RAPPD: A language and prototype for recipient-accountable private personal data

Yuan J. Kang
Department of Computer Science
Columbia University
New York, New York 10027–7003
Email: yuanjk@cs.columbia.edu

Allan M. Schiffman
CommerceNet
Palo Alto, California
Email: ams@commerce.net

Jeff Shrager
CommerceNet & Stanford
Palo Alto, California
Email: jshrager@stanford.edu

***Abstract*—We often communicate private information in informal settings such as email, where we trust that the recipient shares our assumptions regarding the disposition of this information. Sometimes we informally express our desires in this regard, but there is no formal means in such settings to make our wishes explicit, nor to hold the recipient accountable. Here we describe a system and prototype implementation called Recipient-Accountable Private Personal Data, which lets the originator express his or her privacy desires regarding data transmitted in email, and provides some accountability. Our method only assumes that the recipient is reading the email online and on an email reader that will execute HTML and JavaScript.**

I. Introduction

Doveryai, no proveryai. (Trust, but verify.)
– *Russian Proverb*

We often hand over our private information to others under the assumption that there is mutual agreement about what is to be done with it. For example, we give our doctors details of our health and *trust* – that is, *assume* – that they will keep it to themselves, store it in protected databases, and so on. Perhaps they will communicate it to other doctors in the interest of our health, but we assume that these communications will either be confidential, or appropriately anonymized.

In this, and in the numerous similar transactions we engage in every day, we are depending upon a complex network of technologies, but even more so upon a complex network of *trust*. Some of the details of what is intended to happen to our information is explicated by policies such as HIPAA, or by the licenses that we click "I agree" on, but never read. Even if we did read these policies and licenses, we could not fully understand them. In part this is because they are complex and written in "legalese" (i.e., language deliberately meant to be understood only by specialists), but also because they were written and are agreed to by us in particular contexts, and almost certainly do not cover all possible future cases. For example, contracts, such as website privacy policies, can be terminated by a bankruptcy court – a place where many, if not all, of the startups who have your personal data will end up.

The privacy literature is replete with technical approaches to securing, communicating, and storing private information that we have communicated, but for the most part, this literature rests upon a fundamental, and fundamentally invalid assumption, that there is a well defined agreement between the parties as to the intended disposition of the information. In some very rare cases, mostly when there are lawyers involved on all sides, this is nearly true, but generally we take it on faith that the parties with whom we share our private information aren't going to do something inappropriate with it. Yet it is not technically difficult to state what one expects a recipient to do with one's private information. Indeed, we do this all the time in regular, interpersonal communications.

When we communicate information to others in non-professional settings, such as in emails or text messages, or even just chatting, we generally assume that the recipient shares our assumptions regarding the privacy of this information. If we are unsure that our privacy assumptions are shared, i.e., when we think that there is the possibility that these privacy assumptions will be violated, or when we desire them to be violated (as, for example, when we wish our friends to indeed tell other friends that we are ill), we will almost always make this explicit. For example, we might add a note to an email such as: "Please don't pass this on without asking me first." or "Feel free to pass this on to your [co-workers/friends/family]." And many of us are familiar with the paragraph of disclaimers often appended to professional communications, for example from lawyers or doctors, with wording such as: "This message is intended only for the indicated recipient..." Such implorations can become quite contorted.

The "Creative Community" – writers, artists, film-makers, and so on – have the formal concepts of copyright and licensing, which make explicit certain expectations of how their products will be used, and a large, if controversial, body of formal law, and even engineering mechanisms (such as DRM) attempt to hold others accountable, through these mechanisms, to the desires of the creator or owner (e.g., via assignment) of a work. A significant fraction of this Creative Community has adopted hybrid technical/legal approaches, such as *Creative Commons* (*CC*) [1], to enable creators/owners to specify how they intend their works to be utilized, but no such means of specification or enforcement exists for the sorts of things that we don't usually think of as "creative", i.e., copyrightable, works, such as interpersonal communications in email or SMS messaging, or the data that we send to others or provide to online services. Yet our email messages, at least, could easily be considered acts of creativity (some more so than others!), and other sorts of information, such as personal data, whereas not generally thought of as "creative" per se, has most of the same properties, specifically of the "creator" having certain desires regarding the disposition of the information, as exemplified above. But in these cases we are generally at the whim of the assumptions, or verbal pleadings, described above with regard to what actually happens to the information that we have provided, and must resort to trust (or prayer) in terms of either accountability or enforcement. If one does not say what one intends regarding the disposition of one's information, one must rely upon shared assumptions.

