]. In contrast, Penumbra supports a wider, more flexible set of distributed policies targeting personal data. In addition, while PCFS relies on constructing and caching proofs prior to access, we consider the efficiency of proof generation.

One important benefit of logic-based access control is meaningful auditing; logging proofs provides unforgeable evidence of which policy credentials were used to allow access. This can be used to reduce the trusted computing base, to assign blame for unintended accesses, and to help users detect and fix policy misconfigurations [55].

3 System overview

This section describes Penumbra's architecture as well as important design choices.

3.1 High-level architecture

Penumbra encompasses an ensemble of devices, each storing files and tags. Users on one device can remotely access files and tags on other devices, subject to access control. Files are managed using semantic (i.e., tag-based) object naming and search, rather than a directory hierarchy. Users query local and remote files using tags, e.g., *type=movie* or *keyword=budget*. Access-control policy is also specified semantically, e.g., Alice might allow Bob to access files with the tags *type=photo* and *album=Hawaii*. Our concept of devices can be extended to the cloud environment. A cloud service can be thought of as a large multi-user device, or each cloud user as being assigned her own logical "device." Each user runs a *software agent*, associated with both her global public-key identity and her local uid, on every device she uses. Among other tasks, the agent stores all the *authorization credentials*, or cryptographically signed statements made by principals, that the user has received.

Each device in the ensemble uses a file-system-level *reference monitor* to control access to files and tags. When a system call related to accessing files or tags is received, the monitor generates a challenge, which is formatted as a logical statement that can be proved true only if the request is allowed by policy. To gain access, the requesting user's agent must provide a logical proof of the challenge. The reference monitor will verify the proof before allowing access. To make a proof, the agent assembles a set of relevant authorization credentials. The credentials, which are verifiable and unforgeable, are specified as formulas in an access-control logic, and the proof is a derivation demonstrating that the credentials are sufficient to allow access. Penumbra uses an intuitionistic first-order logic with predicates and quantification over base types, described further in Sections 3.3 and 4.

The challenges generated by the reference monitors have seven types, which fall into three categories: authority to read, write, or delete an existing file; authority to read or delete an existing tag; and authority to create content (files or tags) on the target device. The rationale for this is explained in Section 3.2. Each challenge includes a nonce to prevent replay attacks; for simplicity, we omit the nonces in examples. The logic is not exposed directly to users, but abstracted by an interface that is beyond the scope of this paper.

For both local and remote requests, the user must prove to her local device that she is authorized to access the content. If the content is remote, the local device (acting as client) must additionally prove to the remote device that the local device is trusted to store the content and enforce policy about it. This ensures that users of untrusted devices cannot circumvent policy for remote data. Figure 1 illustrates a remote access.

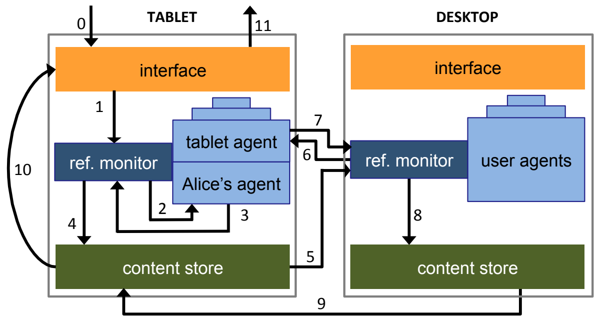

Figure 1: Access-control example. (0) Using her tablet, Alice requests to open a file stored on the desktop. (1) The interface component forwards this request to the reference monitor. (2) The local monitor produces a challenge, which (3) is proved by Alice's local agent, then (4) asks the content store for the file. (5) The content store requests the file from the desktop, (6) triggering a challenge from the desktop's reference monitor. (7) Once the tablet's agent proves the tablet is authorized to receive the file, (8) the desktop's monitor instructs the desktop's content store to send it to the tablet. (9–11) The tablet's content store returns the file to Alice via the interface component.

3.2 Metadata

Semantic management of access-control policy, in addition to file organization, gives new importance to tag handling. Because we base policy on tags, they must not be forged or altered without authorization. If Alice gives Malcolm access to photos from her Hawaiian vacation, he can gain unauthorized access to her budget if he can change its type from *spreadsheet* to *photo* and add the tag *album=Hawaii*. We also want to allow users to keep tags private and to disagree about tags for a shared file.

To support private tags, we treat each tag as an object independent of the file it describes. Reading a tag requires a proof of access, meaning that assembling a file-access proof that depends on tags will often require first assembling proofs of access to those tags (Figure 2).

For tag integrity and to allow users to disagree about tags, we implement tags as cryptographically signed credentials of the form *principal* signed tag(*attribute*, *value*, *file*). For clarity in examples, we use descriptive file names; in reality, Penumbra uses globally unique IDs. For example, Alice can assign the song "Thriller" a four-star rating by signing a credential: Alice signed tag(rating, 4, "Thriller"). Alice, Bob, and Caren can each assign different ratings to "Thriller." Policy specification takes this into account: if Alice grants Bob permission to listen to songs where Alice's rating is three stars or higher, Bob's rating is irrelevant. Because tags are signed, any principal is free to make any tag about any file. Principals

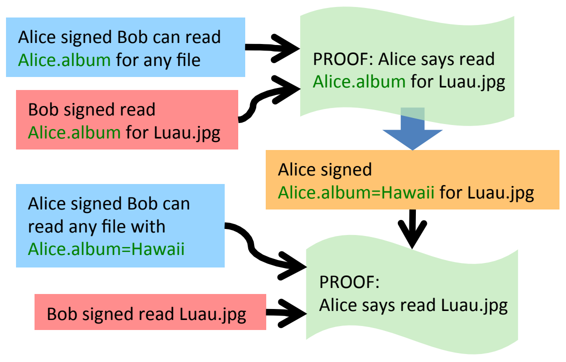
Alice signed Bob can read Alice.album for any file
Bob signed read Alice.album for Luau.jpg
PROOF: Alice says read Alice.album for Luau.jpg
Alice signed Alice.album=Hawaii for Luau.jpg
Alice signed Bob can read any file with Alice.album=Hawaii
Bob signed read Luau.jpg
PROOF: Alice says read Luau.jpg

Figure 2: Example two-stage proof of access, expressed informally. In the first stage, Bob's agent asks which album Alice has placed the photo Luau.jpg in. After making the proof, Bob's agent receives a metadata credential saying the photo is in the album *Hawaii*. By combining this credential with Bob's authority to read some files, Bob's agent can make a proof that will allow Bob to open Luau.jpg.

can be restricted from storing tags on devices they do not own, but if Alice is allowed to create or store tags on a device then those tags may reference any file.

Some tags are naturally written as attribute-value pairs (e.g., *type=movie*, *rating=PG*). Others are commonly value-only (e.g., photos tagged with *vacation* or with people's names). We handle all tags as name-value pairs; value-only tags are transformed into name-value pairs, e.g., from "vacation" to *vacation=true*.

