Toward strong, usable access control for shared distributed data

Michelle L. Mazurek, Yuan Liang, Manya Sleeper, Lujo Bauer,
Gregory R. Ganger, Nitin Gupta, and Michael K. Reiter

CMU-PDL-13-110
June 2013

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

As non-expert users produce increasing amounts of personal digital data, providing them with usable access control becomes critical. Current approaches are frequently unsuccessful, either because they insufficiently protect data or confuse users about policy specification. We present a distributed file-system access-control infrastructure designed to match users' mental models while providing principled security. Our design combines semantic, tag-based policy specification with logic-based access control, enabling flexible support for intuitive policies while providing high assurance of correctness. We support private and conflicting tags, decentralized policy enforcement, and unforgeable audit records. Our logic can express a variety of policies that map well to real users' needs. To evaluate our design, we develop a set of detailed, realistic case studies drawn from prior research into users' access-control needs. The case studies can also be applied to other systems in the personal access-control space. Using simulated traces generated from the case studies, we demonstrate that our prototype implementation can enforce users' policies with acceptable overhead.

Acknowledgements: This research was supported by the National Science Foundation via grants 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 a Facebook graduate fellowship. We thank the members and companies of the PDL Consortium (including Actifio, APC, EMC, Emulex, Facebook, Fusion-io, Google, Hewlett-Packard Labs, Hitachi, Intel, Microsoft Research, NEC Laboratories, NetApp, Oracle, Panasas, Riverbed, Samsung, Seagate, STEC, Symantec, VMware, and Western Digital) for their interest, insights, feedback, and support. We thank Lorrie Cranor, Raja Sambasivan, and Blase Ur for suggestions and feedback.

Keywords: Access control, file systems, security, usability, tagging, evaluation

1 Introduction

Non-expert computer users produce increasing amounts of personal digital data, distributed across their devices (laptops, tablets, phones, etc.) and in the cloud (Gmail, Facebook, Flickr, etc.). These users are interested in accessing this content seamlessly from any device, as well as sharing it with friends, relatives and co-workers. Thus, systems and services designed to meet these needs are proliferating [40, 46, 37, 38, 33, 7].

In this environment, access control is critical. Users who are confident that their data is protected will take full advantage of available services, sharing varied content with a range of people. Without this confidence, however, some users who worry about identity theft, trouble at work, embarrassment and other problems related to unintended disclosure may limit the content they are willing to share. Users who do not limit their sharing may experience a breach of data they had believed to be protected – due to either misconfiguration or a security problem in the access-control model – and may be hurt or angry and sever ties with the offending service entirely. At the same time, access-control configuration is a secondary task most users do not want to spend much time on.

Commercial systems generally provide some access control, but their approaches often fail, either by insufficiently guarding access or by confusing users about policy creation. This was demonstrated by the now defunct Google Buzz product, a social network that created privacy issues by automatically drawing followers from a user's Google mail contacts and then making them public [15]. A 2012 survey also found that approximately 13 of the 150 million US Facebook users did not know how to adjust or did not use their privacy settings [2], showing that users do not sufficiently understand how to create policies. This was further supported by the 37% of participants in a recent lab study who were concerned about friends seeing posted content, a concern that was not fully addressed by Facebook's privacy settings [23].

These problems generally have two sources: ad-hoc security mechanisms that lead to unforeseen behavior, and policy authoring that does not match users' mental models. These problems cannot be fixed by only improving the user interface; the underlying access-control infrastructure must provide principled security while aligning with users' understanding [25]. In this paper, we present a distributed access-control infrastructure designed to support users' policy needs while providing principled security. We provide flexible policy specification meant to support real access-control policies, which are complex, frequently include exceptions, and change over time [31, 39, 32]. Our system is designed to meet important requirements identified by prior work.

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 [41, 42, 27]. In our system, both content and policy are organized using tags, rather than hierarchically.

Second, because tags are central to how users organize their data, they must be treated accordingly. Our design takes into consideration that tags can be private, as well as that different users may disagree on which attributes best describe a piece of content. In addition, tags must be authoritative to specify policy: resistant to unauthorized changes and forgery.

Third, our system is designed to work within a distributed, decentralized, multi-user environment, in which users access files from various devices without need for a dedicated central server, an increasingly important environment [41]. We also support multi-user devices; although these devices are becoming less common [14], they remain important cases, particularly in the home [31, 24, 52]. Cloud environments are also inherently multi-user.

Fourth, we consider an environment where files belonging to different people can be freely shared; unlike, for example, a fixed military policy hierarchy. Further, we enhance support for complex and dynamic policies by providing a meaningful audit trail that allows policy misconfiguration detection and correction.

We make three main contributions:

1. We describe a file-system access-control architecture that combines semantic policy specification with

logic-based credentials, providing an intuitive, flexible policy model without sacrificing correctness. Our design supports distributed file access, private and conflicting metadata, decentralized policy enforcement, and unforgeable audit records that describe who accessed what content and why that access was allowed. Our logic can express a variety of flexible policies that map well to real users' needs.

2. We develop a set of realistic access-control case studies, drawn from user studies of non-experts' policy needs and preferences. To our knowledge, our case studies, which are also applicable to other systems in the space of personal content sharing, 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 that our infrastructure is capable of expressing these policies.

3. Using our case studies and a prototype implementation, we demonstrate that semantic, logic-based policies can be enforced reasonably efficiently. In particular, the overheads for the logic-based access control are small enough for the interactive use representative of our target environment.

2 Background and related work

We discuss related work in four areas: users' access-control needs and preferences; systems that use tags for access control; systems that use tags to manage placement and replication of personal data across devices; and logic-based access control. We then provide a brief introduction to logic-based access control.

Access-control policies and preferences Users' access-control preferences for personal data are nuanced, complex, dynamic and context-dependent. [4, 39, 32]. Some sharing preferences can be captured broadly but others require more fine-grained rules, and exceptions are frequent and important [35, 31]. Users want to protect personal content from strangers, and are also concerned about managing identity among family, friends, and acquaintances [12, 5, 30, 23]. Furthermore, when formal tools fail, users fall back on ad-hoc coping mechanisms [49]. Our access-control infrastructure is designed to support personal polices that are complex, dynamic, and cover a range of sharing preferences.

Capturing such complex preferences is challenging [4]. Sharing decisions may also be impacted by the difficulty of setting and updating policies, so users must be able to easily set up access-control policies [43, 47, 45]. We believe our infrastructure helps users express their desired policies more naturally.

Tags for access control Our system relies on tags for defining access-control policies. Researchers have prototyped tag-based access-control systems for specific content domains, including web photo albums [8], RDF tags [16], microblogging services [17], and encrypting sensitive portions of legal and corporate documents [44].

Prior work has also indicated that tag-based access control makes sense to non-expert users. Hart et al. designed a tag-based access-control system for blogs. They found that role-playing participants could construct tag-based policies and that policies could be represented using existing tags [22]. In a lab study using users' own photos, tags, and policy preferences, Klemperer et al. also found accurate policies could be created[27].

Tags for personal distributed file systems Many distributed file systems use tags for file management, an idea that was first introduced by Gifford et al. [21], and that Seltzer suggests will completely eclipse hierarchical management [42].