II. Designing RAPPD

The remainder of this paper describes RAPPD (Recipient Accountable Private Personal Data), a light weight prototype that enables one to communicate data through email, formally express one's desires with regard to the disposition of that data, and hold a trusted recipient accountable, to a limited extent, for carrying through on these desires.

A. *Use Case*

The typical use case that we envision for RAPPD is a patient sending personal medical information via email to an individually identifiable and trusted medical professional (e.g., their personal physician), or, more generally, an individual sending information via email to some trusted licensed professional. Other similar cases are those where we see the email disclaimers described above, such as a lawyer or physician sending information to one of his or her clients or patients. The key feature here is that both parties are individually identifiable, and one of them is some sort of professional so that there is an expectation of legitimate intent, at least on the part of the professional. Our central assertion is that even in settings where all parties are trustworthy, it is worth making explicit one's expectations regarding the disposition of transmitted private information, and providing technical means for verifying that these expectations are met, to the extent possible.

B. *Design Goals*

RAPPD conforms to a number of design goals:

1. Hybrid Dispositional Specifications (HDS): The originator's desires regarding the disposition of the information should be intelligible both to humans and to computers. The originator (presumed owner) of the information wants to be able to express his or her desires with regard to the disposition of his information in a way that is clear to the recipient. At the same time, we want the RAPPD system to be able to track the fulfillment of those desires.

RAPPD borrows from Creative Commons the idea of a "hybrid" human/machine-readable specification that the originators can associate with their email communication of the information in order to specify their intentions with regard to the disposition of the information. These specifications are "hybrid" in the sense that they are represented *both* in an easily understandable graphical format *and* in an internal formal format, visible only to the RAPPD system.

2. Initial Access Accountability (IAA): The originators want RAPPD to track who accesses their information and what these recipients do with it, at least upon first access by the intended recipient, and, if possible, subsequent accesses resulting from the primary recipient having (legitimately) forwarded the information to secondary recipients, and so on. If there is a leak, the originator should at least be able to trace it to the last accountable recipient.

3. Recipient Transparency (RT): The RAPPD process should operate within the recipient's normal communications workflow – in our case this is their email system. The recipients should not have to register or sign in beyond their typical email requirements, nor should legitimate secondary recipients have to do so. If we meet this goal, an honest recipient will have little motivation to bypass the RAPPD process.

4. Legitimate disposition accountability (LDA): Assuming that IAA and RT are met, recipients should be able to, and be motivated to use the RAPPD system to take action with respect to the information. We are typically concerned here with legitimate forwarding. RAPPD provides a tracked forwarding system that recipients may use if the originator has authorized forwarding of the information. Of course, a recipient could always work around RAPPD 's forwarding mechanism, for example by copy/pasting the data into another email message, but if RAPPD meets RT, recipients should have little reason to do so.

Unfortunately, without sign on, etc., RT places severe limits on what RAPPD can use to confirm IAA. Our prototype only records circumstantial data such as the accesser's IP address and access time. But given that the transmission is taking place within an email system, this is enough to confirm IAA, and someone who has received the data in violation must have gotten it from a legitimate recipient, so that if the originator did not give the primary recipient permission to pass the information on, the originator can at least contact the primary recipient to try to trace the leak.

C. *High Level Specification*

To meet these goals the RAPPD prototype has these components:

- A way of formally specifying the sender's intentions regarding the disposition of the content of an email communication.
- A graphical representation of those specifications, so that the recipient may be made aware of the sender's intentions.
- A way of encapsulating private data sent with the email.
- A callback mechanism that enables the sender to see when the recipient has obtained the private information.
- A way that an honest recipient may cleanly (if permitted by the sender) forward the information to a third party, retaining the sender's privacy intentions.

III. Background

A. *Control of Information after it is Transmitted*

The central question we face is: How can we reap the benefits of communicating private data and at the same time minimize the damage that might result from its leakage? A subset problem is: How can one control or at least track the disposition of private information after is has been transmitted? Existing answers have focused on prevention and recourse, with the former category based mostly technical means, while the latter is based on a combination of technical means and legal recourse.

One way to minimize the damage from information leakage is to not communicate it, or to distort it ([2], [3], [4]). In the present work we assume that the recipient requires the real data, as would almost certainly be the case for medical or financial transactions. Another way to minimize the damage from information leakage is to limit unauthorized use of the information. The government, entertainment industry, and more recently, social media have attempted to implement technical solutions to this end. The Bell-LaPadula model ([5]) takes into account that unauthorized parties could learn the information from authorized ones, and so implements rules that forbid authorized parties from writing any information at all where it could be viewed by unauthorized parties. Unfortunately, these rules are difficult to enforce in a distributed environment.