Creating tags and files. Because tags are cryptographically signed, they cannot be updated; instead, the old credential is revoked (Section 4.4) and a new one is issued. As a result, there is no explicit write-tag authority.

Unlike reading and writing, in which authority is determined per file or tag, authority to create files and tags is determined per device. Because files are organized by their attributes rather than in directories, creating one file on a target device is equivalent to creating any other. Similarly, a user with authority to create tags can always create any tag in her own namespace, and no tags in any other namespace. So, only authority to create any tags on the target device is required.

3.3 Devices, principals, and authority

We treat both users and devices as principals who can create policy and exercise authority granted to them. Each principal has a public-private key pair, which is consistent across devices. This approach allows multi-user devices and decisions based on the combined trustworthiness of a user and a device. (Secure initial distribution of a user's private key to her various devices is outside the scope of this paper.)

Access-control logics commonly use $A \text{ signed } F$ to describe a principal cryptographically asserting a statement F. $A \text{ says } F$ describes beliefs or assertions F that can be derived from other statements that A has signed or, using modus ponens, other statements that A believes (says):

$$\frac{A \text{ says } F \quad A \text{ says } (F \to G)}{A \text{ says } G}$$

Statements that principals can make include both delegation and use of authority. In the following example, principal A grants authority over some action F to principal B, and B wants to perform action F.

$$A \text{ signed deleg } (B,\ F) \qquad (1)$$
$$B \text{ signed } F \qquad (2)$$

These statements can be combined, as a special case of modus ponens, to prove that B's action is supported by A's authority:

$$\frac{(1) \quad (2)}{A \text{ says } F}$$

Penumbra's logic includes these rules, other constructions commonly used in access control (such as defining groups of users), and a few minor additions for describing actions on files and tags (see Section 4).

In Penumbra, the challenge statements issued by a reference monitor are of the form $\mathit{device} \text{ says } \mathit{action}$, where action describes the access being attempted. For Alice to read a file on her laptop, her software agent must prove that $\mathit{AliceLaptop} \text{ says readfile}(f)$.

This design captures the intuition that a device storing some data ultimately controls who can access it: sensitive content should not be given to untrusted devices, and trusted devices are tasked with enforcing access-control policy. For most single-user devices, a default policy in which the device delegates all of its authority to its owner is appropriate. For shared devices or other less common situations, a more complex device policy that gives no user full control may be necessary.

3.4 Threat model

Penumbra is designed to prevent unauthorized access to files and tags. To prevent spoofed or forged proofs, we use nonces to prevent replay attacks and rely on standard cryptographic assumptions that signatures cannot be forged unless keys are leaked. We also rely on standard network security techniques to protect content from observation during transit between devices.

Penumbra employs a language for capturing and reasoning about trust assertions. If trust is misplaced, violations of intended policy may occur—for example, an authorized user sending a copy of a file to an unauthorized user. In contrast to other systems, Penumbra's flexibility allows users to encode limited trust precisely, minimizing vulnerability to devices or users who prove untrustworthy; for example, different devices belonging to the same owner can be trusted differently.

4 Expressing semantic policies

This section describes how Penumbra expresses and enforces semantic policies with logic-based access control.

4.1 Semantic policy for files

File accesses incur challenges of the form *device* says *action*(f), where f is a file and *action* can be one of readfile, writefile, or deletefile.

A policy by which Alice allows Bob to listen to any of her music is implemented as a conditional delegation: If Alice says a file has *type=music*, then Alice delegates to Bob authority to read that file. We write this as follows:

$$\text{Alice signed } \forall f: \text{tag(type,music,}f) \rightarrow \text{deleg(Bob,readfile(}f)) \quad (3)$$

To use this delegation to listen to "Thriller," Bob's agent must show that Alice says "Thriller" has *type=music*, and that Bob intends to open "Thriller" for reading, as follows:

$$\text{Alice signed tag(type,music,“Thriller”)} \quad (4)$$
$$\text{Bob signed readfile(“Thriller”)} \quad (5)$$

$$\frac{\dfrac{(3) \qquad (4)}{\text{Alice says deleg(Bob,readfile(“Thriller”))}} \quad (5)}{\text{Alice says readfile(“Thriller”)}}$$

In this example, we assume Alice's devices grant her access to all of her files; we elide proof steps showing that the device assents once Alice does. We similarly elide instantiation of the quantified variable.

We can easily extend such policies to multiple attributes or to groups of people. To allow the group "co-workers" to view her vacation photos, Alice would assign users to the group (which is also a principal) by issuing credentials as follows:

$$\text{Alice signed speaksfor(Bob, Alice.co-workers)} \quad (6)$$

Then, Alice would delegate authority to the group rather than to individuals:

$$\text{Alice signed } \forall f: \text{tag(type,music,}f) \rightarrow \text{deleg(Alice.co-workers,readfile(}f)) \quad (7)$$

4.2 Policy about tags

Penumbra supports private tags by requiring a proof of access before allowing a user or device to read a tag. Because tags are central to file and policy management, controlling access to them without impeding file system operations is critical.

Tag policy for queries. Common accesses to tags fall into three categories. A *listing query* asks which files belong to a category defined by one or more attributes, e.g., list all Alice's files with *type=movie* and *genre=comedy*. An *attribute query* asks the value of an attribute for a specific file, e.g., the name of the album to which a photo belongs. This kind of query can be made directly by users or by their software agents as part of two-stage proofs (Figure 2). A *status query*, which requests all the system metadata for a given file—last modify time, file size, etc.—is a staple of nearly every file access in most file systems (e.g., the POSIX *stat* system call).

Tag challenges have the form *device* says *action*(*attribute list*,*file*), where *action* is either readtags or deletetags. An *attribute list* is a set of (principal,attribute,value) triples representing the tags for which access is requested. Because tag queries can apply to multiple values of one attribute or multiple files, we use the wildcard * to indicate all possible completions. The listing query example above, which is a search on multiple files, would be specified with the attribute list [(Alice,type,movie), (Alice,genre,comedy)] and the target file *. The attribute query example identifies a specific target file but not a specific attribute value, and could be written with the attribute list [(Alice,album,*)] and target file "Luau.jpg." A status query for the same file would contain an attribute list like [(AliceLaptop,*,*)].

Credentials for delegating and using authority in the listing query example can be written as:

$$\text{Alice signed } \forall f: \text{deleg(Bob,readtags(}[\text{(Alice,type,movie),(Alice,genre,comedy)}],f)) \quad (8)$$
$$\text{Bob signed readtags(}[\text{(Alice,type,movie),(Alice,genre,comedy)}],*) \quad (9)$$

These credentials can be combined to prove Bob's authority to make this query.