Several systems allow tag-based file management, but do not explicitly provide access-control capabilities (e.g., [46]). Ensemblue uses a central-server model and a limited form of attribute-based file management with partial replication [36]. Anzere and Perspective both provide peer-to-peer sharing with replication explicitly governed by tag-based policies [40].

Other attribute-based systems provide access-control. Homeviews offers applications a database view of remote files, based on name-value metadata pairs, with capability-based access control. Remote files, however, are read-only within the system, and each capability governs only files local to one device [20]. Our approach provides more principled policy enforcement, and we allow policy to be specified generally for files across several devices.

Cimbiosys offers partial replication based on tag filtering, using a tree topology that ensures eventual filter consistency [38]. Replication is governed by access control that prevents items from being replicated to unauthorized devices and discards content or tag updates from unauthorized sources. This approach uses fixed hierarchical policies, in which permissions granted to a set of items automatically apply to subsets [51]. Research indicates that real personal policies don't follow this fixed hierarchical model; our more flexible system builds policies around non-hierarchical, editable tags, and does not require a centralized trusted authority.

Logic-based access control Our work builds a tag-based, logic-based access-control system intended for a distributed file system. Prior work has explored theoretical and practical implementations of logic-based access-control systems and frameworks. One early example is Wobber et al.'s Taos, in which authentication requests correspond to proofs, and which allows compound principals that express complex relationships and delegations of authority [50].

Appel and Felten developed proof-carrying authentication (PCA), a framework for expressing application-specific access-control logics using a more general higher-order logic [6]. PCA has since been applied to control access to web sites, doors within an office building, and public key infrastructures (PKI) [10, 9, 28].

Other work has also applied logic to access control. KeyNote is a trust-management system for PKI designed for simplicity and expressivity [13]. RT combines trust management and role-based access-control for policy delegation in distributed systems [29]. SecPal is an access-control language focused on usability that allows flexible authority delegation and constraint specification [11].

Perhaps most similar to our work is PCFS, in which Garg and Fenning apply PCA to a local file system [19]. In PCFS, proofs are constructed and verified prior to requesting access, then cached as capabilities that can be verified much faster. PCFS is evaluated using a realistic case study based on government policy for classified and sensitive data. In that environment, policies are fixed and hierarchical. Our approach is designed to support a much broader, more flexible set of policies.

More broadly, Abadi et al. provide a formal structure for a logic-based distributed access-control architecture [3]. Keromytis and Smith also provide a set of requirements against which to evaluate distributed access-control systems: allow for a range of policy-management front ends, tailored to individual circumstances; support policy requirements for a variety of applications; use both policy-management and enforcement mechanisms that can scale with increasing numbers of users and resources; allow easy addition and revocation of users and privileges; and impose low overhead on network services themselves [26]. The authors suggest an architecture with decentralized policy management and enforcement, flexible credentials, and multi-layered delegation of authority, combined with a well-designed revocation mechanism. Our approach incorporates most of these features.

2.1 Background: Logic-based access control

In our infrastructure, as in other logic-based access-control systems, a resource cannot be accessed until a proof that the access is allowed is generated. Suppose that to read the file "report," which is owned by Bob, Alice must prove that Bob supports that access. She can do this by assembling a set of *credentials*, or cryptographically signed statements made by principals. These verifiable and unforgeable statements are specified as formulas in an access-control logic, and the proof is a derivation demonstrating that the credentials are sufficient to allow access. Our system uses an intuitionistic first-order logic with predicates and quantification over base types.

Credentials can include delegations of authority. The following example indicates that principal *A* grants authority over some action *F* – for example, reading a file or deleting a piece of metadata – to principal *B*:

$$A \text{ signed delegate } (B,\ F)$$

Other credentials are uses of authority; the following example indicates that principal *A* wants to perform action *F*.

$$A \text{ signed } F$$

Statements made with signed strictly reflect assertions made directly by a principal, while says statements describe any belief of a principal that can be derived in the logic from some set of signed statements. Any signed statement can be used to derive a corresponding says statement, as follows:

$$\frac{A \text{ signed } F}{A \text{ says } F}$$

For simplicity in examples, we elide this step in further proofs throughout the paper.

Further inferences can be made using modus ponens:

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

To extend our Alice and Bob example from above, Alice can combine a delegation from Bob and her own use of authority to make a proof that Bob supports her access, as follows:

$$\frac{\text{Bob signed deleg(Alice, readfile(report))} \quad \text{Alice signed readfile(report)}}{\text{Bob says readfile(report)}}$$

3 System overview

This section provides an overview of our architecture and discusses design choices related to managing tags, organizing permissions, and establishing authority.

3.1 High-level architecture

Our design encompasses an ensemble of devices, each storing files and tags. Users on one device can remotely access files and tags on other devices. File management is governed using semantic (i.e., tag-based) object naming and search, rather than a hierarchical directory structure. Users can query local and remote files using tags, such as searching for files with “type=movie” or “keyword=budget.” We extend this semantic management to include access-control policy specification. For example, 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 we could think of each cloud user as being assigned her own logical “device.”

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, formatted as a logical statement that access is authorized. To gain access, the requesting user must provide a logical proof of the challenge statement. Each user runs a software *user agent*, which stores all the authorization credentials the user has received, and can use them to generate the requested proof. The local monitor then verifies the proof.

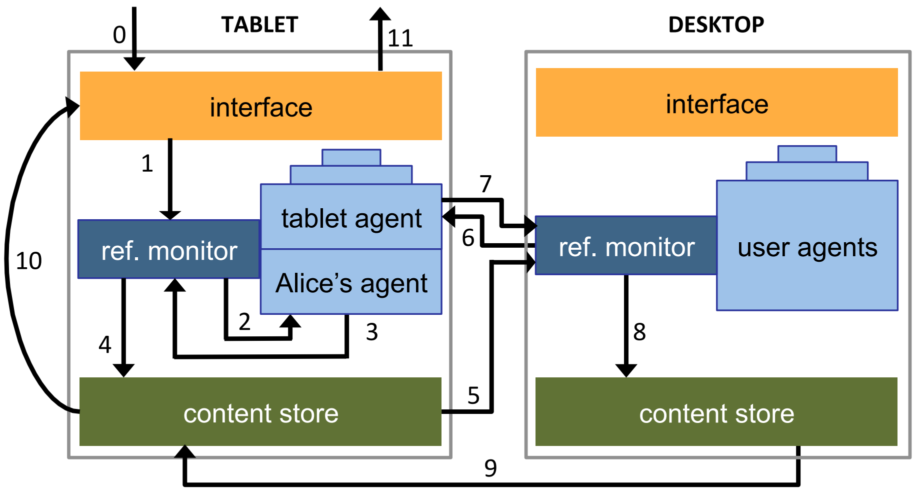

Figure 1. Access-control architecture. Using her tablet, Alice requests to open a file stored on the desktop (0). The interface component forwards this request to the reference monitor to be validated (1). The local monitor produces a challenge, which is proved by Alice's local agent (2-3), then asks the content store for the file (4). The content store requests the file from the desktop (5), triggering a challenge from the desktop's reference monitor (6). Once the tablet's agent proves that the tablet is authorized to receive the file (7), the desktop monitor instructs the desktop content store to send it to the tablet (8). The tablet's content store returns the file to Alice via the interface component (10-11).