Enforcement of this sort of constraint is the role of the Digital Rights Management (DRM) process, which associates cryptographic keys with licenses [6]. But this requires installing software or hardware on the consumer's systems [7]. Moreover, DRM fails to take into account legitimate, albeit unforeseen, uses of information, such as fair use [6], or even more importantly, for example, making potentially life-saving private information available to paramedics in an emergency.

B. Accountability

Generally, taking action after detecting an actual violation is more feasible than frustrating potential ones. [6], [8], [9] note that enforcement of existing laws in everyday life occurs only after a violation, and argue that society could provide cyber-security by the same pattern. This kind of enforcement assumes that the victim can seek redress against the violator, which may not always be the case in transactions performed over the Internet. Nevertheless, we expect that most sanctioned transactions fall under this category. In our target use case the recipients, and therefore initial accessor of information will most often be licensed professionals or businesses which are much more easily held accountable than individuals.

Accountability policies, more loosely than their access control counterparts, classify actions as compliance or violations. [6], [8], [9], [10] all require a mechanism for setting such policies. Furthermore, [10] refers to the concept of "break-glass" scenario where parties are allowed to make unauthorized use of data due to an unforeseen emergency, albeit with some manual steps to avoid accidental violations, to show that the violations that occurred were deliberate, and most importantly, to make a traceable record of the intentional violation.

Such definitions of accountability are found in a number of legal and technical systems. In the U.S., the Fair Credit Reporting Act includes policies that require limited usage of financial data, and their accuracy, and mechanisms for consumers to know about adverse decisions based on their information, and to correct any inaccurate data [6]. HIPAA applies similar standards to health care providers [11], [12], [13]. For medical data, [14] proposes a framework that includes not only patients and health care professionals, but also a health authority that sets overall access policies that override the patients' choices in case of a conflict. For accountability, the framework allows violations of these policies, but records the transactions, and lets patients request justification for these actions.

However laws do not regulate the transaction of all private data with equal rigor, and in the absence of such general laws, formation of authoritative bodies that determine general policies and provide enforcement will be a contentious matter, and data-recipients can simply refuse to register for such accountability services.

In privacy, as well as copyright and other areas of information security, there has therefore been efforts to let the interested parties determine their own privacy policies, with various levels of legal backing, and enforce those policies, or at least audit adherence to them. On the policy side, the largest effort has been the *Platform for Privacy Preferences* (*P3P*) standard, which seeks to systematize how web sites publish their data usage policies to their visitors [15]. However, this standard has faced criticism due to its lack of enforcement –the policies only state how the website plans to use visitor data, and do not provide a mechanism for ensuring or determining that the promises are fulfilled –and its inconsistency with appropriate laws [16], a problem that is further exacerbated by the fact that the data collectors or recipients, rather than the originators, choose the rules.

In other areas of information security, the data originator can use standardized systems to choose policies. Websites that want to reduce interference from web crawlers that visit their pages can use a file named `robots.txt` to tell cooperative web crawlers how they should behave [17]. Creative Commons, mentioned in the introduction, allows authors of creative works to choose from a set of licenses to share their work while retaining some of their copyright rights [18]. Although the goal of Creative Commons is to promote openness, and the rights of the authors are already legally enforceable due to copyright laws [18], they provide a model for simplifying the task of choosing a data usage policy. However, as with P3P, by themselves, neither of these methods provide technical means of tracking violations.

On the enforcement side, there have been efforts to track violators in more adversarial situations when prevention fails. [19] is one of the earliest, best-known accounts of tracking unauthorized access to catch an attacker who has already bypassed the preventative measures. More recently, [20] has created a system for deploying beacons to track unauthorized accesses of files by users who have the technical privileges, but not the right to look at the data. [19], [20] both use fabricated, decoy files, and secretly track the activities of the intruders, which is not entirely appropriate in our case, where the recipient has the right to know the real data.

Nevertheless, tracking unpreventable violations is a useful technical solution. Some social media applications, such as Snapchat and the Facebook Poke App, let individuals control the distribution of their information by allowing the senders to limit the amount of time their messages will be viewable to the recipients [21], [22]. To our goal of IAA, some of these applications allow the originator to learn if the limitation was bypassed, for example by the recipient taking a screenshot of a message [21], [23]. The RAPPD approach is similar, but seeks to make the tracking more cooperative.

IV. The RAPPD System

RAPPD is a prototype implementation of a lightweight methodology which, in the trustworthy setting that we consider, can assist parties in appropriate disposition of the information. As described in the design goals above, it enables an individual to attach various policies to arbitrary kinds of data sent to different recipients, and to separately track the access of the messages containing the data through primary, and certain kinds of downstream transmissions.