Implications of tag policy. One subtlety inherent in tag-based delegation is that delegations are not separable. If Alice allows Bob to list her Hawaii photos (e.g., files with *type=photo* and *album=Hawaii*), that should not imply that he can list all her photos or non-photo files related to Hawaii. However, tag delegations should be additive: a user with authority to list all photos and authority to list all Hawaii files could manually compute the intersection of the results, so a request for Hawaii photos should be allowed. Penumbra supports this subtlety.

Another interesting issue is limiting the scope of queries. Suppose Alice allows Bob to read the album name only when *album=Hawaii*, and Bob wants to know the album name for "photo127." If Bob queries the album name regardless of its value (attributelist[(Alice,album,*)]), no proof can be made and the request will fail. If Bob limits his request to the attribute list [(Alice,album,Hawaii)], the proof succeeds. If "photo127" is not in the Hawaii album, Bob cannot learn which album it is in.

Users may sometimes make broader-than-authorized queries: Bob may try to list all of Alice's photos when he only has authority for Hawaii photos. Bob's agent will then be asked for a proof that cannot be constructed. A straightforward option is for the query to simply fail. A better outcome is for Bob to receive an abridged list containing only Hawaii photos. One way to achieve this is for Bob's agent to limit his initial request to something the agent can prove, based on available credentials—in this case, narrowing its scope from all photos to Hawaii photos. We defer implementing this to future work.

4.3 Negative policies

Negative policies, which forbid access rather than allow it, are important but often challenging for access-control systems. Without negative policies, many intuitively desirable rules are difficult to express. Examples taken from user studies include denying access to photos tagged with *weird* or *strange* [28] and sharing all files other than financial documents [34].

The first policy could naively be formulated as forbidding access to files tagged with *weird=true*; or as allowing access when the tag *weird=true* is not present. In our system, however, policies and tags are created by many principals, and there is no definitive list of all credentials. In such contexts, the inability to find a policy or tag credential does not guarantee that no such credential exists; it could simply be located somewhere else on the network. In addition, policies of this form could allow users to make unauthorized accesses by interrupting the transmission of credentials. Hence, we explore alternative ways of expressing deny policies.

Our solution has two parts. First, we allow delegation based on tag inequality: for example, to protect financial documents, Alice can allow Bob to read any file with *topic≠financial*. This allows Bob to read a file if his agent can find a tag, signed by Alice, placing that file into a topic other than financial. If no credential is found, access is still denied, which prevents unauthorized access via credential hiding. This approach works best for tags with non-overlapping values—e.g., restricting children to movies not rated R. If, however, a file is tagged with both *topic=financial* and *topic=vacation*, then this approach would still allow Bob to access the file.

To handle situations with overlapping and less-well-defined values, e.g., denying access to weird photos, Alice can grant Bob authority to view files with *type=photo* and *weird=false*. In this approach, every non-weird photo must be given the tag *weird=false*. This suggests two potential difficulties. First, we cannot ask the user to keep track of these negative tags; instead, we assume the user's policymaking interface will automatically add them (e.g., adding *weird=false* to any photo the user has not marked with *weird=true*). As we already assume the interface tracks tags to help the user maintain consistent labels and avoid typos, this is not an onerous requirement. Second, granting the ability to view files with *weird=false* implicitly leaks the potentially private information that some photos are tagged *weird=true*. We assume the policymaking interface can obfuscate such negative tags (e.g., by using a hash value to obscure *weird*), and maintain a translation to the user's original tags for purposes of updating and reviewing policy and tags. We discuss the performance impact of adding tags related to the negative policy (e.g., *weird=false*) in Section 7.

4.4 Expiration and revocation

In Penumbra, as in similar systems, the lifetime of policy is determined by the lifetimes of the credentials that encode that policy. To support dynamic policies and allow policy changes to propagate quickly, we have two fairly standard implementation choices.

One option is short credential lifetimes: the user's agent can be set to automatically renew each short-lived policy credential until directed otherwise. Alternatively, we can require all credentials used in a proof to be online countersigned, confirming validity [29]. Revocation is then accomplished by informing the countersigning authority. Both of these options can be expressed in our logic; we do not discuss them further.

5 Realistic policy examples

We discussed abstractly how policy needs can be translated into logic-based credentials. We must also ensure that our infrastructure can represent real user policies.

It is difficult to obtain real policies from users for new access-control capabilities. In lab settings, especially without experience to draw on, users struggle to articulate policies that capture real-life needs across a range of scenarios. Thus, there are no applicable standard policy or file-sharing benchmarks. Prior work has often, instead, relied on researcher experience or intuition [41,46,52,60]. Such an approach, however, has limited ability to capture the needs of non-expert users [36].

To address this, we develop the first set of access-control-policy case studies that draw from target users' needs and preferences. They are based on detailed results from in-situ and experience-sampling user studies [28, 34] and were compiled to realistically represent diverse policy needs. These case studies, which could also be used to evaluate other systems in this domain, are an important contribution of this work.

We draw on the HCI concept of *persona* development. Personas are archetypes of system users, often created to guide system design. Knowledge of these personas' characteristics and behaviors informs tests to ensure an application is usable for a range of people. Specifying

An access-control system should support ...	Sources	Case study
access-control policies on metadata	[4, 12]	All
policies for potentially overlapping groups of people, with varied granularity (e.g., family, subsets of friends, strangers, “known threats”)	[4, 12, 25, 40, 44, 50]	All
policies for potentially overlapping groups of items, with varied granularity (e.g., health information, “red flag” items)	[25, 34, 40, 44]	All
photo policies based on photo location., people in photo	[4, 12, 28]	Jean, Susie
negative policies to restrict personal or embarrassing content	[4, 12, 28, 44]	Jean, Susie
policy inheritance for new and modified items	[4, 50]	All
hiding unshared content	[35, 44]	All
joint ownership of files	[34, 35]	Heather/Matt
updating policies and metadata	[4, 12, 50]	—

Table 1: Access control system needs from literature.

individuals with specific needs provides a face to types of users and focuses design and testing [62].

To make the case studies sufficiently concrete for testing, each includes a set of users and devices, as well as policy rules for at least one user. Each also includes a simulated trace of file and metadata actions; some actions loosely mimic real accesses, and others test specific properties of the access-control infrastructure. Creating this trace requires specifying many variables, including policy and access patterns, the number of files of each type, specific tags (access-control or otherwise) for each file, and users in each user group. We determine these details based on user-study data, and, where necessary, on inferences informed by HCI literature and consumer market research (e.g., [2, 57]). In general, the access-control policies are well-grounded in user-study data, while the simulated traces are more speculative.

In line with persona development [62], the case studies are intended to include a range of policy needs, especially those most commonly expressed, but not to completely cover all possible use cases. To verify coverage, we collated policy needs discussed in the literature. Table 1 presents a high-level summary. The majority of these needs are at least partially represented in all of our case studies. Unrepresented is only the ability to update policies and metadata over time, which Penumbra supports but we did not include in our test cases. The diverse policies represented by the case studies can all be encoded in Penumbra; this provides evidence that our logic is expressive enough to meet users' needs.