The challenges generated by the reference monitors have seven types, which fall into three categories: permission to read, write, or delete an existing file; permission to read or delete an existing tag; and permission to create content (files or tag) on the target device. The rationale for this is explained in Section 3.3. Each challenge includes a nonce to prevent replay attacks; for simplicity, we omit the nonces in the paper. The logic is not exposed directly to users, but abstracted by an interface that is beyond the scope of this paper.

Local and remote requests For all requests, whether local or remote, the user must prove to the local device that she is authorized to access the file. If the desired content is remote, a second proof is also required: the local device must prove to the remote device that the local device is trusted to store the requested content and enforce policy about it. This ensures that while untrusted devices may allow local users to misbehave, they cannot circumvent policy for remote data. Figure 1 illustrates a remote access.

3.2 Designing strong metadata

Semantic management of both files and access-control policy lends new importance to tag handling. Because we base policy on tags, tags must not be altered without authorization or forged. Suppose, as in the earlier example, Alice gives Malcolm access to photos from her Hawaiian vacation; Malcolm could gain unauthorized access to her budget plan if he can change its type from "spreadsheet" to "photo" and add the tag "album=Hawaii." We also want to allow users to keep their tags private, as well as to allow users to disagree on the tags for a shared file. Thus, we make two key design decisions.

First, to support private tags, we treat each tag as a first-class object, independent of the file it describes. As with files, reading a tag requires a proof of access. This means that assembling a file access proof that depends on tags will often require first assembling a proof of access to those tags. Figure 2 shows an example of a two-stage proof, in which Bob's agent must first prove access to a tag and then use that tag to prove access to a file.

Second, to maintain tag integrity and allow conflicting tags, we represent tags as cryptographically

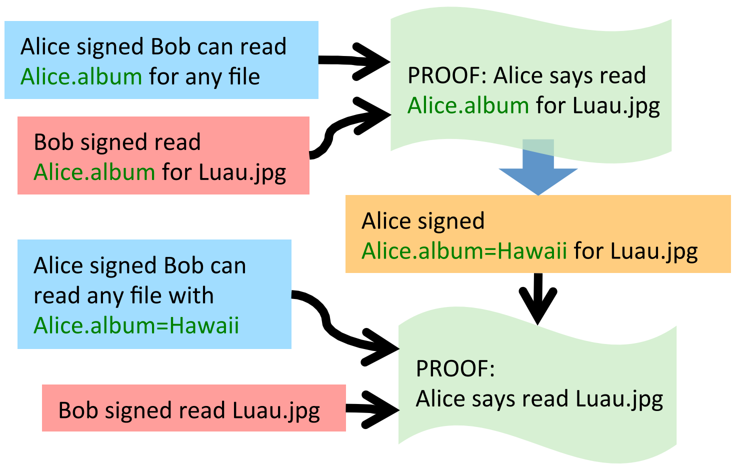
Alice signed Bob can read Alice.album for any file
Bob signed read Alice.album for Luau.jpg
PROOF: Alice says read Alice.album for Luau.jpg
Alice signed Alice.album=Hawaii for Luau.jpg
Alice signed Bob can read any file with Alice.album=Hawaii
Bob signed read Luau.jpg
PROOF: Alice says read Luau.jpg

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.

signed credentials of the form *principal* signed tag(*attribute*, *value*, *file*). (For clarity in examples, we use descriptive file names; in reality, our system uses globally unique ID numbers, as described in Section 7.) For example, Alice can assign the song "Tangled Up in Blue" a four-star rating by signing a credential:

Alice signed tag(rating, 4, "Tangled Up in Blue")

Using this representation, Alice, Bob and Caren can all assign different ratings to "Tangled Up in Blue." 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 tags about any file. Principals can be restricted from storing tags on devices they do not own, but if Alice is allowed to create or store tags on her laptop then those tags may reference any file.

Some tags are naturally written as attribute-value pairs (e.g., "type=music", "rating=PG"). Others are commonly value-only (e.g., photos tagged with "vacation" or with the name of people in the photo). To maximize flexibility, we handle all tags as name-value pairs; value-only tags can be rewritten by the infrastructure as name-value pairs, e.g., by transforming "vacation" into "vacation=true."

3.3 Defining access permissions

Our tag design influences the way access permissions are organized. As described in Section 3.1, the first class of permissions allows reading, writing, and deleting individual files in the expected manner. Permissions to read and delete tags are similarly straightforward. Tags are never updated; instead, the signed credential is revoked and a new signed credential is issued. As a result, there is no explicit per-item write tag permission.

Permission to create files and tags is organized per device rather than per item. Because files are organized by their attributes rather than in a directory structure, creating one file on a target device is equivalent to creating any other. Similarly, a user with permission to create tags can always create any tag in her own namespace, and no tags in any other namespace, so only permission to create any tags on the target device is required.

3.4 Devices, principals, and authority

We treat both users and devices as principals who can make statements. Each principal has an identifying public-private key pair; users' keys remain consistent across devices. This approach accounts for multi-user devices and decisions based on the combined trustworthiness of a user and a device. (The problem of distributing the keys securely is outside the scope of this paper.)

Devices ultimately control who accesses the information they store, so devices are the final arbiters of access in the logic. The challenge statements issued by any reference monitor are of the form device $\textsf{says}$ action, where action reflects the type of permission the requester is seeking. For example, to read a file on her laptop, Alice's software agent must prove that

$$\mathit{AliceLaptop}\ \textsf{says readfile}(f)$$

This design captures the intuition that sensitive content should not be transferred to untrusted devices, while trusted devices are responsible for enforcing access-control policy. For most single-user devices, a straightforward default policy in which the device delegates all of its authority to its owner is appropriate. From this delegation, the device owner can make additional policy. For shared devices or other less common situations, a more complex device policy may be necessary.

4 Expressing semantic policies

Our system uses logic-based access control to support semantic policy management. We describe how users' semantic policies could be expressed and enforced in our system.

4.1 Semantic policy for files

The simplest example of a semantic policy is allowing access to a file that has an attribute or attributes. File accesses incur challenge statements of the form device $\textsf{says}$ $\mathit{action}(f)$, where f is a file and action can be one of $\textsf{readfile}$, $\textsf{writefile}$, or $\textsf{deletefile}$.

As a first example, let's assume Alice wants to allow Bob to listen to any of her music. This policy is implemented as a delegation credential limited by an implication: if Alice says that a given file has type=music, then Alice delegates to Bob permission to read that file. This type of delegation also includes quantification over files. We write this policy as follows:

$$\textsf{Alice signed } \forall f : \textsf{tag(“type”,“music”,}f\textsf{)} \rightarrow \textsf{deleg(Bob,readfile(}f\textsf{))} \tag{1}$$

To use this delegation to listen to "Stairway to Heaven," Bob's agent must combine it with a proof that Alice says "Stairway to Heaven" has type=music, along with a statement of Bob's intent to open "Stairway to Heaven" for reading. The resulting proof is as follows:

$$\textsf{Alice signed tag(“type”,“music”,“Stairway to Heaven”)} \tag{2}$$

$$\textsf{Bob signed readfile(“Stairway to Heaven”)} \tag{3}$$

$$\frac{\dfrac{(1) \qquad (2)}{\textsf{Alice says deleg(Bob, readfile(“Stairway to Heaven”))}} \qquad (3)}{\textsf{Alice says readfile(“Stairway to Heaven”)}}$$

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

This idea is easily extended to multiple attributes as well as to groups of people. For example, Alice might allow anyone in the group she calls co-workers to view her vacation photos. To do this, Alice simply assigns users to groups (each such assignment is a signed credential), then delegates authority to the group rather than to an individual. Only users with credentials signed by Alice establishing their membership in the group can use the delegation.

