Received: 30 July 2021 | Accepted: 4 January 2022 | Published online: 26 February 2022

DOI: 10.1002/agj2.21007

Agronomy Journal

SPECIAL SECTION: BIG DATA PROMISES AND OBSTACLES: AGRICULTURAL DATA OWNERSHIP AND PRIVACY

Agriculture data sharing: Conceptual tools in the technical toolbox and implementation in the Open Ag Data Alliance framework

Aaron Ault | **Servio Palacios** | **John Evans**

Purdue University, School of Electrical and Computer Engineering and School of Agricultural and Biological Engineering, West Lafayette, IN, USA

Correspondence
Aaron Ault, Purdue University, School of Electrical and Computer Engineering and School of Agricultural and Biological Engineering, West Lafayette, IN, USA.
Email: ault@purdue.edu

Assigned to Associate Editor David Clay.

Abstract

Data privacy has become a critical issue within the agriculture and food industry. The real-time conversion of data to information has been shown to be incredibly valuable to the industry but often requires sharing data with software, platforms, customers, and regulators outside the data owner's control. While much work in this area has focused on legal protections for data privacy, less has been devoted to technical architectures to support different sharing models. This paper defines some "tools in the toolbox" for designing such systems that are accessible to both technical and non-technical audiences as well as several "sharing design patterns" using the Open Ag Data Alliance (OADA) framework. These tools and patterns are helpful in classifying and understanding both existing and future data flows in agriculture and their privacy implications.

1 | INTRODUCTION

Data privacy is an issue that affects us all. From banking information to healthcare, how our data are shared and secured is a continual concern in the modern world. The convergence of cloud computing, high-speed cellular coverage, and connected agricultural machinery has brought the issue of data privacy to the forefront in the agriculture and food industry as well. While much focus has been given to technical privacy concepts, generally, and legal questions of agriculture data privacy, specifically (Carbonell, 2016; Ferris, 2017; Nielsen, 2019; Sykuta, 2016), this work focuses on technical privacy concepts specifically in the agriculture and food industry.

Questions on data privacy and sharing in agriculture often revolve around legal user agreements. Addressing this, the American Farm Bureau Federation and an associated group of agriculture technology providers set forth the "Privacy and Security Principles for Farm Data" (American Farm Bureau Federation, 2016). AgGateway also maintains a list of concepts (AgGateway, 2017) that they recommend be included in data contract generation. The Ag Data Transparency evaluator (https://www.agdatatransparent.com/) grew out of such efforts as an organization which reads and evaluates existing legal data agreements in agriculture. Such efforts share a common theme of increasing transparency between the farmer and agriculture technology companies.

There is a fundamental limit on the privacy guarantees that can be made by technological solutions alone. The only real limit to what a data recipient can do with data to which they have read-access is a legally based or trust-based limit. A data owner may inherently trust a data recipient to act in their best

Abbreviations: API, application programming interface; DMZ, demilitarized zone; OADA, Open Ag Data Alliance; OSC, oblivious smart contract; PAC, private automated certification; REST, representational state transfer; URL, uniform resource locator.

interest with their data, or they can trust a legal agreement to govern allowable uses of their data, but there are no foolproof, technology-based limitations to what a recipient can do with data they can see. This paper describes some exciting new technical tools (see the Advanced Sharing Design Patterns section) that are better equipped to handle access control management in these "released into the wild" situations than most existing options.

This paper focuses on the technical tools (e.g., software code and system architectures) that can be used to model data privacy and sharing issues in the agriculture and food industry rather than the traditional legal (e.g., rights, promises, contracts) topics. We provide an overview of some "technical tools in the toolbox" to design such systems, intended to be accessible to audiences of both technical and nontechnical backgrounds in an effort to broaden participation in creating real-world systems for agriculture and the food supply chain that can give proper consideration to privacy and sharing. We also provide several sharing design patterns using these concepts through the open-source Open Ag Data Alliance (OADA) framework (https://github.com/OADA).

Core Ideas

- There are several composable, reusable concepts for designing privacy-focused systems in agriculture.
- Data privacy concepts can be deployed via the OADA framework
- Clear privacy design patterns exist for various types of use cases in agriculture.
- Standardization of privacy components across systems provides useful advanced patterns.

1.1 | Technical data privacy conceptual tools

Since there is no one-size-fits-all "correct" privacy model (Wilgenbusch et al., 2020), it is important for the designers of data systems in agriculture and the food supply chain to have a full understanding of the available tools in the technical privacy toolbox. It is also important for agricultural practitioners to understand the ramifications of various sharing models in order to better understand what happens to their data. We present the following list of concepts followed by some sharing design patterns using the OADA data framework.

1.1.1 | Concept 1: Access types

Data privacy can be considered as rules for three types of access to digital data: *read access* (view and understand bits of data), *write access* (change bits of data), and *admin access* (set the read-write-admin access to bits of data). Note that someone that can see encrypted data, and therefore cannot properly interpret its meaning, is not generally considered to have read access, although decryptable data does have risk of key compromise that would allow it to be decrypted by malicious actors.

1.1.2 | Concept 2: Users or accounts and permissions

Users are the core of setting permissions and performing actions on a platform. They typically refer to a conceptual entity within one system that does not cross system boundaries. For example, the same person may have used both Google Drive and Dropbox cloud storage services, but that person is considered as two distinct users—one in each platform. Users have permissions to specific resources on a platform.