A. Core Components

To support existing data formats and channels between the originator and the recipient, RAPPD wraps a privacy-enhancing layer around data that the originator wishes to keep private. This layer states the originator's desired privacy policy, and enables tracking accesses to the data, so that the originator, or an authorized auditor, can check that the recipient or recipients are following the policy. So in addition to the originator, recipient, and the data transfer and storage systems, RAPPD includes a tracking service that logs accesses and re-transmissions of the wrapped data.

We assume that the originator can transmit data by reference to an external data storage service. This allows him to transmit the same data to multiple recipients, for whom he may wish to specify different policies, and track separately. This external storage assumption also covers the common case when the originator does not own, or due to difficulties such as file size, cannot or does not wish to store the data over which he wants to assert his rights, as might be the case for medical data, which may often be stored in a hospital's EMR. Not only does the external storage assumption lessen the burden on the originator, recipient, and tracking service, but it also lets us use traditional security mechanisms to hide the data from our tracking mechanisms in the future. As we specifically assume that the data is remotely accessible via reference, such as a URL, our prototype provides such a service, permitting anonymous file uploads. Access to such a file requires a "unique secret," which consists of unique and random components. The unique component acts as an identifier that lets the storage service locate a requested file, and the random component acts as an authenticator, preventing attackers from efficiently guessing the secret required to access the page.

In the simplest case, the originator plays two roles: the sender and the auditor. The sender has all the credentials associated with the message, while the auditor only has the audit credentials. Thus the sender can also act as the auditor, but to allow the case that the role division is strict, our design provides that the auditor will not have all the sender's information regarding the data, in particular its contents and the identity of the intended recipient. As for the recipient, she only has the data access credentials.

The key part of the RAPPD system is the tracking service, which needs to be online when the sender registers the data, the recipient wants to view it, and the auditor wants to inspect the access log; therefore for practical purposes it needs to be persistently online. The tracking service contains records of all messages, which we call *transactions*, to authorize auditors and recipients who have provided their respective credentials, as well as the file accesses, or *views*. The tracking service needs to be able to uniquely identify transactions to which it associates the views, and to provide enough information for the recipient to view the sender's data, i.e., the reference of the file in the storage service. But the tracking service does not need to store the data itself.

The transaction object is a record of one message from a particular sender to a particular recipient, and therefore contains the corresponding privacy policies, so that there is a many-to-one relationship between the transactions and the posted files. The distinction between the transaction and data has the additional feature of keeping an auditor of one transaction from tracking the use of the same data by other recipients. To help a human auditor of multiple transactions distinguish between different recipients, the transaction includes a field for identifying the primary intended recipient. To minimize the privileges of an auditor who is not the sender, we allow the auditor to distinguish between different recipients without revealing email identities – the field is an opaque identifier. And while the transaction does not include the contents of the data to which it refers, it does include the sender-chosen subject line to label the file for easy organization.

To track accesses, each record contains two permissions: viewing and auditing. We keep them mutually exclusive, since not only does the sender want to prevent the recipient from becoming her own auditor, but the sender may also want to prevent the auditor from knowing the actual data. During the viewing step, the service needs to be able to retrieve the transaction, so that it can show the recipient the file. During the auditing step, the server needs to be able to find the same transaction, to show the associated views. Therefore, the recipient and auditor each need to know a unique identifier for the transaction. Although the recipient's identifier does not have to equal that of the auditor, there must be a one-to-one correspondence. The most simple identifier is the primary key given given by an SQL database entry. However, the primary keys are simply consecutive, and therefore highly predictable. We therefore use a similar scheme as the storage service, and require the primary key be accompanied by a randomly-generated secret for viewing the file, and the same key, accompanied by a different secret for inspecting the transaction's access log.

The view object is a record of a recipient-side access event of a transaction object's data. The relationship between views and transactions is many-to-one, and given a transaction, we can list all of its views. Aside from the primary recipient, which the auditor can infer from the transaction, the view itself also contains information about the access, which can vary between views, and in the case of a violation, possibly contradict the transaction's policy. The view model records at least the access time and the IP address of the viewer.

B. RAPPD Protocol in Detail

Use of the RAPPD system consists of three phases. The first phase, registration and sending, occurs once per transaction. The last two phases, viewing and audit, can occur as many times as the recipient and auditor want.