4.2 Policy about tags

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

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

We handle these query types using an *attribute list* primitive. An attribute list is a set of (principal, attribute, value) triples representing the tags for which access is requested. Each tag access request (to read or delete) specifies an attribute list and a target file.

Because tag queries can apply to multiple values of one attribute or multiple files, we use the wildcard notation * to indicate a request for all possible completions. The listing query example above, which is a search on multiple files, would be implemented with the attribute list [(Alice, type, movie), (Alice, genre, comedy)] and the target file *. The attribute query example, by contrast, identifies a specific target file but not a specific value, and would therefore be written with an attribute list like [(Alice, album, *)] and a target file "Luau.jpg." A status query for the same file might be written with an attribute list [(AliceLaptop, *, *)]. Within the logic, the wildcard is a normal string constant with no special properties.

Tag challenges have the form $\mathit{device}\ \mathsf{says}\ \mathit{action}(\mathit{attrlist},\ \mathit{file})$, where action can be either $\mathsf{readtags}$ or $\mathsf{deletetags}$.

Here is a proof sketch for the listing query example:

$$\mathsf{Alice\ signed}\ \forall f : \mathsf{deleg(Bob,readtags([(Alice,type,movie),(Alice,genre,comedy)],}\ f\mathsf{))} \tag{4}$$

$$\mathsf{Bob\ signed\ readtags([(Alice,type,movie),(Alice,genre,comedy)],*)} \tag{5}$$

$$\frac{(4) \qquad\qquad (5)}{\mathsf{Alice\ says\ readtags([(Alice,type,movie),(Alice,genre,comedy)],*)}}$$

Implications of tag policy Interactions between tag delegation credentials raise issues. Tag 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 permission to list all photos and Hawaii files could manually compute the intersection of the results, so a request for Hawaii photos should be provable. Our system supports this approach.

Another interesting issue related to tag policies is the question of limiting scope. Returning to the attribute query example, Alice can allow Bob to read the album name for any file, or only for files in the

Hawaii album. If she chooses the latter, and Bob requests to read the album name regardless of its value (attribute list [(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 succeed, and Bob can learn for any photo only whether or not it's in the Hawaii album. If it's not in the Hawaii album, no information about what other album it might be in (if any) is leaked.

A related point is how to handle unprovable requests. Users will likely sometimes query for a broader set of tags than they have access to; Bob may ask to list all of Alice's photos when he only has permission for Hawaii photos. Bob's agent will then be asked to make an impossible proof. The most straightforward option would be for the query to fail and provide no results. A better outcome would be for Bob to receive an abridged list of files containing only Hawaii photos. In practice, one solution might be for Bob's agent to limit his initial request to something the agent can prove, based on available credentials – in this case, rewriting Bob's request from all photos to Hawaii photos. We leave this as a future optimization.

4.3 Negative policy

Negative policies, based on forbidding access rather than allowing it, are important but often challenging for access-control systems. If explicitly negative policies are allowed, the system must resolve conflicting deny and allow policies; 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" [27] and sharing all files other than financial documents [31]. Explicit deny policies do not make sense in our context; all access is denied unless a proof supporting it is presented. We therefore must find an alternative way to express these intuitive policies.

One naive solution might be to allow Alice to grant Bob permission to view her photos when the tag "weird=true" is not present. However, this is unworkable; principals may not know about all tags that exist, because they are private or because they have not yet been distributed. Writing credentials of this form could also allow Bob to make unauthorized accesses by interrupting the transmission of tag credentials.

Our solution has two parts. First, we allow delegation based on alternate tags: 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 a defined set of non-overlapping categories – e.g., restricting children to movies not rated R. If, however, a file is tagged with both "topic=financial" and "topic=vacation," then the policy given above would still allow Bob to access that file.

To handle situations with overlapping and less-well-defined categories, e.g., denying access to weird photos, Alice can grant Bob permission to view files with "type=photo" and "weird=false." For 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 tracks them. As we already assume the interface tracks previously made tags to help the user maintain consistent labels and avoid typos, this is not an onerous requirement. Second, granting someone the ability to view files with "weird=false" suggests that some photos with "weird=true" exist; this leaks potentially private information. We assume the same policymaking interface can obfuscate this kind of negative tag (e.g., by using a hash value), while maintaining a translation to the user's original meaning 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 our design, 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 very short credential lifetimes. This ensures that credentials relating to old policies will not remain valid for long after changes. The user's agent can be set to automatically renew each short-lived credential until the user manually revokes it. Alternatively, we can require all credentials used in a proof to be countersigned, confirming validity. Both of these options can be expressed in our logic; for simplicity, we do not reference them further.

5 Realistic policy examples

We previously discussed abstractly how policy needs can be translated into logic-based credentials. We must next ensure that we can represent real users' policies within our infrastructure.

Finding realistic scenarios for personal data is difficult; users often don't know or cannot articulate their policies. We know of no standard policy or file-sharing benchmarks for this scenario. Prior research has often relied on policy and file-usage scenarios that are based on the researcher's own files, those of her officemates, or intuition [51, 40, 36, 46]. PCFS presents a well-grounded case study of access control for the military/intelligence environment [19]; however, this scenario is inappropriate for a personal policy space defined by fine-grained rules with many exceptions [31]. Our usage scenario also assumes distributed sharing and device integration that's not yet realistic for most users. Commercial systems that bridge this divide, e.g., Dropbox and Google Drive, are reluctant to release usage traces out of respect for user privacy.

We developed a set of access-control-policy case studies, grounded in detailed user study results (provided by the authors of [31] and [27]). These case studies, which could be used to evaluate other systems in this domain, are selected to provide a representative slice of diverse users' policy needs. Each case study includes a set of users and devices, as well as policy rules for at least one of the users. Each also includes a simulated trace of file and metadata actions, some of which loosely mimic real accesses and others which validate specific properties of the access-control infrastructure.

Building a simulated trace requires specifying many variables, including policy and access patterns, how many files of each type, how many and which specific tags (access-control related or otherwise) to add to each file, how many users to include in each user group, and other details. The case studies are drawn from user interviews that varied in specificity; where necessary, we make inferences to fill in gaps, relying when possible on the HCI literature and consumer market research (e.g., [48, 1]). Generally the access-control policy is well grounded, while the simulated access patterns are more speculative. We instantiate each case study in multiple ways, varying the numbers of users and files, to permit more thorough evaluation. Although these case studies are far from perfect, we believe they represent a significant improvement to commonly used evaluation strategies.