1.1.3 | Concept 3: Scopes and tokens

A token is basically an app-specific password that external systems send when they make a data request. Typical software-level requests for data are self-contained—the request is allowed to proceed if the token provided on the request is valid regardless of other requests that may have taken place earlier or later. Tokens are generally handed out by users, with a "scope" defining what the application or service is allowed to do on their behalf. Scope can restrict a token to particular types of data (harvest, planting, etc.) or to data that has certain properties (harvest data from only this field). Note that scopes are different than permissions: users are limited by permissions, and tokens are limited by both scope and the user's permissions who handed out the token. The user typically cannot grant privileges to a token beyond what the user is allowed to do themself. The time when a user authorizes a token represents the end of technical privacy protection and the start of legal protection, hence, it is a good time to make privacy statements available to users as done in the OADA framework. For example, when a user sees the screen "Do you want to allow Application ABC to read your harvest data," on that same screen is a link to the platform's privacy and use statement.

1.1.4 | Concept 4: Shares

Local users on a platform can share data with other users. Sharing is initiated by a user with admin access to particular data and typically does not involve explicit consent on the part

of the recipient. From the recipient's perspective, they have a set of incoming 'shares' from other users. An odd consequence of this concept is that the shares list belongs to the data recipient, but they cannot restrict other users from adding to their shares list.

1.1.5 | Concept 5: Data organization

Most people are familiar with organizing their data in hierarchical graphs, that is, files and folders in common parlance. These structures have immense ramifications in a person's ability to understand sharing rules—sharing a folder means sharing everything in that folder. An alternative to sharing hierarchically is property-based sharing, that is, share data based on a property like a time range or geospatial area. Many real-world sharing scenarios require both. For example, a farmer that wants to share this year's harvest maps for a particular field, which means that their total as-harvested data must be restricted to only the points that fall inside that field's boundary and were harvested in this year.

1.1.6 | Concept 6: Synchronization and caching

Most data sharing from users to applications involves an application making a copy of a user's data from one platform into their account on another platform. How the integrity of this cached copy is maintained over time is a crucial part of the user's data sharing experience. Possible questions that arise include the following: Do they have to manually import and export things between platforms? Does a platform have to wait some interval before it can ask if new data is available? Can data be pushed in real-time? Systems requiring import–export semantics may, for all practical purposes, be incapable of achieving data privacy and sharing goals for many users because of the user time required just to move the data. This occurs frequently in production agriculture where a farmer may prefer to limit a service provider's access to data from only a few fields, but due to lack of time to import and re-export data, the farmer simply gives the data card from their combine with their entire season's recorded data to their service provider and hope they only take the data they really need.

1.1.7 | Concept 7: Obfuscation

This is often the de facto method in agriculture by which a company prevents other companies from accessing data in their systems. This typically happens when a company makes up their own file formats for their data and then does not publish descriptions of those formats for other software developers. In layman's terms, imagine a farmer sharing a spreadsheet to their agronomist that the farmer created themselves, full of cryptic abbreviations that only the farmer understands, and then the agronomist would be expected to reverse engineer the meanings in order to try to do their job. It achieves some effective level of privacy but sacrifices reusability and portability.

In addition to simply file formats, critical to practical data sharing is enabling platforms to agree on how to authorize, find data, read and write data, request updates, negotiate formatting, alter permissions, among other actions. If these methods also are obfuscated from software developers in the industry, then privacy is still achieved by the fact that no other software can figure out how to use it.

Relying on obfuscation can result in overconfidence with your data privacy because often it only reliably works when a malicious attacker is not interested in your data in the first place. If data has real value, interested adversaries can often reverse engineer the obfuscation. Primarily, however, if data sharing is a goal, obfuscation is a very poor tool because it largely prevents sharing in the first place.

1.1.8 | Concept 8: Cryptography

Cryptography uses mathematical techniques to provide data integrity and confidentiality. Integrity ensures that a recipient can tell if data has been tampered with, often through hashing functions (described later). Confidentiality ensures that an adversary cannot derive information from an encrypted message. As a corollary to confidentiality, key-based semantics can also provide identity, in other words, you cannot produce the correct encrypted text from a clear text if you do not know the password (private key).

1.2 | Implementation in the OADA framework

While a full exposition of the OADA application programming interface (API) framework is beyond the scope of this paper, we here describe a few features that will help us demonstrate some of the sharing tools as real-world examples. While the sharing patterns and tools described in this paper can generically be applied in a wide variety of circumstances, the concepts and language to understand and implement sharing that we have developed within OADA are extremely useful to a cohesive treatment of this topic. Therefore, we present here a minimal overview of the necessary terms and concepts that we will use in this paper.

The OADA framework defines a standard for how software code interacts with data on any OADA-conformant system. Digital systems exchange data primarily through APIs. In the most common type of API, a representational state transfer

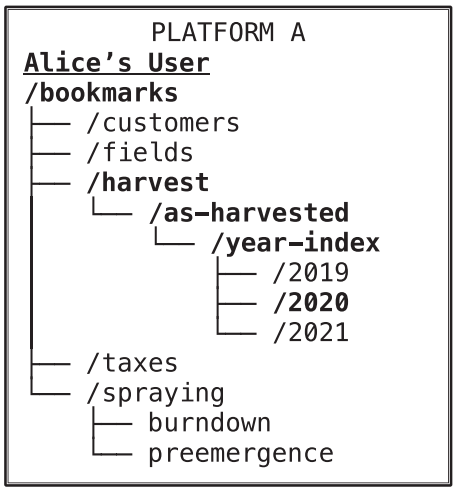

FIGURE 1 Simplified example bookmarks tree representing a standardized structure for where to place as-harvested data relative to other data for a farmer in an application programming interface. The highlighted path would be specified as `/bookmarks/harvest/as-harvested/year-index/2020`

(REST) API, data access generally takes the form of a series of uniform resource locators (URLs) where software may find, read, write, and delete digital resources. Such APIs are widely accepted and well known as the standard form of data exchange between most agriculture data providers today. The OADA codifies existing best practice in REST API design, providing standard requirements that any API must meet to be considered OADA-conformant. Existing APIs can be adapted to become OADA-conformant, or new APIs can be developed using OADA principles from the start. Understanding some of these requirements and concepts will help to understand how API-driven platforms can share data, which is the primary focus of this paper.