Case study 1: Susie. This case (Figure 3), drawn from a study of tag-based access control for photos [28], captures a default-share mentality: Susie is happy to share most photos widely, with the exception of a few containing either highly personal content or pictures of children she works with. As a result, this study exercises several somewhat-complex negative policies. This study focuses exclusively on Susie's photos, which she accesses from several personal devices but which other users access only via simulated “cloud” storage. No users besides Susie have write access or the ability to create files and tags. Because the original study collected detailed information on photo tagging and policy preferences, both the tagging and the policy are highly accurate.

Case study 2: Jean. This case study (Figure 3) is drawn from the same user study as Susie. Jean has a default-protect mentality; she only wants to share photos with people who are involved in them in some way. This includes allowing people who are tagged in photos to see those photos, as well as allowing people to see photos from events they attended, with some exceptions. Her policies include some explicit access-control tags—for example, restricting photos tagged *goofy*—as well as hybrid tags that reflect content as well as policy. As with the Susie case study, this one focuses exclusively on Jean's photos, which she accesses from personal devices and others access from a simulated “cloud.” Jean's tagging scheme and policy preferences are complex; this case study includes several examples of the types of tags and policies she discussed, but is not comprehensive.

Case study 3: Heather and Matt. This case study (Figure 3) is drawn from a broader study of users' access-control needs [34]. Heather and Matt are a couple with a young daughter; most of the family's digital resources are created and managed by Heather, but Matt has full access. Their daughter has access to the subset of content appropriate for her age. The couple exemplifies a default-protect mentality, offering only limited, identified content to friends, other family members, and co-workers. This case study includes a wider variety of content, including photos, financial documents, work documents, and entertainment media. The policy preferences reflect Heather and Matt's comments; the assignment of non-access-control-related tags is less well-grounded, as they were not explicitly discussed in the interview.

Case study 4: Dana. This case study (Figure 3) is drawn from the same user study as Heather and Matt. Dana is a law student who lives with a roommate and has a strong default-protect mentality. She has confidential documents related to a law internship that must be

SUSIE

Individuals: Susie, mom
Groups: friends, acquaintances, older friends, public
Devices: laptop, phone, tablet, cloud
Tags per photo: 0-2 access-control, 1-5 other
Policies:
Friends can see all photos.
Mom can see all photos except mom-sensitive.
Acquaintances can see all photos except personal, very personal, or red flag.
Older friends can see all photos except red flag.
Public can see all photos except personal, very personal, red flag, or kids.

JEAN

Individuals: Jean, boyfriend, sister, Pat, supervisor, Dwight
Groups: volunteers, kids, acquaintances
Devices: phone, two cloud services
Tags per photo: 1-10, including mixed-use access control
Policies:
Anyone can see photos they are in.
Kids can only see kids photos.
Dwight can see photos of his wife.
Supervisor can see work photos.
Volunteers can see volunteering photos.
Boyfriend can see boyfriend, family reunion, and kids photos.
Acquaintances can see beautiful photos.
No one can see goofy photos.

HEATHER AND MATT

Individuals: Heather, Matt, daughter
Groups: friends, relatives, co-workers, guests
Devices: laptop, two phones, DVR, tablet
Tags per item: 1-3, including mixed-use access control
Policies:
Heather and Matt can see all files
Co-workers can see all photos and music
Friends and relatives can see all photos, TV shows, and music
Guests can see all TV shows and music
Daughter can see all photos; music, TV except inappropriate
Heather can update all files except TV shows
Matt can update TV shows

DANA

Individuals: Dana, sister, mom, boyfriend, roommate, boss
Groups: colleagues, friends
Devices: laptop, phone, cloud service
Tags per item: 1-3, including mixed-use access control
Policies:
Boyfriend and sister can see all photos
Friends can see favorite photos
Boyfriend, sister, friends can see all music and TV shows
Roommate can read and write household documents
Boyfriend and mom can see health documents
Boss can read and write all work documents
Colleagues can read and write work documents per project

Figure 3: Details of the four case studies

protected. This case study includes documents related to work, school, household management, and personal topics like health, as well as photos, e-books, television shows, and music. The policy preferences closely reflect Dana's comments; the non-access-control tags are drawn from her rough descriptions of the content she owns.

6 Implementation

This section describes our Penumbra prototype.

6.1 File system implementation

Penumbra is implemented in Java, on top of FUSE [1]. Users interact normally with the Linux file system; FUSE intercepts system calls related to file operations and redirects them to Penumbra. Instead of standard file paths, Penumbra expects semantic queries. For example, a command to list G-rated movies can be written 'ls "query:Alice.type=movie & Alice.rating=G".'

Figure 4 illustrates Penumbra's architecture. System calls are received from FUSE in the front-end interface, which also parses the semantic queries. The central controller invokes the reference monitor to create challenges and verify proofs, user agents to create proofs, and the file and (attribute) database managers to provide protected content. The controller uses the communications module to transfer challenges, proofs, and content between devices. We also implement a small, short-term authority cache in the controller. This allows users who

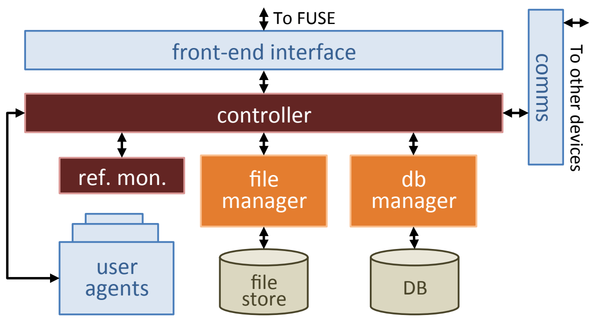

Figure 4: System architecture. The primary TCB (controller and reference monitor) is shown in red (darkest). The file and database managers (medium orange) also require some trust.

have recently proved access to content to access that content again without submitting another proof. The size and expiration time of the cache can be adjusted to trade off proving time with faster response to policy updates.

The implementation is about 15,000 lines of Java and 1800 lines of C. The primary trusted computing base (TCB) includes the controller (1800 lines) and the reference monitor (2500 lines)—the controller guards access to content, invoking the reference monitor to create challenges and verify submitted proofs. The file manager (400 lines) must be trusted to return the correct content for each file and to provide access to files only through the controller. The database manager (1600 lines) similarly must be trusted to provide access to tags only through the controller and to return only the requested

System call	Required proof(s)
mknod	create file, create metadata
open	read file, write file
truncate	write file
utime	write file
unlink	delete file
getattr	read tags: (system, *, *)
readdir	read tags: attribute list for *
getxattr	read tags: (principal, attribute, *)
setxattr	create tags
removexattr	delete tags: (principal, attribute, *)

Table 2: Proof requirements for file-related system calls

tags. The TCB also includes 145 lines of LF (logical framework) specification defining our logic.