The sender contacts the tracking service during the registration and sending phase. The sender enters a reference to the data on a storage service. Besides the file pointer, the client program also asks the sender for metadata. Like in an email client, our program asks the sender for the recipient's email address and the subject. In addition, the user will submit a vector of privacy restrictions, from categories such as usage and transfer restrictions, and data retention time. The user can set the policy directly, or use a preset, recommended policy by specifying the type of message he is sending, and the type of organization that is receiving it. In addition, the sender has the option of entering a note to the recipient that the recipient can preview before deciding to open the private data, and expose herself to the tracker. When the sender registers his message, the client sends the recipient address, the subject line, and the reference of the file to the tracking service. The tracking service generates a new transaction, storing the shortened hash of the recipient email, and exact copies of the other submitted information. The service also assigns a unique key to the new transaction, and randomly generates the viewing and audit credentials. It stores cryptographic hashes of the credentials, and sends the unique key, and the original credentials to the sender. When the client receives the data, it combines the unique key and the audit credential, and displays it to the sender as the unique secret for auditing the access log of the transaction. Using the unique key and the viewing credential, as well as the privacy policies that the sender chose (but not the message and recipient types, if the sender used the recommendation system), the client generates the wrapper page for accessing the file, and displays it as a link that the sender can save and send.

The recipient contacts the tracking service during the receiving phase, which involves almost no manual steps. The key component of the message that the recipient opens is derived from URL shortening services. URL shortening services allow users to replace the URL to a desired web page using one pointing to the shortening service [24]. A side effect, which existing shortening services already exploit (albeit insecurely), is that they offer a unified tracking mechanism of accesses to the site [25]. This tracking approach requires explicit action by the recipient. It therefore contrasts with those of hidden *web bugs* (bug as in eavesdropping rather than programming error) [26], and *beacons* in [20]. Besides minimizing the impact on the privacy of the recipient, we make sure that the she is aware of the originator's expectations for privacy displayed with the link; only when the recipient clicks a link to the tracking service will the server receive the viewing credentials associated with the transaction, as well as the IP address of the recipient's host. This step addresses the catch-22 found in many confidentiality notices of having to read a message that you shouldn't perhaps be able to read in order to read the specification. Next, the tracker extracts the unique key and viewing credentials from the request, checks that a transaction with the unique key exists, and has a matching viewing credential. If the credentials match, the tracker redirects the recipient to the hidden location of the file, and generates a view with the IP address of the client, and the time measured at the server, and inserts it in the set of views of the transaction. After accessing the data, the recipient could save it and attach it to an email intended for a secondary recipient. However, for a user who does not intend to maliciously bypass the system, it is easier to simply forward the sender's email, assuming that the sender has specified that this is a permitted action.

The auditor contacts the tracking service during the audit phase. The auditor enters the unique ID and the audit key from the registration phase into a form served by the tracking service. As with the receiving phase, the tracking service authenticates the client, but checks the audit credential rather than the viewing credential. If the credentials match, the service displays a table of all the views, showing the access times, the viewer IP addresses, and a short hash of the intended recipient. Although both the viewing and auditing credentials easily identify the same transaction, as they contain the same unique ID, we stress that the secret components are independent.

C. Implementation Details

We implemented the tracking service using the Django [27] framework. The RAPPD backend is implemented as a REST-like web server, while the user interfaces are written as web pages and emails with embedded HTML. However, JavaScript's same-domain policy prevents clients from directly communicating with the server [28]. We use HTTP GET requests, with parameters and credentials in the URLs, to bypass this restriction. Because access takes place in a browser, the view object can include browser and operating system information in the HTTP request header, which may be useful for digital forensics. However, an attacker with basic network programming skills can spoof this information rather easily. Not so for access time, which is determined at the server, or IP address, since the receiver still needs the server to send back a TCP handshake [29] as well as the desired data, and so require more effort to fake.

D. Additional Features

In addition to the components that RAPPD requires, our implementation includes some features to more closely track different recipients, and which leverage existing infrastructure to make the user experience more convenient.

1) Promoting Accountability: The transaction object only contains information about the primary recipient, and the view object only contains information that the viewer's browser automatically sends. However, the IP address in the view object may not be enough. For example, in the case when various employees behind a NAT box are resending the data, it appears that the the same individual is making the accesses. Furthermore, in case of a breach the originator may not only want to hold the actual violator accountable, but also take actions against whoever gave her the data, which will be complicated in the case of multiple downstream recipients. At the same time, when the primary recipient is a sender, she may have an interest in tracing downstream activity as well.