The first central feature of an OADA-conformant platform is that the data platform has users—distinct entities capable of logging in. Each user has a set of resources organized in published hierarchical structures formed when resources link to each other. These hierarchical structures are known as a "directed graph" or just "graph," and resemble a folder hierarchy on a normal computer system. A user's software tools use the graph structures that it knows about to find the data needed on a user's platform to accomplish tasks for them. The root of each user's graph is called their "bookmarks." Most real-world graphs tend to grow downward and have few cycles (i.e., lower folders do not point back up to higher folders); therefore, we often refer to bookmarks as a "tree" structure. See Figure 1 for a visual example.

A good analogy for this concept is to envision if every farmer and agronomist used Google Drive to store their data, and they all used the same folder names and file names to organize their data. Their software could then accomplish many data-centric tasks for them by knowing how to connect to Google Drive and interacting with those standard structures. Sharing data would then largely mean moving data between those user accounts.

A bookmarks tree is arranged according to specified graph schemas typically with higher-level nodes representing a semantic organization (i.e., segregating types of data) and lower-level nodes representing an index-based organization (i.e., splitting large data sets of the same type of data into smaller consumable chunks). For example, consider the highlighted graph path of Figure 1, represented by the URL path `/bookmarks/harvest/as-harvested/year-index/2020`. This has the higher-level semantic organization of "harvest" and "as-harvested," indicating as-harvested data is to be found here as opposed to spraying data or planting data. The lower-level "year-index" of 2020 means that only the as-harvested data from the year 2020 is expected to be found at that level. Any software looking for as-harvested data would look on a user's data platform according to this API path when discovering if the user has as-harvested data for 2020.

The concept of the semantic structure of a bookmarks tree is recursively reusable throughout any system. For example, a user may have many customers who are other users on the same platform. In this case, it is useful to first organize their list of customers, and then, for each customer, store a link to that customer's bookmarks tree. Since the hard work of modeling the data space for a particular use case was already done once, reusing this tree structure where applicable both promotes interoperability and improves code efficiency.

Users share resources with other users by assigning permissions on a resource. Any resources reachable downward in the graph inherit the permissions of the shared resource. For example, if a user has access to `/bookmarks/harvest`, then they also have access to `/bookmarks/harvest/as-harvested`. In addition to a bookmarks tree, each user has a shares resource, that is, a list of links to all resources that are shared with them from other users.

Applications obtain a client identifier through dynamic client registration, allowing systems that have never before known about each other to be connected by the user, putting the user in control of their data. The software statement provided by the application during registration includes a link to their privacy and use statements that is shown to the user when they attempt to authorize access to their data. Users authorize tokens for registered clients using scopes that can be content-type based (i.e., all harvest data) and graph-based (only harvest data from 2020). As described above in Concept 3, a token is basically a unique password that applications send along with their requests to a platform. The application acts on behalf of the user that granted the token to their data, and the scope associated with that token specifies restrictions as to what the user will allow that token to do on their behalf.

Applications can register a "watch" on any resource or subtree of resources for real-time change feeds, that is, a real-time stream of data representing all the changes that happen to a resource. When any application writes to a resource, an idempotent merge document is created, which is a document representing data to "upsert" into parts of a given resource. Upsert is a contraction of "update" and "insert," meaning that any specified parts of the resource will be replaced if they already exist or created if they do not. The change document is idempotent because it can be merged into the resource multiple times and the end state of the resource will be the same regardless of how often that merge was applied since it will just keep replacing the same parts of the resource with the same data each time.

OADA defines the ordered set of idempotent merge documents as the changes necessary to transform a resource from one version into any subsequent version. Any external service can use them to do things like synchronizing between platforms, transforming into different formats, and maintaining filtered subtrees for sharing purposes, to name a few of many examples. In fact, any two OADA-conformant platforms can inherently synchronize any subtree automatically by using OADA's built-in ability to replay change feeds on a remote platform. In other words, since OADA defines precisely how writes occur in an OADA-conformant API, one OADA-conformant platform inherently knows exactly how to write to a destination OADA-conformant platform to make the destination platform's copy of a resource look the same as its own copy. It can simply "replay" the original change at the destination.

And finally, the Trellis framework (https://github.com/trellisfw) project is a parallel brand of the OADA framework that focuses the OADA framework's capabilities specifically for supply chains such as those found in fresh produce. Trellis adds to OADA the ability to apply digital signatures to documents to verify authenticity as they are exchanged with trading partners as well as supporting several of the advanced sharing use cases that we will describe in later sections.

2 | SHARING DESIGN PATTERNS

Design patterns can prove useful when designing data sharing systems. We present several patterns that have proven useful in real-world industry cases presented in the context of OADA. The patterns are summarized in Table 1.

2.1 | Pattern 1: Shared bookmarks

Each user's bookmarks resource is the same actual resource (Figure 2). This model is good for small groups of people who all trust each other, and all have similar permissions; they simply need to have the exact same data served to all their applications such as a manager and all their employees on a small farm. This model avoids any potential data inconsistencies in that all users are updating the same actual resources, but no single user can act in their own bookmarks separately from the group. It is nearly equivalent to all employees on a farm using the same user account but has the advantage of knowing which users made which changes, and users can have their access revoked entirely as needed such as in the case of an employee no longer working on that farm. This pattern can be succinctly represented by stating that the path `/bookmarks` for every user is the same underlying resource.