Mapping system calls to proof goals. Table 2 shows the proof(s) required for each system call. For example, calling readdir is equivalent to a listing query—asking for all the files that have some attribute(s)—so it must incur the appropriate read-tags challenge.

Using "touch" to create a file triggers four system calls: getattr (the FUSE equivalent of stat), mknod, utime, and another getattr. Each getattr is a status query (see Section 4.2) and requires a proof of authority to read system tags. The mknod call, which creates the file and any initial metadata set by the user, requires proofs of authority to create files and metadata. Calling utime instructs the device to update its tags about the file. Updated system metadata is also a side effect of writing to a file, so we map utime to a write-file permission.

Disconnected operation. When a device is not connected to the Penumbra ensemble, its files are not available. Currently, policy updates are propagated immediately to all available devices; if a device is not available, it misses the new policy. While this is obviously impractical, it can be addressed by implementing eventual consistency (see for example Perspective [47] or Cimbiosys [43]) on top of the Penumbra architecture.

6.2 Proof generation and verification

Users' agents construct proofs using a recursive theorem prover loosely based on the one described by Elliott and Pfenning [19]. The prover starts from the goal (the challenge statement provided by the verifier) and works backward, searching through its store of credentials for one that either proves the goal directly or implies that if some additional goal(s) can be proven, the original goal will also be proven. The prover continues recursively solving these additional goals until either a solution is reached or a goal is found to be unprovable, in which case the prover backtracks and attempts to try again with another credential. When a proof is found, the prover returns it in a format that can be submitted to the reference monitor for checking. The reference monitor uses a standard LF checker implemented in Java.

The policy scenarios represented in our case studies generally result in a shallow but wide proof search: for any given proof, there are many irrelevant credentials, but only a few nested levels of additional goals. In enterprise or military contexts with strictly defined hierarchies of authority, in contrast, there may be a deeper but narrower structure. We implement some basic performance improvements for the shallow-but-wide environment, including limited indexing of credentials and simple fork-join parallelism, to allow several possible proofs to be pursued simultaneously. These simple approaches are sufficient to ensure that most proofs complete quickly; eliminating the long tail in proving time would require more sophisticated approaches, which we leave to future work.

User agents build proofs using the credentials of which they are aware. Our basic prototype pushes all delegation credentials to each user agent. (Tag credentials are guarded by the reference monitor and not automatically shared.) This is not ideal, as pushing unneeded credentials may expose sensitive information and increase proving time. However, if credentials are not distributed automatically, agents may need to ask for help from other users or devices to complete proofs (as in [9]); this could make data access slower or even impossible if devices with critical information are unreachable. Developing a strategy to distribute credentials while optimizing among these tradeoffs is left for future work.

7 Evaluation

To demonstrate that our design can work with reasonable efficiency, we evaluated Penumbra using the simulated traces we developed as part of the case studies from Section 5 as well as three microbenchmarks.

7.1 Experimental setup

We measured system call times in Penumbra using the simulated traces from our case studies. Table 3 lists features of the case studies we tested. We added users to each group, magnifying the small set of users discussed explicitly in the study interview by a factor of five. The set of files was selected as a weighted-random distribution among devices and access-control categories. For each case study, we ran a parallel control experiment with access control turned off—all access checks succeed immediately with no proving. These comparisons account for the overheads associated with FUSE, Java, and our database accesses—none of which we aggressively optimized—allowing us to focus on the overhead

Case study	Users	Files	Deleg. creds.	Proofs	System calls
Susie	60	2,349	68	46,646	212,333
Jean	65	2,500	93	30,755	264,924
Heather/Matt	60	3,098	101	39,732	266,501
Dana	60	3,798	89	27,859	74,593

Table 3: Case studies we tested. Proof and system call counts are averaged over 10 runs.

of access control. We ran each case study 10 times with and 10 times without access control.

During each automated run, each device in the case study was mounted on its own four-core (eight-thread) 3.4GHz Intel i7-4770 machine with 8GB of memory, running Ubuntu 12.04.3 LTS. The machines were connected on the same subnet via a wired Gigabit-Ethernet switch; 10 pings across each pair of machines had minimum, maximum, and median round-trip times of 0.16, 0.37, and 0.30 ms. Accounts for the people in the case study were created on each machine; these users then created the appropriate files and added a weighted-random selection of tags. Next, users listed and opened a weighted-random selection of files from those they were authorized to access. The weights are influenced by research on how the age of content affects access patterns [57]. Based on the file type, users read and wrote all or part of each file's content before closing it and choosing another to access. The specific access pattern is less important than broadly exercising the desired policy. Finally, each user attempted to access forbidden content to validate that the policy was set correctly and measure timing for failed accesses.

7.2 System call operations

Adding theorem proving to the critical path of file operations inevitably reduces performance. Usability researchers have found that delays of less than 100 ms are not noticeable to most users, who perceive times less than that as instantaneous [39]. User-visible operations consist of several combined system calls, so we target system call operation times well under the 100 ms limit.

Figure 5 shows the duration distribution for each system call, aggregated across all runs of all case studies, both with and without access control. Most system calls were well under the 100 ms limit, with medians below 2 ms for getattr, open, and utime and below 5 ms for getxattr. Medians for mknod and setxattr were 20 ms and 25 ms. That getattr is fast is particularly important, as it is called within nearly every user operation. Unfortunately, readdir (shown on its own axis for scale) did not perform as well, with a median of 66 ms. This arises from a combination of factors: readdir performs the most proofs (one local, plus one per remote device); polls each

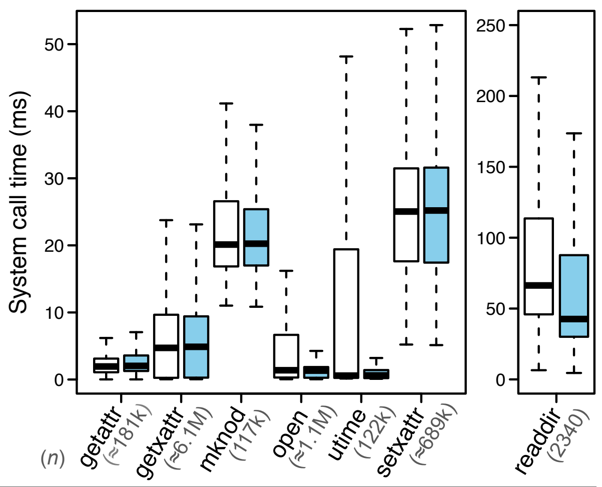

Figure 5: System call times with (white, left box of each pair) and without (shaded, right) access control, with the number of operations (n) in parentheses. ns vary up to 2% between runs with and without access control. Other than readdir (shown separately for scale), median system call times with access control are 1-25 ms and median overhead is less than 5%.

remote device; and must sometimes retrieve thousands of attributes from our mostly unoptimized database on each device. In addition, repeated readdirs are sparse in our case studies and so receive little benefit from proof caching. The results also show that access-control overhead was low across all system calls. For open and utime, the access control did not affect the median but did add more variance.