We increase the detail in the access log by extending our definition of transactions to all messages, and not only the message between the primary sender and the primary recipient (called the primary transaction, v. secondary or downstream transactions). The primary transactions resemble our original definition of transactions. Secondary transactions, however, do not store the link to the data. Instead, they store references to the original, primary transaction that does have this link, and to the parent transaction, which the intermediate recipient used to forward the data. Note that recursively tracing the parents of a series of such transactions leads to the original one, and reveals information about the intermediate recipients who created each downstream transaction. Most of the latter information is available in the parent transaction. One exception is the IP address with which the intermediate recipient accessed the parent transaction to create its child. In this respect, a secondary transaction resembles a view, except that our extension to the transaction model lets a secondary transaction have children. So we both record a retransmission as a special view, stating that the message was "viewed for retransmission" in the access log, and store secondary transactions with the intermediate recipient's IP address and email to trace the retransmission of data.

When a secondary recipient wants to access the data, given the viewing credentials of a secondary transaction, the tracking service will fetch the original transaction to which the secondary transaction refers, return its URL, and create a view that will be visible to the auditor of the the secondary transaction, as well as those of all of the ancestor transactions. We chose to group views by the immediate transaction through which the data was accessed, rather than the original, as the audit log of any intermediate transaction can then, by any tree-search algorithm, enumerate the transaction's own views, and recursively find the views of all of its descendants. The reference to the immediate transaction also lets the access log recursively trace the ancestors to display the trace of the retransmissions between the view and the original.

We also need to modify the client-side interfaces to support retransmission. Unlike the original sender's page, a recipient's forwarding page only asks for the next recipient's email address and an optional note. In addition, as a "break glass" step, a warning (which the recipient needs to acknowledge and click through) will block the forwarding page if the privacy conditions of the message do not allow retransmission. After the recipient submits his input, the page will contact the tracking service to register a new transaction, and generate a copy of itself, with the viewing credentials of this new transaction. Fortunately, JavaScript lets us copy and store the entire original document as a string, after the browser loads it. When the intermediate recipient chooses to resend the message, the script finds the values in the string that we want to change, and replaces them with the new credentials returned from the tracking service.

2) Promoting Convenience: Aside from privacy, RAPPD can make certain communication between the originator and recipient more convenient. Privacy requirements may complicate communications when the originator wishes to remain anonymous from secondary recipients. For example, the originator submits medical data, and requests that derivative data anonymize his identity or other personal information. A research institution that receives the data may want to contact him for further trials, but would not know how to reach him. However, the RAPPD audit log can permit the originator to receive feedback from downstream recipients without revealing the originator to these downstream agents. The main differences between a view object and a reply to the originator are direct control of the replier over the contents of the reply, and the fact that the replier may not have the right to view the data as the originator sent it. Therefore, we add a weaker permission to a transaction. A potential replier receives the credentials (in the usual key-secret form, albeit with a shorter secret) embedded in a URL. This URL lets the replier open a form served by the tracking service, and compose and submit a message. The tracking service thence stores the message as a *Reply* object, which will be enumerable given the original transaction, so that the originator can read it when viewing the audit page. At no point does the downstream agent need to know the identity of the originator in this transaction.

An additional convenience enhancement is to integrate email usage with the RAPPD sender interface, so that it delivers messages on behalf of the originator without his having to switch to another program: with the addition of an optional `From` address field, originators can send themselves their audit credentials, and, with the recipient's address, the sender (or intermediate recipient) can send a recipient the wrapper page of the data. The latter enhancement is especially more straightforward to use. The basic implementation only displays a link to a data URL of the webpage. The sender can send it three ways: he can copy the link location, i.e. the URL, and paste it into the body of his email to the recipient; he can use the "Send Link" option to open an email client, and compose a message containing the URL; he could save the link location as an HTML page, and attach it to the email. The first two solutions may be inconvenient for the recipient, who will see a string that not only does not resemble a usual HTTP or HTTPS URL, but is also extremely long. Furthermore, all three approaches assume that the sender is familiar with the process of right-clicking a link to copy, send or save its URL.

Unfortunately, the feasible implementations of this enhancement all pose security risks. A simple `mailto` link that lets the sender transmit mail in a convenient HTML format containing his privacy icons, using his own mail client or mail service will generally not work, since the IETF standard for the `mailto` URL does not require support for most fields, including the attachment and data type fields, and in fact considers them unsafe [30]. We therefore need to implement a separate email service, either using the sender's actual email address, or an anonymous address. It would be inconvenient and unsafe for the email service to use the sender's actual email account, as that would require the sender to input the SMTP server of his actual email provider, as well as his login credentials. On the other hand, spammers can easily abuse an anonymous email service. While we chose the latter approach, if we wish to do so in the final implementation, we would have to use CAPTCHAs and other anti-bot techniques [31], as is commonly done with most free email services, to slow down automated senders.