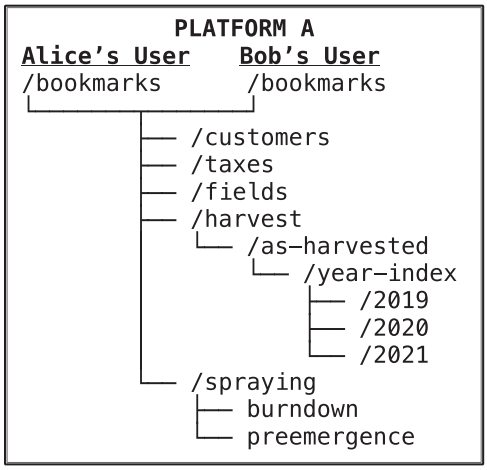

FIGURE 2 Shared bookmarks pattern. Both users on the same platform use the same bookmarks tree. Alice's apps and Bob's apps see the same resources at all endpoints: easy to stay in synchronized, but no cross-user privacy

2.2 | Pattern 2: Shared nodes

Each user has a different top-level bookmarks resource, but the parts of their bookmarks tree, which should be shared among other users are linked as the same actual resources (Figure 3). For example, a farm manager and his accountant may have a common `/bookmarks/taxes` resource, but the farm employees' bookmarks trees do not share that endpoint. An employee could use an application that creates their own `/bookmarks/taxes` resource and still function properly, but they would not be able to see the farm manager's resource. This model requires that graph paths perfectly match privacy goals of the data owner since they do not limit access to the graph's contents below the shared node.

2.3 | Pattern 3: Service users

Many services often cooperate to achieve overall goals. It can be a convenient means of service isolation and tracking to

TABLE 1 Summary of sharing design patterns

No.	Pattern and description	Advantage	Disadvantage	Examples
1	Shared bookmarks: Each user's bookmark is the same resource	Trivial synchronization: all users see same data	Nothing private between users	Employees on a small farm sharing all data
2	Shared nodes: Only some parts of data are the same resources between users	Trivial to keep shared data synchronized with additional setup	Some data can be private, but others easily shared	Employees on a small farm with limited access to some data
3	Service users: Create a special-purpose user for each intended service	Isolates services and makes them easy to track	Same as no. 2	Microservices that operate on user's data
4	Single-authority filtered graph: Use a service to maintain a filtered copy of all shared resources	High control over permissions; easy to merge data from multiple platforms	Data only flows one direction	As-applied fertilizer maps from service providers
5	Multi-authority filtered graph: Use a service to synchronize data to and from multiple platforms	High control over permissions; changes replicated across platforms	Data conflicts between platforms; prone to infinite cycles	Field boundaries shared across multiple applications and tractors on a farm
6	Mirror user: Create specialized users to hold all data shared with each outside source	Easy to know what is shared and to whom at all times; easy to synchronize	Requires many users and using services to maintain sharing to those users	Agronomist with multiple farmer clients who all use apps to see recommendations
7	Single-token shares: One user shares data to another user as a "suggestion" and the other user accepts	Only requires one low-privilege token to perform sharing	Recipient must appropriately handle incoming shares	Farmer-to-farmer sharing on the same platform
8	Domain admin with in-and-out demilitarized zones: Create specialized users for all sharing partners and maintain independent copies of everything coming and going from and to that partner	Clearest and cleanest sharing model for large numbers of customers or vendors; enables approval process for incoming data	High level of complexity and maintenance required for approvals and sharing	Enterprise sharing of food safety documents; co-ops with many farmer customers sharing data in-and-out

create individual users (that are prohibited from logging in) for each service and then setting the bookmarks tree for that user to be the same resource as the primary tree it was supposed to operate on without this model. Applying the shared nodes pattern above tends to create the best blend between efficiency and access granularity, for example, the service user has access to fields but not taxes. The service administrator then simply needs to communicate a token to the service tied to that service user. In this way, the actions of this particular service can be isolated, permissioned, and tracked easily just as if it were a farm employee.

2.4 | Pattern 4: Single-authority filtered graph

Each user has their own bookmarks resource, but a copy of parts of a master bookmarks tree is maintained across users rather than sharing resources directly (Figure 4). This model is more complex than Patterns 1 and 2 because it requires an actively running service with permissioned tokens capable of writing to all involved users. This service must maintain synchronization in real time between copies. It sets a watch on the nodes in the master tree and replays changes in the copied resources that are linked in the downstream user's trees. Because the master copy is considered authoritative, it only maintains a one-way synchronization. This means if the downstream users write to their copies, this data stays in their own tree and does not migrate up to the master copy. If the master copy overlaps with data in the copies, the data in the copy is overwritten when changed in the master.

This allows granular control of sharing by a user to any other users according to whatever rules drive the filter-plus-sharing service. This model can perform property-based sharing, that is, sharing only some data points within the same resource based on their value and not others. For example, only sharing as-harvested data points that fall within a particular field boundary. If data points in the harvest resource lie outside a given boundary, they could be excluded from sharing.

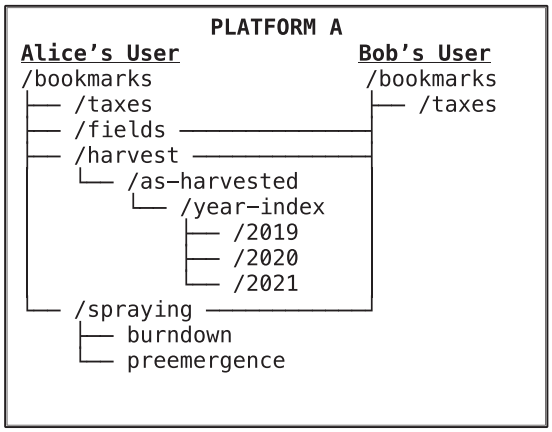

FIGURE 3 Shared nodes pattern. Alice's apps and Bob's apps see the same resources under `/bookmarks/fields` and `/bookmarks/harvest` but not others such as `/bookmarks/taxes`. Alice's "tax" apps interact with her "taxes" endpoint and not Bob's, but changes to her "fields" endpoint will be reflected in Bob's apps as well