In general, we did little optimization on our simple prototype implementation; that most of our operations already fall well within the 100 ms limit is encouraging. In addition, while this performance is slower than for a typical local file system, longer delays (especially for remote operations like readdir) may be more acceptable for a distributed system targeting interactive data sharing.

7.3 Proof generation

Because proof generation is the main bottleneck inherent to our logic-based approach, it is critical to understand the factors that affect its performance. Generally system calls can incur up to four proofs (local and remote, for the proofs listed in Table 2). Most, however, incur fewer—locally opening a file for reading, for example, incurs one proof (or zero, if permission has already been cached). The exception is readdir, which can incur one local proof plus one proof for each device from which data is requested. However, if authority has already been cached no proof is required. (For these tests, authority cache entries expired after 10 minutes.)

Proving depth. Proving time is affected by proving depth, or the number of subgoals generated by the prover along one search path. Upon backtracking, proving depth decreases, then increases again as new paths are explored. Examples of steps that increase proving depth include using a delegation, identifying a member of a group, and solving the "if" clause of an implication. Although in corporate or military settings proofs can sometimes extend deeply through layers of authority, policies for personal data (as exhibited in the user studies we considered) usually do not include complex redelegation and are therefore generally shallow. In our case studies, the maximum proving depth (measured as the greatest depth reached during proof search, not the depth of the solution) was only 21; 11% of observed proofs (165,664 of 1,468,222) had depth greater than 10.

To examine the effects of proving depth, we developed a microbenchmark that tests increasingly long chains of delegation between users. We tested chains up to 60 levels deep. As shown in Figure 6a, proving time grew linearly with depth, but with a shallow slope—at 60 levels, proving time remained below 6 ms.

Red herrings. We define a *red herring* as an unsuccessful proving path in which the prover recursively pursues at least three subgoals before detecting failure and backtracking. To examine this, we developed a microbenchmark varying the number of red herrings; each red herring is exactly four levels deep. As shown in Figure 6b, proving time scaled approximately quadratically in this test: each additional red herring forces additional searches of the increasing credential space. In our case studies, the largest observed value was 43 red herrings; proofs with more than 20 red herrings made up only 0.5% of proofs (7,437 of 1,468,222). For up to 20 red herrings, proving time in the microbenchmark was generally less than 5 ms; at 40, it remained under 10 ms.

Proving time in the case studies. In the presence of real policies and metadata, changes in proving depth and red herrings can interact in complex ways that are not accounted for by the microbenchmarks. Figure 7 shows proving time aggregated in two ways. First, we compare case studies. Heather/Matt has the highest variance because files are jointly owned by the couple, adding an extra layer of indirection for many proofs. Susie has a higher median and variance than Dana or Jean because of her negative policies, which lead to more red herrings. Second, we compare proof generation times, aggregated across case studies, based on whether a proof was made by the primary user, by device agents as part of remote operations, or by other users. Most important for Penumbra is that proofs for primary users be fast, as users do not expect delays when accessing their own content; these proofs had a median time less than 0.52 ms in each case study. Also important is that device proofs are fast, as they are an extra layer of overhead on all remote operations. Device proofs had median times of 1.1-1.7 ms for each case study. Proofs for other users were slightly slower, but had medians of 2-9 ms in each case study.

We also measured the time it takes for the prover to conclude no proof can be made. Across all experiments, 1,375,259 instances of failed proofs had median and 90th-percentile times of 9 and 42 ms, respectively.

Finally, we consider the long tail of proving times. Across all 40 case study runs, the 90th-percentile proof time was 10 ms, the 99th was 45 ms, and the maximum was 1531 ms. Of 1,449,920 proofs, 3,238 (0.2%) took longer than 100 ms. These pathological cases may have several causes: high depth, bad luck in red herrings, and even Java garbage collection. Reducing the tail of proving times is an important goal for future work.

Effects of negative policy. Implementing negative policy for attributes without well-defined values (such as the allow *weird=false* example from Section 4.3) requires adding inverse policy tags to many files. A policy with negative attributes needs $n \times m$ extra attribute credentials, where n is the number of negative attributes in the policy and m is the number of affected files.

Users with default-share mentalities who tend to specify policy in terms of exceptions are most affected. Susie, our default-share case study, has five such negative attributes: *personal*, *very personal*, *mom-sensitive*, *red-flag*, and *kids*. Two other case studies have one each: Jean restricts photos tagged *goofy*, while Heather and Matt restrict media files tagged *inappropriate* from their young daughter. Dana, an unusually strong example of the default-protect attitude, has none. We also reviewed detailed policy data from [28] and found that for photos, the number of negative tags ranged from 0 to 7, with median 3 and mode 1. For most study participants, negative tags fall into a few categories: synonyms for private, synonyms for weird or funny, and references to alcohol. A few also identified one or two people who prefer not to have photos of them made public. Two of 18 participants

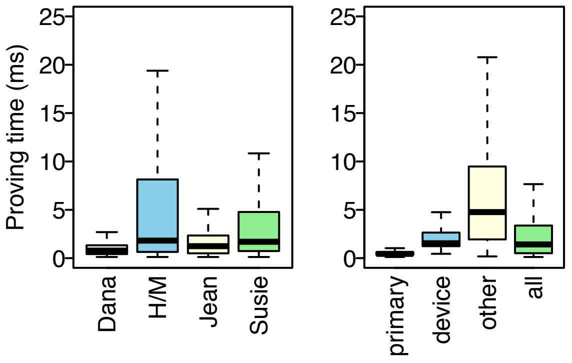

Figure 7: Proving times organized by (left) case study and (right) primary user, device, and other users.

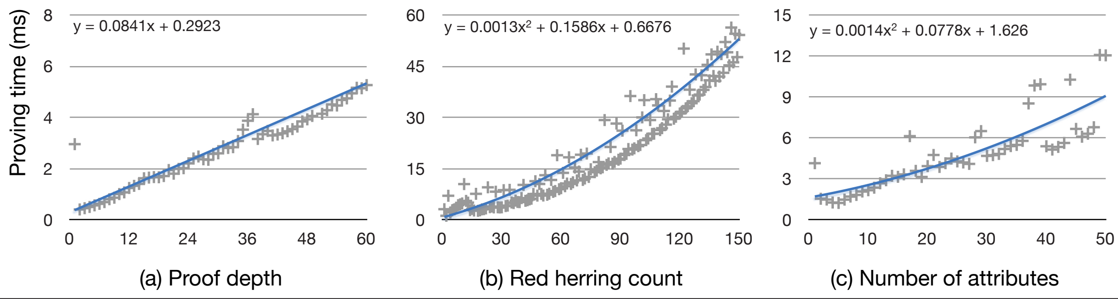

Figure 6: Three microbenchmarks showing how proving time scales with proving depth, red herrings, and attributes-per-policy. Shown with best-fit (a) line and (b,c) quadratic curve.

used a wider range of less general negative tags.