The final convenience feature we added is, as previously mentioned, our own file storage system that lets the user transmit or share the contents of a file without already having the URL to an uploaded file. It accepts uploads via HTTP GET requests. The sender can select a local file, or enter text. The client-side program first uploads the data to our storage service, and receives the URL, which the client stores in the same JavaScript variable that would have stored a manually-entered URL to be sent to the tracker.

V. Future Work

A. Enhancements to RAPPD

We can improve RAPPD with greater integration and flexibility. If we can let the sender's program automatically communicate with all the popular file storage and mail services, then we can eliminate our storage and mail services, and let the sender automatically use the services he normally uses outside of RAPPD. We can also eliminate the need for the sender to find the URL and then open the sending form, if he could simply register and send the data he is viewing on his browser. And if the browser opened a local file, i.e., the URL starts with the `file://` prefix, the sender's client program can again upload it automatically. Alternatively, we could let the sender stay on his user email client, and register his email after he composed it. This arrangement also achieves the goal of integrating the sender's interface with a mail service. However, all the work on integration would require either code outside of the webpage to avoid the same-origin policies, at least in the form of a browser add-on, or cooperation from various email and storage service providers.

To let originators maintain control of their privacy in more diverse situations, we want RAPPD to support and track more legitimate ways for recipients to use the data. In research, the recipient may combine the data of several senders before retransmitting them. The forwarding interface could support the insertion of references, so that when the combined data is accessed, all senders will see the access on their logs. Moreover, the forwarding form could also use the references to the component transactions to make sure that the combination's privacy terms are at least as restrictive as its components'. We also want to give the sender flexibility over the terms that he chooses. We can increase the number of terms by two ways. We can review relevant privacy laws, and create both new direct privacy options and recommendations that can be enforced by those laws, as is the case of the Creative Commons licenses [18]. This approach has the benefit of not only supporting enforceable privacy options, but also educating users about their privacy rights in different cases. However, this approach has the disadvantage of limiting the new options we can add, and creating the need to keep track of the laws in different jurisdictions. The second approach is to let users choose and publish their own policies. While this allows users to set terms exactly to their own choosing, the proliferation of privacy options creates the challenges of keeping their meanings clear, the options relevant, and the interface simple to use.

B. RAPPD with other Solutions

We can can expand RAPPD with other solutions to improve both preventive security where it is truly necessary, as well as accountability.

Currently, the three services are storing information that may make a user wary. The mail relay, or even the sender's mail service, have all of the credentials of the transaction. It can therefore access the data and audit log, although in the former case, such actions will be detected; but the tracking service can access the data undetected, if the storage service is not doing its own tracking; the storage service has direct access to the data. In addition to promoting convenience, if integration between the storage service and the recipient's interface lets the recipient's program recombine data divided between itself and arbitrary storage services, then we can use cryptography, storing the key and ciphertext on the storage service and email, or the other way around, ensuring that no single server has unaccountable access to the complete data.

Because RAPPD provides a remote data access tracking mechanism, it can make use of some of the accountability methods intended for single file systems and data bases. To fully reap the benefits of the following enhancements, the audit log will have a separate, more visible section that lists recipient actions deemed more suspicious than the rest. One set of actions that should fall under the special section is those that contradict the privacy policies (some of which we will have to create). As in [14], we warn the recipient if she is violating a policy, and require a manual confirmation before flagging the action. The tracking service itself can also flag certain activities it deems suspicious, either by a predetermined set of rules, or by anomaly detection [32].

VI. Conclusion

As the saying goes: *Doveryai, no proveryai.* (Trust, but verify). We have argued that even in settings where all parties are trustworthy, it is worth making explicit one's expectations regarding the disposition of transmitted private information, and providing technical means for verifying that these expectations are met, to the extent possible. Furthermore, we have argued that it is valuable to to provide technical means for acting upon and passing on these expectations in downstream communications, or at least "break glass" technologies where one wants to explicitly escape from them. RAPPD is a prototype that demonstrates how these goals can be accomplished in browser-based email, a very common locus of interpersonal information transmission. There is clearly much yet to be accomplished in this important setting.

Acknowledgment

This work was supported by CommerceNet.

References

[1] C. C. Corp., "About creative commons," http://creativecommons.org/about/, accessed: 2014-1-22.

[2] I. Dinur and K. Nissim, "Revealing information while preserving privacy," in *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2003, pp. 202–210.

[3] R. Chow and P. Golle, "Faking contextual data for fun, profit, and privacy," in *Proceedings of the 8th ACM workshop on Privacy in the electronic society*. ACM, 2009, pp. 105–108.