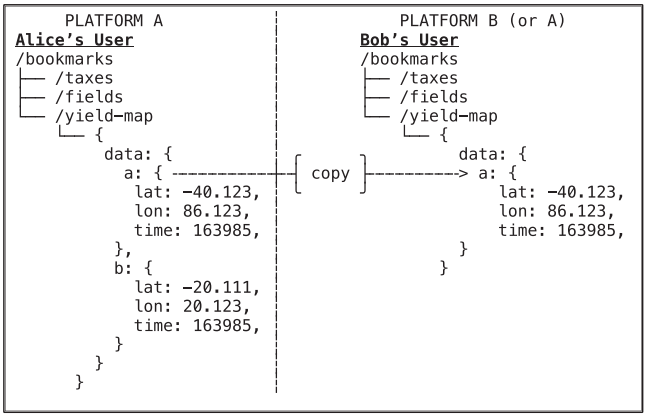

FIGURE 4 Single-authority filtered nodes pattern. Alice's user on Platform A is the authoritative source for data point "a" within the resource at `/bookmarks/yield-map`. A separate service maintains a copy of Alice's point "a" under Bob's user on his own platform, but Alice's point "b" remains only on her platform. If Bob changes point "a" on his platform, Alice's user does not see his change

Since the service performs its task using the OADA API, and any OADA-conformant platform speaks that API, then this is the first of the sharing design patterns that does not care if the two users are on the same platform or different platforms. In other words, this model allows a user to share data just as easily across platforms as across users on the same platform.

As a corollary, consider a person with users at multiple platforms where data is created. They are each authoritative over the data in their platform. The user would like to collect all the data together into a single user on a single platform. For example, a farmer that uses multiple service providers to spread their fertilizer. Each provider would have as-applied

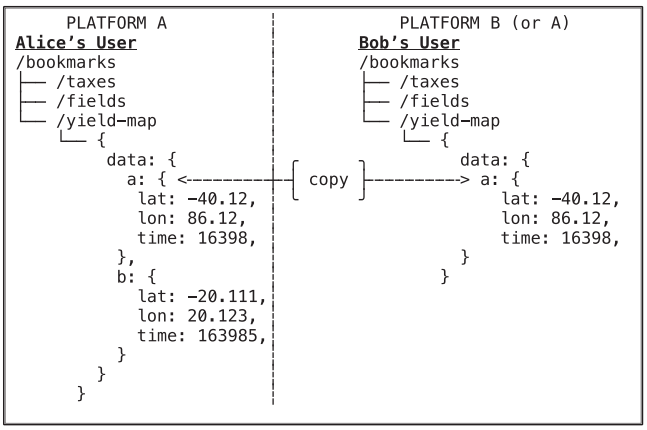

FIGURE 5 Multi-authority filtered nodes pattern. Both Alice and Bob have a copy of data point "a," but a change to either side is replayed at the other. The only difference from the single-authority model above is the bidirectional arrow on the copy service

fertilizer data for the farmer on their own platform. The farmer would like to get all of their as-applied fertilizer data into one place, so the farmer has a user account at their preferred cloud provider to collect the data from the farmer's users on the various service provider platforms.

For the sake of simplicity, assume all the platforms store as-applied fertilizer at the location `/bookmarks/fertilizer/as-applied`. The farmer's preferred platform can configure the resource at each outside platform to replay their changes under the farmer's user at the preferred platform. If the keys that identify the underlying data points in the `/bookmarks/fertilizer/as-applied` tree are randomly generated strings, then the net result at the preferred platform after merging will be a subtree with the same structure but containing all the merged data from all sites. In effect, using random string keys allow resources to be merged together without conflict because the data points never overlap (i.e., conflict) at the destination.

In this way, each platform can be the authoritative copy for the data they generate. Counterintuitively, despite the grower thinking of their primary platform as the definitive place where they store all their fertilizer data, it is actually not the authoritative copy in this case since it simply accepted the data generated at other places.

2.5 | Pattern 5: Multi-authority filtered graph

Each user has their own bookmarks and their own copies of all shared resources, but either user may write to their copy and expect it to be reflected in the other copy. In other words, neither copy is authoritative (Figure 5). This is the most commonly used case for real-world users, but it is also the most

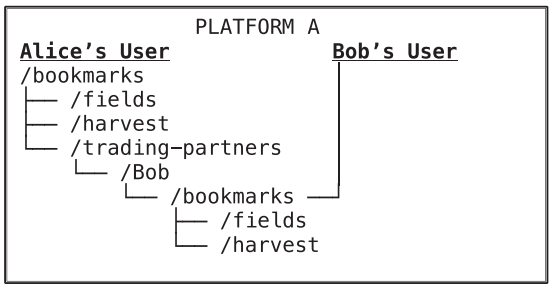

FIGURE 6 Mirror user. Alice's user creates a user for Bob on Alice's platform and links Bob's bookmarks under her own tree. Alice's apps can interact with Bob's bookmarks as Bob. Alice still has her own fields and harvest data

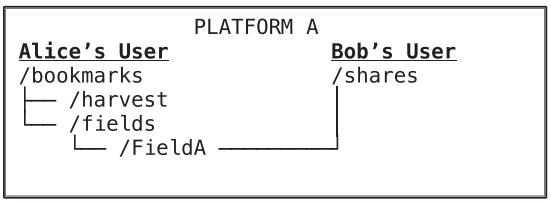

FIGURE 7 Single-token shares pattern. Alice's user can share Field A with Bob's user, but Bob must "accept" the share by deciding what to do with that field for his own bookmarks tree. Alice only needs a token with permission to her own user accomplish this pattern

complex. If the exact same data paths are not written on either side, conflicts do not overwrite. In many situations, simply placing data at a key that is a randomly generated string ensures no conflicts, although this can suffer from duplicate data if the same data is written on both sides and then synchronized. The vast majority of cases where conflicts may occur can be reasonably solved with a last-write-wins conflict resolution model, but this should be considered carefully when dealing with such situations. For example, if a user's field boundaries contain a field named "Smith30," and one employee changes the name to "Smith29" in a tractor while another employee changes it to "Smith31" on their phone, whichever change is written last would be considered the "correct" name. Care must be taken here to avoid infinite cycles, that is, Platform A writes Smith30 while Platform B writes Smith29, then Platform A replays the Smith30 write to Platform B, but Platform B is also simultaneously replaying the Smith29 write to Platform A, which Platform A then replays back to B, and so on, and so on.