We next briefly describe each case study. Two focus on photo sharing; the third is more general.

Case study 1: Susie This case study, drawn from a study of tag-based access control for photos [27], captures a default-share mentality: Sue 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 Sue's photos, which she accesses from several personal devices but which other users access only via simulated "cloud" storage. No users besides Sue 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. Details of this case study are listed in Figure 3(a).

SUSIE

Individuals: Susie, Mom
Groups: friends, acquaintances, older friends, public
Devices: laptop, phone, tablet, cloud
Access-control categories: red flag, personal, very personal, mom-sensitive, kids
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*

(a)

JEAN

Individuals: Jean, boyfriend, sister, Pat, supervisor, Dwight
Groups: volunteers, kids, acquaintances
Devices: phone, two cloud services
Access-control categories: *goofy*, *beautiful*, *kids*, *work*, *volunteering*, names of people and events
Tags per photo: 1-10, including mixed-use access control
Policies:
People can see photos they are in.
People can see photos from events they attended.
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.
Sister can see *sister's wedding* photos.
Acquaintances can see *beautiful* photos.
No one can see *goofy* photos.

(b)

HEATHER AND MATT

Individuals: Heather, Matt, daughter
Groups: friends, other family members, co-workers, people in the house
Devices: laptop, phone, iPod, DVR, camera
Access-control categories: photos, text messages, work documents, TV shows, personal resume documents, personal financial documents, private personal documents, *not_inappropriate*
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 if Heather gives permission
Friends and other family members can see all photos and music
Heather can write all files except TV shows
Matt can write TV shows
Daughter can see all photos and *not_inappropriate* TV shows
People in the house can see all TV shows, music, photos

(c)

Figure 3. Some details of the Susie (a), Jean (b), and Heather and Matt (c) case studies.

Case study 2: Jean This case study 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. Details of this case study are listed in Figure 3(b).

Case study 3: Heather and Matt This case study (detailed in Figure 3(c)) is drawn from a broader study of users' access control needs [31]. 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 to them. Their daughter

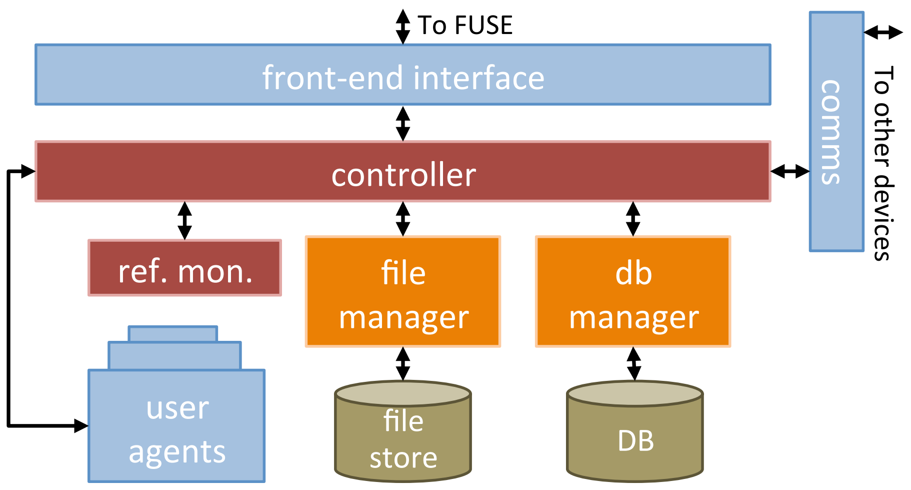

Figure 4. System architecture. The primary TCB (controller and verifier) is shown in red. The file and database manager, shown in orange, also require some trust.

has access to a limited 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.

Our ability to encode these diverse policies in our logic provides informal evidence that it is sufficiently expressive for users' needs.

6 Implementation

We built a prototype of our access-control infrastructure. Our implementation both validates the design and demonstrates that it can operate reasonably efficiently, as we show in Section 7. In this section, we discuss our prototype's architecture and our implementation of proof generation and checking.

6.1 File system implementation

Our file system 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 our file system. Instead of standard file paths, our system 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 the system architecture. System calls are received from FUSE in the front-end interface, which also parses the semantic queries. The controller is the heart of the implementation: it invokes the reference monitor to create challenges and verify proofs, the various users' agents to create proofs, and the file and database managers to access authorized content. The controller also uses the communications module to transfer challenges, proofs, and content between devices.

The implementation is about 14,000 lines of Java code, plus 1800 lines of C code used to interface with FUSE. The primary trusted computing base (TCB) includes the controller (1800 lines) and the reference

[1] http://fuse.sourceforge.net/

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 metadata: (system, *, *)
readdir	read metadata: attribute list for *
getxattr	read metadata: (principal, attribute, *)
setxattr	create metadata
removexattr	delete metadata: (principal, attribute, *)

Table 1. Proof requirements for file-related system calls

monitor (2100 lines) – the controller enforces that no content can be accessed without a proof, and the reference monitor creates challenges and verifies submitted proofs. It can be argued that the file manager (350 lines) and database manager (1500 lines) also require trust, as they must be trusted to return content only when authorized by the controller. The file manager must also be trusted to return the requested content; because metadata are implemented as signed credentials, content returned by the database cannot be forged.

The trusted computing base also includes 145 lines of LF specification defining the rules of our logic.

Mapping system calls to proof goals One important consideration was how to map high-level file operations (and the POSIX-style system calls they invoke) to the request types we defined in Section 3.3. Consider for example using the "touch" utility to create a file. At first glance, it might seem that permission to create a file on the current device is the only permission required. Consider, however, that in a hierarchical file system, users create files in directories. In our system, we offer users a similar opportunity to organize a file at create time: users can add initial metadata to any files they create. As a result, creating a file may also require proving the user is allowed to create metadata on the current device.

In practice, using touch triggers four system calls: getattr (the FUSE equivalent of stat), mknod, utime, and then another getattr. Each getattr is a query for system metadata (a status query as defined in Section 4.2) and must be protected accordingly. The mknod system call, which creates the file and its initial metadata, incurs challenges for both. Calling utime instructs the device to update its metadata 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.

In our system, 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-metadata challenge. Table 1 identifies the reference monitor challenges associated with each system call.

We also implement a small, short-term permission cache. This allows users who have recently proved access to some content to use that access again without needing to submit an additional proof. The size and expiration time of the cache can be adjusted to trade off proving overhead against faster response to policy updates.

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 [18]. The prover starts from the goal (the challenge statement provided by the verifier) and works backward, searching through its store of credentials for any that may be relevant. A relevant credential may prove the goal directly, or it may imply that if some additional goal(s) can be proven, the original goal will also be proven. The prover continues recursively solving 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 solution is found, the prover must report not only the existence of the solution but also its complete contents, so that it can be submitted to the verifier as a proof.

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. Other similar systems – for example, PCFS [19] – have a deeper but narrower structure.

We implement some basic optimization for the shallow-but-wide environment, including indexing to reduce the search space of credentials and 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 some more sophisticated approaches, which we leave to future work.

To validate proofs, we use a standard LF checker implemented in Java.