The value of m is determined in part by the complexity of the user's policy: the set of files to which the negative attributes must be attached is the set of files with the positive attributes in the same policy. For example, a policy on files with *type=photo & goofy=false* will have a larger m-value than a policy on files with *type=photo & party=true & goofy=false*.

Because attributes are indexed by file in the prover, the value of n has a much stronger affect on proving time than the value of m. Our negative-policy microbenchmark tests the prover's performance as the number of attributes per policy (and consequently per file) increases.

Figure 6c shows the results. Proving times grew approximately quadratically but with very low coefficients. For policies of up to 10 attributes (the range discussed above), proving time was less than 2.5 ms.

Adding users and devices. Penumbra was designed to support groups of users who share with each other regularly – household members, family, and close friends. Based on user studies, we estimate this is usually under 100 users. Our evaluation (Section 7) examined Penumbra's performance under these and somewhat more challenging circumstances. Adding more users and devices, however, raises some potential challenges.

When devices are added, readdir operations that must visit all devices will require more work; much of this work can be parallelized, so the latency of a readdir should grow sub-linearly in the number of devices. With more users and devices, more files are also expected, with correspondingly more total attributes. The latency of a readdir to an individual device is approximately linear in the number of attributes that are returned. Proving time should scale sub-linearly with increasing numbers of files, as attributes are indexed by file ID; increasing the number of attributes per file should scale linearly as the set of attributes for a given file is searched. Adding users can also be expected to add policy credentials. Users can be added to existing policy groups with sub-linear overhead, but more complex policy additions can have varying effects. If a new policy is mostly disjoint from old policies, it can quickly be skipped during proof search, scaling sub-linearly. However, policies that heavily overlap may lead to increases in red herrings and proof depths; interactions between these could cause proving time to increase quadratically (see Figure 6) or faster. Addressing this problem could require techniques such as pre-computing proofs or subproofs [10], as well as more aggressive indexing and parallelization within proof search to help rule out red herrings sooner.

In general, users' agents must maintain knowledge of available credentials for use in proving. Because they are cryptographically signed, credentials can be up to about 2 kB in size. Currently, these credentials are stored in memory, indexed and preprocessed in several ways, to streamline the proving process. As a result, memory requirements grow linearly, but with a large constant, as credentials are added. To support an order of magnitude more credentials would require revisiting the data structures within the users' agents and carefully considering tradeoffs among insertion time, deletion time, credential matching during proof search, and memory use.

8 Conclusion

Penumbra is a distributed file system with an access-control infrastructure for distributed personal data that combines semantic policy specification with logic-based enforcement. Using case studies grounded in data from user studies, we demonstrated that Penumbra can accommodate and enforce commonly desired policies, with reasonable efficiency. Our case studies can also be applied to other systems in this space.

9 Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. 0946825, CNS-0831407, and DGE-0903659, by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389 and W911NF-09-1-0273 from the Army Research Office, by gifts from Cisco Systems Inc. and Intel, and by Facebook and the ARCS Foundation. We thank the members and companies of the PDL Consortium (including Actifio, APC, EMC, Facebook, Fusion-io, Google, Hewlett-Packard Labs, Hitachi, Huawei, Intel, Microsoft Research, NEC Laboratories, NetApp, Oracle, Panasas, Riverbed, Samsung, Seagate, Symantec, VMware, and Western Digital) for their interest, insights, feedback, and support. We thank Michael Stroucken and Zis Economou for help setting up testing environments.

References

[1] FUSE: Filesystem in userspace. http://fuse.sourceforge.net.

[2] Average number of uploaded and linked photos of Facebook users as of January 2011, by gender. Statista, 2013.

[3] M. S. Ackerman. The intellectual challenge of CSCW: The gap between social requirements and technical feasibility. *Human-Computer Interaction*, 15(2):179–203, 2000.

[4] S. Ahern, D. Eckles, N. S. Good, S. King, M. Naaman, and R. Nair. Over-exposed? Privacy patterns and considerations in online and mobile photo sharing. In *Proc. ACM CHI*, 2007.

[5] A. W. Appel and E. W. Felten. Proof-carrying authentication. In *Proc. ACM CCS*, 1999.

[6] Apple. Apple iCloud. https://www.icloud.com/, 2013.

[7] C.-M. Au Yeung, L. Kagal, N. Gibbins, and N. Shadbolt. Providing access control to online photo albums based on tags and linked data. In *Proc. AAAI-SSS:Social Semantic Web*, 2009.

[8] O. Ayalon and E. Toch. Retrospective privacy: Managing longitudinal privacy in online social networks. In *Proc. SOUPS*, 2013.

[9] L. Bauer, S. Garriss, and M. K. Reiter. Distributed proving in access-control systems. In *Proc. IEEE SP*, 2005.

[10] L. Bauer, S. Garriss, and M. K. Reiter. Efficient proving for practical distributed access-control systems. In *ESORICS*, 2007.

[11] L. Bauer, M. A. Schneider, and E. W. Felten. A general and flexible access-control system for the Web. In *Proc. USENIX Security*, 2002.

[12] A. Besmer and H. Richter Lipford. Moving beyond untagging: Photo privacy in a tagged world. In *Proc. ACM CHI*, 2010.

[13] A. J. Brush and K. Inkpen. Yours, mine and ours? Sharing and use of technology in domestic environments. In *Proc. UbiComp*. 2007.

[14] Facebook & your privacy: Who sees the data you share on the biggest social network? Consumer Reports Magazine, June 2012.

[15] D. Coursey. Google apologizes for Buzz privacy issues. *PCWorld*. Feb. 15, 2010.

[16] J. L. De Coi, E. Ioannou, A. Koesling, W. Nejdl, and D. Olmedilla. Access control for sharing semantic data across desktops. In *Proc. ISWC*, 2007.

[17] E. De Cristofaro, C. Soriente, G. Tsudik, and A. Williams. Hummingbird: Privacy at the time of Twitter. In *Proc. IEEE SP*, 2012.

[18] K. W. Edwards, M. W. Newman, and E. S. Poole. The infrastructure problem in HCI. In *Proc. ACM CHI*, 2010.

[19] C. Elliott and F. Pfenning. A semi-functional implementation of a higher-order logic programming language. In P. Lee, editor, *Topics in Advanced Language Implementation*. MIT Press, 1991.

[20] D. Garg and F. Pfenning. A proof-carrying file system. In *Proc. IEEE SP*, 2010.

[21] R. Geambasu, M. Balazinska, S. D. Gribble, and H. M. Levy. Homeviews: Peer-to-peer middleware for personal data sharing applications. In *Proc. ACM SIGMOD*, 2007.