As with the single-authority sharing design pattern, a separate synchronization service is needed if filtering is to be performed, and the users involved need not be on the same platform.

2.6 | Pattern 6: Mirror user

When a user wants to share data with a remote platform—or even another user—they create a separate user whose bookmarks tree represents the full set of data to be synchronized and optionally link that new user's bookmarks tree into their own. (Figure 6). By storing an exact bookmarks tree of everything destined for a different user or outside platform, it is very simple to audit and debug synchronization. It allows the primary user to use an application as if they were the other user to ensure the shared user's experience is appropriate.

This effectively splits the task of synchronization into two phases: first, make the mirror tree look like you want, then make sure the remote looks just like the mirror. If a user ever wants to ask a question like "Exactly what data did I share to company X?" this model works very well. Because the synchronization is logically split from the sharing, the built-in OADA synchronization can tackle keeping the mirror tree synchronized with the remote, vastly simplifying the task of the sharing application. Conveniently, this model also does not require the sharing application (which only writes to the mirror tree) to have a token authorized for the remote user. Only the synchronization of the mirror to the remote needs such a token.

2.7 | Pattern 7: Single-token shares

A user can choose a particular subtree of their bookmarks to share with another local user and grant that user permission to that subtree (Figure 7). This granting of permission causes the shared node to appear in the list of shares for the other local user. This action does not link the shared resource into the appropriate place in the other local user's bookmarks tree, leaving that task up to another application or service. This pattern has the advantage of not requiring the original user to have permissions to the other local user's data.

This model has merits for a clean separation of permission—the original user is essentially just suggesting to the other local user that they accept this data. However, it suffers from added dependency that it requires a different piece of software to identify the newly shared subtree and decide what to do with it (i.e., where to put it in the other local user's bookmarks tree). For example, an agronomist that shares a planting prescription with a farmer might prefer that their map just show up in the planter tractor for the farmer rather than requiring action on the part of their customer.

2.8 | Pattern 8: Domain admin with in-and-out demilitarized zones

Each user is separate and has their own bookmarks tree, but a top-level domain admin user created all the other users in their

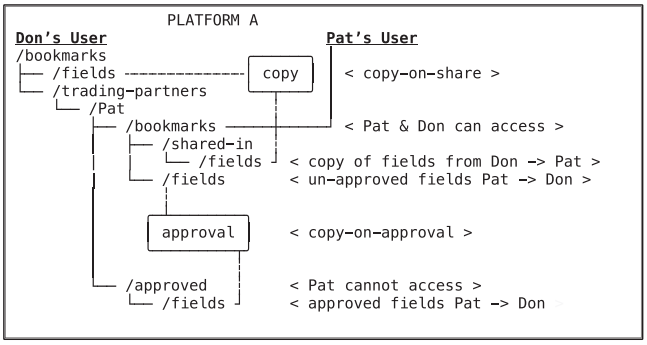

FIGURE 8 Domain admin with in-and-out demilitarized zones pattern. Don, the domain admin, created a user for Pat, the trading partner. Pat sends Don fields at `/bookmarks/fields`, which Don accesses at `/bookmarks/trading-partners/Pat/bookmarks/fields`. Don sends fields to Pat in `/bookmarks/trading-partners/Pat/bookmarks/shared-in/fields`, which Pat's apps access at `/bookmarks/shared-in/fields`

domain and linked their bookmarks trees under the domain admin's bookmarks tree (Figure 8). For the purpose of this discussion, the domain admin user is the authoritative user implementing this scheme, and the partner users are the mirror users that the domain admin created on their platform to facilitate sharing. Each partner user's tree acts like a mirror user described in Pattern 5 above, that is, its bookmarks tree can be directly synchronized out to another platform controlled by the partner user. This model works quite well in enterprise scenarios where an enterprise wants to provide a user on their platform to each of their vendors or customers.

Sharing can be bidirectional (both domain admin to partner and partner to domain admin), but each direction carries its own unique considerations from the standpoint of the domain admin. For data shared out (from domain admin to the partner), it is typically desirable to share copies of data rather than originals to the partner. This way, the partner user's copy is completely isolated, and the partner user cannot gain write access to original internal company documents. This treats the partner user's bookmarks as a sort of "demilitarized zone" (DMZ) separate from the core data of the company.

This DMZ concept becomes even more effective when considering data shared in (partner to domain admin). Treating the partner user's bookmarks as untrustworthy means an approval process must occur before a document from a partner is accepted by the domain admin. To accomplish this, another bookmarks tree is maintained by the domain admin that represents the approved tree for that partner, but it does not grant the partner user direct access to it. Writes that occur to the DMZ bookmarks tree by the partner can then be promoted to the approved tree through rule-based automated processes, a manual approval process, or some combination of both.