User agents build proofs using the set of credentials of which they are aware. For our simple prototype, we push all delegation credentials to each user agent. (Tag credentials are guarded by the reference monitor and not automatically shared.) This arrangement is less than ideal – it may expose some sensitive information unnecessarily, and proving can be slowed down by the presence of unneeded credentials. Agents with limited information, in contrast, could need to ask for help from other devices to complete proofs; this could reduce speed of access or even make some accesses impossible if devices holding 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 our prototype using the simulated traces we developed as part of the case studies from Section 5. Within the case studies, we varied the number of files and users to provide a wider range of tests. Table 2 lists the variations we tested.

Exp. #	Case study	Users	Files	Deleg. creds.	System calls
1	Susie	12	235	91	10 643
2	Susie	12	2 349	91	76 381
3	Jean	13	250	106	20 797
4	Jean	13	2 500	106	172 276
5	Heather/Matt	12	349	138	37 381
6	Heather/Matt	64	310	103	111 932

Table 2. Case study variations we tested.

For each case study variation, we also ran a parallel *control experiment* with access control turned off – that is, all access checks immediately succeed without any proofs being made or verified. These comparison experiments take into account the overheads associated with FUSE, Java, and our unoptimized database accesses, allowing us to focus directly on the overhead of access control.

Experimental setup For convenience while testing, we run all instances of our infrastructure independently on the same machine, which has eight 3.07GHz cores and 12GB of memory. A evaluation run consists of the following steps, which are initiated automatically. At the start of testing, accounts are created for each of the users specified by the case study; the users then create the appropriate files. Next, each user assigns a weighted-random selection of tags to her files. With the files and tags established, each user lists and opens a

	Exp. 1			**Exp. 2**			**Exp. 3**			**Exp. 4**			**Exp. 5**			**Exp. 6**		
Syscall	n	Med.	Ovhd.	n	Med.	Ovhd.	n	Med.	Ovhd.	n	Med.	Ovhd.	n	Med.	Ovhd.	n	Med.	Ovhd.
getattr	6014	5	0	40439	4	0	12032	5	1	96182	4	-1	27859	4	0	85827	4	1
getxattr	235	7	2	2349	77	58	250	6	0	2500	73	68	393	5	0	385	5	-1
mknod	235	45	1	2349	45	1	250	46	2	2500	45	1	349	45	-1	310	48	0
open	819	16	15	2913	37	32	865	2	1	3117	42	41	4166	1	0	12393	1	0
removexattr	90	18	3	894	14	1	743	15	3	7466	13	1	3	22	4	3	31	12
setxattr	2164	18	0	22143	19	-1	5508	20	0	54860	21	1	33	13	0	28	15	1
utime	235	34	6	2349	42	16	250	53	26	2500	46	18	349	29	3	310	27	0
combined	**10643**	**12**	**6**	**76381**	**15**	**2**	**20797**	**13**	**7**	**172276**	**14**	**8**	**37381**	**4**	**0**	**111931**	**4**	**0**

Table 3. Median system call times (ms) across the case study variations we tested. For each system call, we show how many times that call was made, its median duration, and the median overhead compared to a version of our file system with no access control.

weighted random selection of files from those she is authorized to access. The weightings are influenced by research describing how the age of content affects access [48]. Based on the type of file, the user reads and/or writes the file before closing it and choosing another to access. The specific access pattern is less important than broadly exercising the desired policy. Finally, each user attempts to access forbidden content to validate that the policy is set correctly.

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 [34]. User-visible operations consist of several combined system calls, so we target system call operation times well under the 100 ms limit.

Table 3 shows, for each experiment, the median time per system call. We also show the overhead of the access-control portion, calculated by subtracting the median time from the control experiment. Almost all of our results are well within the 100 ms limit. More importantly, the median proving overhead in many cases is under 20 ms. The overhead is often less than the time to make a single proof because proving time is amortized over system calls where the permission is in the cache and no proof is required. Permissions in the cache expire after 10 minutes.

The only system call that doesn't meet the 100 ms standard is readdir. In experiments with large numbers of files, readdir can often take up to several seconds, but almost all of this time comes from unoptimized database queries locating files based on their attributes. Because of the long duration and high variance of the database queries, accurately computing overhead by comparing with the control experiment was not feasible, and so we omit readdir from the table. However, the median proving time for readdir operations was 13 ms across 695 proofs from all experiments, supporting our contention that the access-control overhead is low.

In general, we have done very little optimization on our simple prototype implementation; that most of our operations already fall within the 100 ms limit is encouraging. In addition, while this performance is slower than for a typical local file system, within a distributed system longer delays may be more acceptable.

Proof generation Because proof generation is the main bottleneck inherent to our logic-based approach, it is critical to understand the factors that contribute to its performance. Table 4 summarizes the overall performance of our prover. We break proof generation down into three categories: proofs made by the primary user in the case study scenario, proofs made by device agents as part of remote operations, and proofs made by other users attempting to access content. It's most important that proofs for primary users be fast, as users do not expect delays when accessing their own content; in fact, these proofs have a median time of only 1 ms in each case study. It's also important for device proofs to be fast, as they are an extra layer of overhead within all remote operations. Proofs for devices and other users are a bit slower, but still generally under 20 ms median.

Exp.	**All proofs**		**Primary user**		**Devices**		**Other users**	
#	**n**	**Med.**	**n**	**Med.**	**n**	**Med.**	**n**	**Med.**
1	3 016	3	1 164	1	1 200	3	652	14
2	22 351	2	10 849	1	10 218	3	1 284	18
3	2 561	3	1 314	1	984	9	263	15
4	21 359	2	12 123	1	8 379	11	857	13
5	4 269	9	998	1	1 842	11	1 429	17
6	7 101	17	856	1	1 681	16	4 564	21

Table 4. Median proof times (ms) for various users, across the case study variations we tested. For each variation, we show how many proofs were made and the median proof time across the entire case study, for the primary user in the case study, for all the device agents participating in remote proofs, and for all other users.

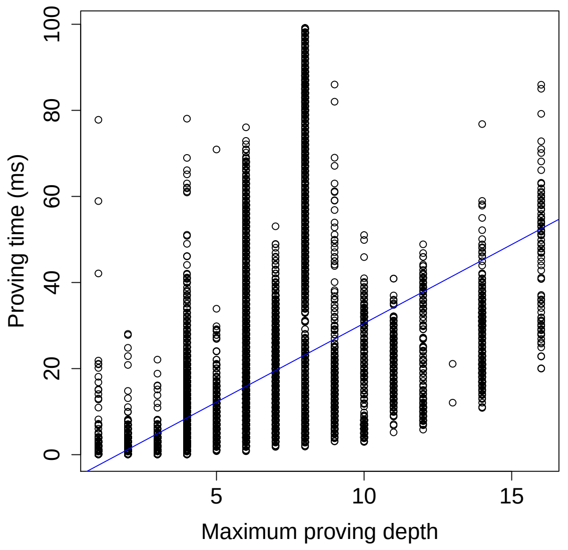

Figure 5. The effect of maximum proving depth on proving time, aggregated across all experiments. Best-fit linear model: $y \approx 3.6x - 6.1$. Only proving times of 100 ms or less are shown; this excludes less than 1% of proofs.