[4] A. Baquero, A. M. Schiffman, and J. Shrager, "Blend me in: Privacy-preserving input generalization for personalized online services," in *PST*, 2013, pp. 51–60.

[5] D. E. Bell and L. J. L. Padula, "Secure computer system: Unified exposition and multics interpretation," http://csrc.nist.gov/publications/history/bell76.pdf, DTIC Document, Tech. Rep., 1976.

[6] D. J. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. Hendler, and G. J. Sussman, "Information accountability," *Communications of the ACM*, vol. 51, no. 6, pp. 82–87, 2008.

[7] M. Russinovich, “Sony, rootkits and digital rights management gone too far,” http://blogs.technet.com/b/markrussinovich/archive/2005/10/31/sony-rootkits-and-digital-rights-management-gone-too-far.aspx, October 2005, accessed: 2013-7-10.

[8] B. Lampson, “Privacy and security usable security: how to get it,” *Communications of the ACM*, vol. 52, no. 11, pp. 25–27, 2009.

[9] L. Kagal and H. Abelson, “Access control is an inadequate framework for privacy protection,” in *In W3C Privacy Workshop*, 2010.

[10] J. Feigenbaum, J. E. Hendler, A. D. Jaggard, D. J. Weitzner, and R. N. Wright, “Accountability and deterrence in online life,” in *Proceedings of the 3rd International Conference on Web Science, ACM*, 2011.

[11] “Code of federal regulations 164.508(a).”

[12] “Code of federal regulations 164.528(a).”

[13] “Code of federal regulations 164.524(a).”

[14] R. Gajanayake, R. Iannella, and T. R. Sahama, “An information accountability framework for shared ehealth policies,” 2012.

[15] W. W. W. Consortium, “The platform for privacy preferences 1.1 (p3p1.1) specification,” http://www.w3.org/TR/P3P11/, November 2006, accessed: 2013-8-27.

[16] E. P. I. Center, “Pretty poor privacy: An assessment of p3p and internet privacy,” http://epic.org/reports/prettypoorprivacy.html, June 2000, accessed: 2013-8-27.

[17] “The web robots pages,” http://www.robotstxt.org/robotstxt.html, accessed: 2013-8-27.

[18] C. C. Corp., “About the licenses,” http://creativecommons.org/licenses/, accessed: 2013-8-27.

[19] C. Stoll, “Stalking the wily hacker,” *Communications of the ACM*, vol. 31, no. 5, pp. 484–497, 1988.

[20] S. J. Stolfo, “Methods, systems, and media for measuring computer security,” June 2011, uS Patent App. 13/166,723.

[21] Snapchat, “Snapchat - real-time picture chatting for ios and android,” http://www.snapchat.com/#, 2013, accessed: 2013-7-10.

[22] Facebook, “Facebook poke — facebook help center — facebook,” https://www.facebook.com/help/397568030328686/, 2013, accessed: 2013-7-11.

[23] ——, “Someone took a screenshot of my poke and i don’t want them to share it. what can i do? — facebook help center — facebook,” https://www.facebook.com/help/461596740570393, 2013, accessed: 2013-7-11.

[24] B. Inc., “What is a shortlink?” http://support.bitly.com/knowledgebase/articles/102427-what-is-a-shortlink-, accessed: 2013-8-22.

[25] S. Chapman, “How to spy on campaigns of competitors who use url shorteners,” http://www.zdnet.com/how-to-spy-on-campaigns-of-competitors-who-use-url-shorteners-7000001088/, August 2012, accessed: 2013-8-12.

[26] R. M. Smith, “The web bug faq,” http://w2.eff.org/Privacy/Marketing/web_bug.html, accessed: 2013-7-11.

[27] D. S. Foundation, “Django at a glance,” https://docs.djangoproject.com/en/1.6/intro/overview/, accessed: 2014-1-28.

[28] M. Corp., “Same-origin policy,” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Same_origin_policy_for_JavaScript, 2011, accessed: 2013-8-27.

[29] M. Tanase, “Ip spoofing: An introduction,” http://www.symantec.com/connect/articles/ip-spoofing-introduction, accessed: 2013-7-22.

[30] M. Duerst, L. Masinter, and J. Zawinski, “The ’mailto’ uri scheme,” https://tools.ietf.org/html/rfc6068, October 2010, accessed: 2013-7-23.

[31] S. Ostermiller, “Contact form and spam,” http://ostermiller.org/contactform/spam.html, 2011, accessed: 2013-8-27.

[32] D. E. Denning, “An intrusion-detection model,” *Software Engineering, IEEE Transactions on*, no. 2, pp. 222–232, 1987.