Therefore, a full implementation of this model involves maintaining three bookmarks trees for each partner: the shared-in DMZ bookmarks tree, the approved copy of the shared-in DMZ bookmarks tree, and the shared-out DMZ bookmarks tree. Since the partner user needs access to both the shared-in DMZ tree (to send data to the domain admin) and the shared-out tree (to receive data from the domain admin), it is convenient to link the shared-out DMZ tree under the shared-in DMZ tree so that the same partner user can access both. Care must be taken to remember that the directions of in and out are relative to the user in question, that is, "in" to the domain admin is "out" to the partner user and vice versa.

The primary simplification here is that a token with only permission to the domain admin's bookmarks tree is inherently capable of caring for the underlying customer users. Combining this pattern with the previous patterns allows for a single master bookmarks resource to be maintained for an entire enterprise but still relegate particular users in the company to perform specific roles through their own bookmarks trees while still able to have permission to care for customer needs.

Most important about this model is that almost all other sharing models can be represented using this model since the partner user could exist on another platform as simply a mirror of the one on the domain admin's platform. It does not matter where the user resides; what matters is the model that there is a shared bookmarks structure defining the API and a clear understanding of which data goes where when sharing in and out.

3 | ADVANCED SHARING DESIGN PATTERNS

Having a standardized API framework enables some exciting privacy paradigms that are difficult to conceive otherwise. This section discusses two advanced cases that enable data owners to have greater control over their data sovereignty while supporting regulatory and reporting needs. First, it is important to introduce two background concepts relevant to more advanced data sharing in food and agriculture: hashes and digital signatures.

3.1 | Cryptography: Hashes and digital signatures

There are two basic concepts in cryptography relevant to these privacy design concepts: hashes and digital signatures. A "hash" is the result of running an arbitrarily long stream of data through a function that produces a deterministic yet hard-to-predict fixed-length piece of data (known as a hash of the input data). A good hashing function has the property that it is considered computationally infeasible to construct

an input data stream that produces a particular desired output hash, and it is considered infeasible to invert the function to learn the original input data using only the hash. Because of this property, if you have the hash of some piece of information and someone gives you what they claim was the original information at a later time, you can easily check their claim by computing the hash of what they give you and see if it is the same as the hash you have.

The second useful concept in cryptography is a digital signature. A digital signature is similar to a hash in that it is generally produced by taking a piece of input data and computing an output. The input data for a digital signature is often a hash of the document being signed. The signature function also takes a password (known as a "private key") as input and produces an encrypted (but decryptable) output. Also like the hash, it is considered computationally infeasible to construct an input that produces a desired output or to take an output and reconstruct the input or private key that produced it.

Critically, the private key has a corresponding public key with the special property that a message encrypted with the private key can only be decrypted with the public key. It is computationally infeasible to figure out the private key given only the public key. Digital signatures use this property such that the trusted signer keeps their private key secret and puts their public key on a list of trusted public keys. When a digital signature and corresponding document are to be validated, one can check that a trusted private key did indeed create the signature for the document by decrypting the signature with the trusted public key and comparing the decrypted hash with the hash of the received document.

3.2 | Advanced sharing design patterns

3.2.1 | Pattern 9: Mask and link

An extremely common occurrence in many areas of life is the need to provide certification to a downstream customer or regulator that you have a piece of information, but you do not want to share it unless absolutely necessary. Most businesses would like such processes to be automated to ensure that data shared outside their firewall adheres to company-wide privacy policies and to avoid polluting the daily logistical tasks of employees with tedious regulatory burdens that fall outside their core business. The combination of mask and link tool with the domain admin with in-and-out demilitarized zones pattern, as illustrated in Figure 9, results in a very powerful and flexible privacy solution.

The basic idea behind mask and link is that a masked copy of a resource in OADA can be created that obscures some data in the resource (but not necessarily all) by replacing the sensitive parts of the resource with a hash of the original and a URL (i.e., a link). In the event of a later audit, the hash can

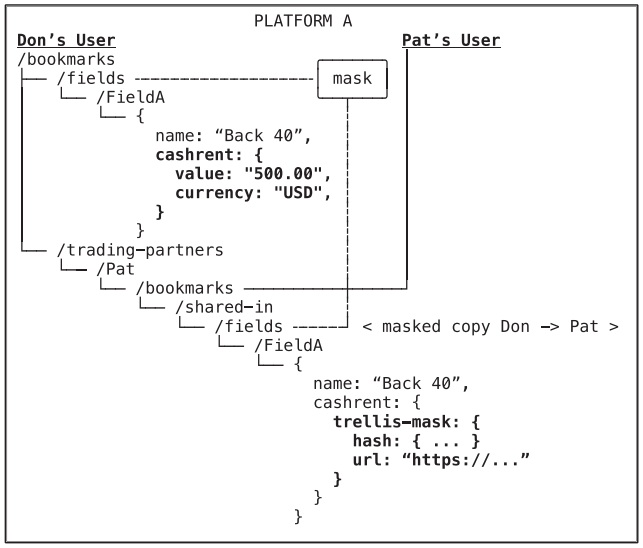

FIGURE 9 Domain admin with mask and link pattern. Using the domain admin sharing model, Don shares his fields with Pat's user. However, Don does not want to share his cash rent amounts for each field with Pat, so his mask and link service automatically masks it away before sharing a masked copy to Pat

be used to prove that the original data has not been changed since it was originally shared, and anyone with permission to the OADA platform where the original lives can access it directly to verify. This allows the sharer to completely control who is allowed to see their sensitive information even after the masked version is shared outside of their control because the downstream recipient of the masked document must access the sharer's platform to retrieve the actual data. A digital signature is also applied to the document after masking that can be used to verify if a trusted party performed the masking process.

This feature is available open source as part of the Trellis framework (https://github.com/trellisfw) including a library for masking and signing any JavaScript object notation (JSON) document, a microservice to automatically mask documents as they arrive at an OADA platform, and a web application to verify masks and signatures for authorized parties.