We next consider how factors including maximum proving depth, "red herrings" pursued by the prover, and the number of available attribute credentials affect proving time. We define proving depth as 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. We measure the maximum depth reached during proof generation, regardless of the depth of the final proof. Figure 5 compares time to find a solution (herafter, *proving time*) to maximum proving depth, aggregated across all six experiments, along with a best-fit linear model for the relationship. For each additional subgoal of depth, the best-fit line indicates proving time increases by 3.6 ms. Because the policies drawn from the user studies we considered do not include a lot of complex redelegation, proofs are generally shallow. The deeper proofs we encounter are often driven by policies concerning multiple attributes: a delegation to read files with "type=photo & beautiful=true & goofy=false" (drawn from Jean) requires solving one proof goal for each attribute. Other sources of depth include delegation between users and devices, especially in remote accesses.

We define a *red herring* as an unsuccessful proving path more than two levels deep. When the prover realizes the proving path will not succeed, it backtracks. Figure 6 graphs proving time against the number of red herrings encountered, again with a best-fit linear model. In this case, the model indicates that proving

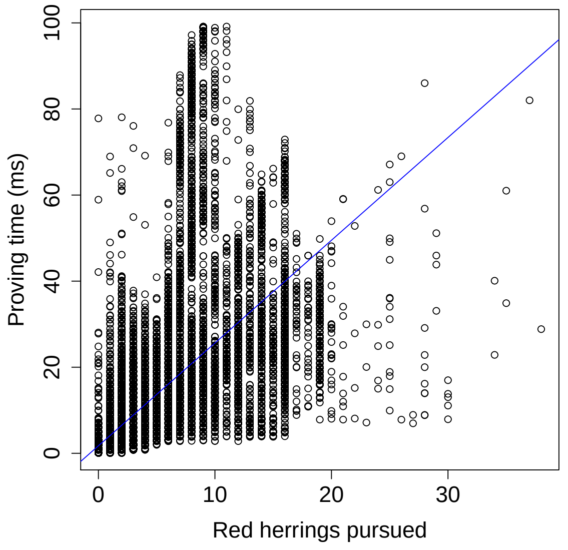

Figure 6. The effect of *red herring* searches on proving time, aggregated across all experiments. Best-fit linear model: $y \approx 2.3x + 1.7$. Only proving times of 100 ms or less are shown; this excludes less than 1% of proof times.

time increases by 2.3 ms per red herring. To reduce the potential for red herrings, we intend to add a least-recently-used cache to the proving algorithm to ensure more frequently used credentials are tried earlier.

Next, we consider how the quantity of attribute credentials affects proving time. The number of attribute credentials grows linearly with the number of files in the system, so it's important that increases in their quantity do not strongly affect proving time. Figure 7 demonstrates that they do not, with a best-fit slope less than 0.0001 ms per attribute credential. A simple index based on file ID prevents the prover from searching fruitlessly through attribute credentials that cannot be useful for the current task.

We also measured the time it takes for the prover to arrive at the conclusion that no proof can be made. Across all experiments, 1 394 instances of failed proofs had a median time of 33 ms.

Finally, we consider the long tail of proving times. Each of the graphs in this section shows only proofs taking 100 ms or less; this demonstrates the relationships between proving times and the various factors more clearly. About 0.3% of proofs fall outside this range. (Outliers were included in best-fit line calculations.) 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 less well-defined categories (such as allow "weird=false" example from Section 4.3) requires adding inverse policy tags to many files. For any policy that contains negative attributes, we need $n \times m$ extra attribute credentials, where n is the number of negative attributes in the policy and m is the number of affected files.

The size of n is determined by how many tags indicating a deny status a user employs, not how many files the status applies to. 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." The other case studies have one each: for Jean, it's photos tagged "goofy,' while for Heather and Matt it's media files tagged "inappropriate" for their young daughter. We also reviewed detailed policy data provided by the authors of [27] and found that the number of negative tags ranged from 0 to 7, with median 3 and mode 1. For most study participants, these negative tags fall into

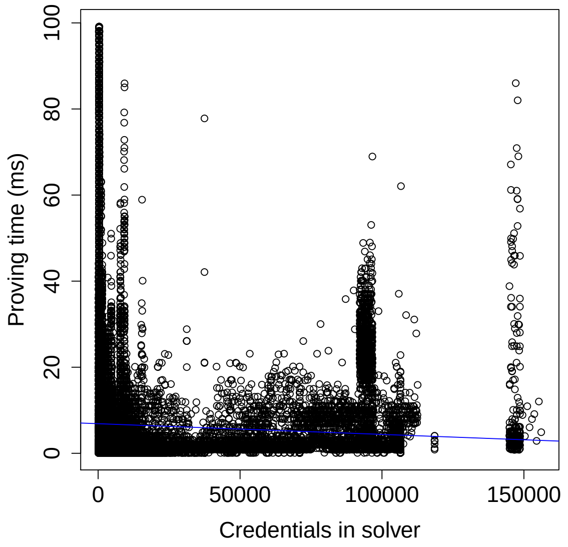

Figure 7. Adding attribute credentials does not increase proving time (aggregated across all experiments). Best-fit linear model: $y \approx 6.9$. Only proving times of 100 ms or less are shown; this excludes less than 1% of proof times.

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 used a wide range of less general negative tags.

The size 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."

Overall, the number of additional attributes $n \times m$ can grow fairly large, but as we show in Figure 7, the speed of our proving system is robust to large numbers of attribute credentials.

8 Conclusion

We present 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 demonstrate that our infrastructure can express and enforce commonly desired policies, with reasonable efficiency. Our case studies can also be applied to other systems in this space.

References

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

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

[3] Martin Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM TOPLAS*, 15(4):706–734, 1993.

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

[5] Shane Ahern, Dean Eckles, Nathaniel S. Good, Simon King, Mor Naaman, and Rahul Nair. Over-exposed?: Privacy patterns and considerations in online and mobile photo sharing. In *Proc. ACM CHI*, April 2007.

[6] Andrew W. Appel and Edward W. Felten. Proof-carrying authentication. In *Proc. ACM CCS*, November 1999.

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

[8] Ching-man Au Yeung, Lalana Kagal, Nicholas Gibbins, and Nigel Shadbolt. Providing access control to online photo albums based on tags and linked data. In *Proc. of AAAI Spring Symposium on Social Semantic Web: Where Web 2.0 Meets Web*, March 2009.

[9] Lujo Bauer, Scott Garriss, and Michael K. Reiter. Distributed proving in access-control systems. In *Proc. IEEE SP*, May 2005.

[10] Lujo Bauer, Michael A. Schneider, and Edward W. Felten. A general and flexible access-control system for the Web. In *Proc. USENIX Security*, August 2002.

[11] Moritz Y Becker, Cédric Fournet, and Andrew D Gordon. Design and semantics of a decentralized authorization language. In *Proc. CSF*, pages 3–15. IEEE, 2007.

[12] Andrew Besmer and Heather Richter Lipford. Moving beyond untagging: photo privacy in a tagged world. In *Proc. ACM CHI*, May 2010.