[22] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O'Toole. Semantic file systems. In *Proc. ACM SOSP*, 1991.

[23] M. Hart, C. Castille, R. Johnson, and A. Stent. Usable privacy controls for blogs. In *Proc. IEEE CSE*, 2009.

[24] K. Hill. Teacher accidentally puts racy photo on students' iPad. School bizarrely suspends students. *Forbes*, October 2012.

[25] M. Johnson, S. Egelman, and S. M. Bellovin. Facebook and privacy: It's complicated. In *Proc. SOUPS*, 2012.

[26] M. Johnson, J. Karat, C.-M. Karat, and K. Grueneberg. Usable policy template authoring for iterative policy refinement. In *Proc. IEEE POLICY*, 2010.

[27] A. K. Karlson, A. J. B. Brush, and S. Schechter. Can I borrow your phone? Understanding concerns when sharing mobile phones. In *Proc. ACM CHI*, 2009.

[28] P. Klemperer, Y. Liang, M. L. Mazurek, M. Sleeper, B. Ur, L. Bauer, L. F. Cranor, N. Gupta, and M. K. Reiter. Tag, you can see it! Using tags for access control in photo sharing. In *Proc. ACM CHI*, 2012.

[29] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Trans. Comput. Syst.*, 10(4):265–310, 1992.

[30] C. Lesniewski-Laas, B. Ford, J. Strauss, R. Morris, and M. F. Kaashoek. Alpaca: Extensible authorization for distributed services. In *Proc. ACM CCS*, 2007.

[31] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *Proc. IEEE SP*, 2002.

[32] L. Little, E. Sillence, and P. Briggs. Ubiquitous systems and the family: Thoughts about the networked home. In *Proc. SOUPS*, 2009.

[33] A. Masoumzadeh and J. Joshi. Privacy settings in social networking systems: What you cannot control. In *Proc. ACM ASIACCS*, 2013.

[34] M. L. Mazurek, J. P. Arsenault, J. Bresee, N. Gupta, I. Ion, C. Johns, D. Lee, Y. Liang, J. Olsen, B. Salmon, R. Shay, K. Vaniea, L. Bauer, L. F. Cranor, G. R. Ganger, and M. K. Reiter. Access control for home data sharing: Attitudes, needs and practices. In *Proc. ACM CHI*, 2010.

[35] M. L. Mazurek, P. F. Klemperer, R. Shay, H. Takabi, L. Bauer, and L. F. Cranor. Exploring reactive access control. In *Proc. ACM CHI*, 2011.

[36] D. D. McCracken and R. J. Wolfe. *User-centered website development: A human-computer interaction approach*. Prentice Hall Englewood Cliffs, 2004.

[37] Microsoft. Windows SkyDrive. http://windows.microsoft.com/en-us/skydrive/, 2013.

[38] R. Needleman. How to fix Facebook's new privacy settings. *cnet*, December 2009.

[39] J. Nielsen and J. T. Hackos. *Usability engineering*, volume 125184069. Academic press Boston, 1993.

[40] J. S. Olson, J. Grudin, and E. Horvitz. A study of preferences for sharing and privacy. In *Proc. CHI EA*, 2005.

[41] D. Peek and J. Flinn. EnsemBlue: Integrating distributed storage and consumer electronics. In *Proc. OSDI*, 2006.

[42] A. Post, P. Kuznetsov, and P. Druschel. PodBase: Transparent storage management for personal devices. In *Proc. IPTPS*, 2008.

[43] V. Ramasubramanian, T. L. Rodeheffer, D. B. Terry, M. Walraed-Sullivan, T. Wobber, C. C. Marshall, and A. Vahdat. Cimbiosys: A platform for content-based partial replication. In *Proc. NSDI*, 2009.

[44] M. N. Razavi and L. Iverson. A grounded theory of information sharing behavior in a personal learning space. In *Proc. ACM CSCW*, 2006.

[45] R. W. Reeder, L. Bauer, L. Cranor, M. K. Reiter, K. Bacon, K. How, and H. Strong. Expandable grids for visualizing and authoring computer security policies. In *Proc. ACM CHI*, 2008.

[46] O. Riva, Q. Yin, D. Juric, E. Ucan, and T. Roscoe. Policy expressivity in the Anzere personal cloud. In *Proc. ACM SOCC*, 2011.

[47] B. Salmon, S. W. Schlosser, L. F. Cranor, and G. R. Ganger. Perspective: Semantic data management for the home. In *Proc. USENIX FAST*, 2009.

[48] S. Schroeder. Facebook privacy: 10 settings every user needs to know. *Mashable*, February 2011.

[49] M. Seltzer and N. Murphy. Hierarchical file systems are dead. In *Proc. USENIX HotOS*, 2009.

[50] D. K. Smetters and N. Good. How users use access control. In *Proc. SOUPS*, 2009.

[51] J. Staddon, P. Golle, M. Gagné, and P. Rasmussen. A content-driven access control system. In *Proc. IDTrust*, 2008.

[52] J. Strauss, J. M. Paluska, C. Lesniewski-Laas, B. Ford, R. Morris, and F. Kaashoek. Eyo: device-transparent personal storage. In *Proc. USENIX-ATC*, 2011.

[53] F. Stutzman, R. Gross, and A. Acquisti. Silent listeners: The evolution of privacy and disclosure on facebook. *Journal of Privacy and Confidentiality*, 4(2):2, 2013.

[54] K. Vaniea, L. Bauer, L. F. Cranor, and M. K. Reiter. Out of sight, out of mind: Effects of displaying access-control information near the item it controls. In *Proc. IEEE PST*, 2012.

[55] J. A. Vaughan, L. Jia, K. Mazurak, and S. Zdancewic. Evidence-based audit. *Proc. CSF*, 2008.

[56] B. Weitzenkorn. McAfee's rookie mistake gives away his location. *Scientific American*, December 2012.

[57] S. Whittaker, O. Bergman, and P. Clough. Easy on that trigger dad: a study of long term family photo retrieval. *Personal and Ubiquitous Computing*, 14(1):31–43, 2010.

[58] P. J. Wisniewski, H. Richter Lipford, and D. C. Wilson. Fighting for my space: Coping mechanisms for SNS boundary regulation. In *Proc. ACM CHI*, 2012.

[59] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos operating system. In *Proc. ACM SOSP*, 1993.

[60] T. Wobber, T. L. Rodeheffer, and D. B. Terry. Policy-based access control for weakly consistent replication. In *Proc. Eurosys*, 2010.

[61] S. Yardi and A. Bruckman. Income, race, and class: Exploring socioeconomic differences in family technology use. In *Proc. ACM CHI*, 2012.

[62] G. Zimmermann and G. Vanderheiden. Accessible design and testing in the application development process: Considerations for an integrated approach. *Universal Access in the Information Society*, 7(1-2):117–128, 2008.