If a company is forced to publish some piece of data to an outside platform for purposes of certification or traceability, they can use mask and link to achieve fine-grain control of what they approve for release outside their company and what they choose to keep internal.

3.2.2 | Pattern 10: AGAPECert

Going one step further than mask and link, which merely obscures data, AGAPECert (https://github.com/agapecert) is a new solution that enables trusted claims about data to be produced and shared instead of sharing data or masks This

model is useful in a wide variety of scenarios: a produce shipper wants to certify that they have all the necessary records to trace where their products came from but doesn't want to leak to all their downstream customers about exactly who they buy from; a livestock producer who wants to certify they have met environmental regulations for manure application without needing to share the actual GPS paths of their manure spreading equipment: or a fishing vessel would like to prove they caught fish within the boundaries of allowable waters without divulging their exact fishing grounds.

AGAPECert does this by providing an auditable, generalized, privacy-enabling certification framework (Interiano, 2020). AGAPECert, like OADA and the Trellis frameworks, is an open-source project of the Open Ag Technology and Systems Center at Purdue University. Published at the AGAPECert code repository (AGAPECert, 2020) are several libraries, services, and applications that can be used to implement private automated certifications (PACs) for any industry. A PAC is basically an attested claim about private data that can be shared in lieu of sharing the actual data.

It is beyond the scope of this paper to fully describe AGAPECert's functionality, but at a basic level AGAPECert allows a privacy-conscious data owner to run preapproved and certified software code in their own environment on their private data to produce a certification (a PAC) about that private data (Interiano, 2020). That certification can then contain proof of correct code execution that can be reasonably trusted by recipients of the certification and is still able to be audited in the future. The software code producing the certification is called an oblivious smart contract (OSC), which can automatically produce certifications about the underlying private data, hence the "automated" part of the PAC name.

A regulator can approve a given OSC for use in automating certifications in an industry, and a data owner can also approve or have their trusted agent approve the OSC prior to running it themselves on their own data, that is, the data owner could look to an industry organization's approval that a particular set of code adheres to proper privacy and will faithfully produce appropriate certifications. This model allows for a company not only to obscure data released to another platform like mask and link, but it also allows for the outside platform to achieve some particular guarantees about the data transformation and processing as well.

4 | CONCLUSION

Data privacy continues to be a serious issue in the agriculture and food industry, and it will only become more important in the next several years. A survey of concepts in the technical toolbox to achieve various privacy and sharing goals was presented as well as examples of how to apply those in various design patterns using the OADA framework. The concepts presented in this paper provide a comprehensive framework and language by which to understand and compare various data sharing models and their privacy implications. Importantly, two newly enabled advanced sharing models are described: mask and link and AGAPECert. The advanced cases provide significantly improved data sovereignty paradigms over existing options and have been implemented using the OADA framework.

ACKNOWLEDGMENTS

Sponsorship for this work was provided by Foundation for Food and Agriculture Research (FFAR) under award 534662 through the Open Agriculture Technology and Systems (OATS) Center at Purdue University.

AUTHOR CONTRIBUTIONS

Aaron Ault: Conceptualization; Data curation; Formal analysis; Funding acquisition; Investigation; Methodology; Project administration; Resources; Software; Supervision; Validation; Visualization; Writing – original draft; Writing – review & editing. Servio Palacios: Conceptualization; Formal analysis; Investigation; Methodology; Project administration; Resources; Software; Supervision; Validation; Visualization; Writing – original draft; Writing – review & editing. John Evans: Conceptualization; Formal analysis; Investigation; Methodology; Software; Validation; Visualization; Writing – original draft; Writing – review & editing.

CONFLICT OF INTEREST

As of the time of this writing, Aaron Ault is the Chief Technology Officer of, and Servio Palacios is currently employed by, The Qlever Company, LLC, a software consulting company whose work includes helping companies include the OADA and Trellis open source projects in their products and infrastructure. Dr. John Evans has no conflicts of interest.

ORCID

Aaron Ault https://orcid.org/0000-0001-7060-6559

REFERENCES

AgGateway. (2017). *Data privacy and use white paper: Version 1.2 updated*. https://www.aggateway.org/eConnectivityActivities/Committees/DataPrivacySecurity.aspx

American Farm Bureau Federation. (2016). *Privacy and security principles for farm data*.

Carbonell, I. M. (2016). The ethics of big data in big agriculture. *Internet Policy Review*, *5*. https://doi.org/10.14763/2016.1.405

Ferris, J. L. (2017). Data privacy and protection in the agriculture industry: Is federal regulation necessary? *Minnesota Journal of Law, Science & Technology*, *18*, 309.

Interiano, S. E. P. (2020). *Auditable computations on (un)encrypted graph-structured data* (M.S. thesis Purdue University Graduate School, West Lafayette, IN). https://doi.org/10.25394/PGS.12721169.v1

Nielsen, K. (2019). Big data and sensitive data. In A. Emrouznejad & V. Charles (Eds.), *Big data for the greater good* (pp. 183–204). https://doi.org/10.1007/978-3-319-93061-9_9

Sykuta, M. E. (2016). Big data in agriculture: Property rights, privacy and competition in ag data services. *International Food and Agribusiness Management Review*, *19*, 57–74. http://doi.org/10.22004/ag.econ.240696

Wilgenbusch, J., Lynch, B., Hospodarsky, N., & Pardey, P. (2020). *Dealing with data privacy and security to support agricultural R&D: Technical practices and operating procedures for responsible agroinformatics data management*. CGIAR Big Data Platform. https://hdl.handle.net/10568/108095

How to cite this article: Ault, A., Palacios, S., & Evans, J. (2022). Agriculture data sharing: Conceptual tools in the technical toolbox and implementation in the Open Ag Data Alliance framework. *Agronomy Journal*, *114,* 2681–2692. https://doi.org/10.1002/agj2.21007