[13] Matt Blaze, Joan Feigenbaum, and Angelos D Keromytis. Keynote: Trust management for public-key infrastructures. In *Security Protocols*, pages 59–63. Springer, 1999.

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

[15] David Coursey. Google apologizes for Buzz privacy issues. PCWorld.

[16] Juri L. De Coi, Ekaterini Ioannou, Arne Koesling, Wolfgang Nejdl, and Daniel Olmedilla. Access control for sharing semantic data across desktops. In *Proc. ISWC*, 2007.

[17] Emiliano De Cristofaro, Claudio Soriente, Gene Tsudik, and Andrew Williams. Hummingbird: Privacy at the time of Twitter. In *SP '12: Proceedings of the 2012 IEEE Symposium on Security and Privacy*, 2012.

[18] Conal Elliott and Frank Pfenning. A Semi-Functional Implementation of a Higher-Order Logic Programming Language. In *Topics in Advanced Language Implementation*. 1991.

[19] Deepak Garg and Frank Pfenning. A proof-carrying file system. In *Proc. IEEE SP*, 2010.

[20] Roxana Geambasu, Magdalena Balazinska, Steven D. Gribble, and Henry M. Levy. Homeviews: peer-to-peer middleware for personal data sharing applications. *Proc. ACM SIGMOD*, 2007.

[21] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole. Semantic file systems. In *Proc. ACM SOSP*, October 1991.

[22] Michael Hart, Claude Castille, Rob Johnson, and Amanda Stent. Usable privacy controls for blogs. In *Proc. CSE*, 2009.

[23] Maritza Johnson, Serge Egelman, and Steven M. Bellovin. Facebook and privacy: it's complicated. In *Proc. SOUPS*, July 2012.

[24] Amy K Karlson, A J Bernheim Brush, and Stuart Schechter. Can I borrow your phone?: Understanding concerns when sharing mobile phones. In *Proc. CHI*, 2009.

[25] Edwards W Keith, Newman Mark W, and Poole Erika Shehan. The infrastructure problem in HCI. In *CHI '10: Proceedings of the 28th International Conference on Human Factors in Computing Systems*, 2010.

[26] Angelos D. Keromytis and Jonathan M. Smith. Requirements for scalable access control and security management architectures. *ACM Transactions on Internet Technology*, 7(2), May 2007.

[27] Peter Klemperer, Yuan Liang, Michelle L. Mazurek, Manya Sleeper, Blase Ur, Lujo Bauer, Lorrie Faith Cranor, Nitin Gupta, and Michael K. Reiter. Tag, you can see it!: Using tags for access control in photo sharing. In *Proc. ACM CHI*, May 2012.

[28] Chris Lesniewski-Laas, Bryan Ford, Jacob Strauss, Robert Morris, and M Frans Kaashoek. Alpaca: extensible authorization for distributed services. In *Proc. ACM CCS*, pages 432–444. ACM, 2007.

[29] Ninghui Li, John C Mitchell, and William H Winsborough. Design of a role-based trust-management framework. In *Proc. IEEE SP*, pages 114–130. IEEE, 2002.

[30] Linda Little, Elizabeth Sillence, and Pam Briggs. Ubiquitous systems and the family: thoughts about the networked home. In *Proc. SOUPS*, 2009.

[31] Michelle L. Mazurek, J. P. Arsenault, Joanna Bresee, Nitin Gupta, Iulia Ion, Christina Johns, Daniel Lee, Yuan Liang, Jenny Olsen, Brandon Salmon, Richard Shay, Kami Vaniea, Lujo Bauer, Lorrie Faith Cranor, Gregory R. Ganger, and Michael K. Reiter. Access control for home data sharing: Attitudes, needs and practices. In *Proc. ACM CHI*, April 2010.

[32] Michelle L. Mazurek, Peter F. Klemperer, Richard Shay, Hassan Takabi, Lujo Bauer, and Lorrie Faith Cranor. Exploring reactive access control. In *Proc. ACM CHI*, May 2011.

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

[34] Jakob Nielsen and JoAnn T Hackos. *Usability engineering*, volume 125184069. Academic press Boston, 1993.

[35] Judith S. Olson, Jonathan Grudin, and Eric Horvitz. A study of preferences for sharing and privacy. In *CHI EA*, April 2005.

[36] Daniel Peek and Jason Flinn. EnsemBlue: integrating distributed storage and consumer electronics. In *Proc. OSDI*, November 2006.

[37] Ansley Post, Petr Kuznetsov, and Peter Druschel. PodBase: transparent storage management for personal devices. In *Proc. IPTPS*, 2008.

[38] Venugopalan Ramasubramanian, Thomas L. Rodeheffer, Douglas B. Terry, Meg Walraed-Sullivan, Ted Wobber, Catherine C. Marshall, and Amin Vahdat. Cimbiosys: A platform for content-based partial replication. In *Proc. NSDI*, April 2009.

[39] Maryam N. Razavi and Lee Iverson. A grounded theory of information sharing behavior in a personal learning space. In *Proc. CSCW*, November 2006.

[40] Oriana Riva, Qin Yin, Dejan Juric, Ercan Ucan, and Timothy Roscoe. Policy expressivity in the Anzere personal cloud. In *Proc. ACM SOCC*, October 2011.

[41] Brandon Salmon, Steven W. Schlosser, Lorrie Faith Cranor, and Gregory R. Ganger. Perspective: semantic data management for the home. In *Proc. USENIX FAST*, 2009.

[42] Margo Seltzer and Nicholas Murphy. Hierarchical file systems are dead. In *USENIX HotOS*, 2009.

[43] Diana K. Smetters and Nathan Good. How users use access control. In *Proc. SOUPS*, July 2009.

[44] Jessica Staddon, Philippe Golle, Martin Gagné, and Paul Rasmussen. A content-driven access control system. In *Proc. IDTrust*, March 2008.

[45] Jessica Staddon, David Huffaker, Larkin Brown, and Aaron Sedley. Are privacy concerns a turn-off?: engagement and privacy in social networks. In *Proc. SOUPS*, July 2012.

[46] Jacob Strauss, Justin Mazzola Paluska, Chris Lesniewski-Laas, Bryan Ford, Robert Morris, and Frans Kaashoek. Eyo: device-transparent personal storage. In *Proc. USENIXATC*, June 2011.

[47] Stephen Voida, W. Keith Edwards, Mark W. Newman, Rebecca E. Grinter, and Nicolas Ducheneaut. Share and share alike: exploring the user interface affordances of file sharing. In *Proc. ACM CHI*, 2006.

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

[49] Pamela J. Wisniewski, Heather Richter Lipford, and David C. Wilson. Fighting for my space: coping mechanisms for SNS boundary regulation. In *Proc. ACM CHI*, May 2012.

[50] Edward Wobber, Martín Abadi, Michael Burrows, and Butler Lampson. Authentication in the Taos operating system. In *Proc. ACM SOSP*, December 1993.

[51] Ted Wobber, Thomas L. Rodeheffer, and Douglas B. Terry. Policy-based access control for weakly consistent replication. In *Proc. Eurosys*, 2010.

[52] Sarita Yardi and Amy Bruckman. Income, race, and class: exploring socioeconomic differences in family technology use. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, pages 3041–3050. ACM, 2012.