Konfidi:* Trust Networks Using PGP and RDF

Dave Brondsema
Calvin College
3201 Burton St., SE
Grand Rapids, MI 49546
dave@brondsema.net

Andrew Schamp
Calvin College
3201 Burton St., SE
Grand Rapids, MI 49546
schamp@gmail.com

ABSTRACT
We ought to write this now!

Keywords
Semantic Web, trust network, FOAF, RDF, PGP, GPG, reputation, propogation, distributed, inference, delegation, social network

1. INTRODUCTION
As internet-based communication grows, there has been rapid growth of unscrupulous users taking advantage of the system to send spam and propagating viruses to unsuspecting users. This gives rise to two questions: How can I be sure that a message really comes from who it says it does? How can I be sure that what the sender has to say is accurate?

There have been a number of attempts to answer either one question or the other. The Pretty Good Privacy (PGP) encryption system has developed a web-of-trust which can help provide verification of an individual's identity; however, it does not allow the expression of any additional information about that individual's trustworthiness on matters other than personal identification. As for the second question, one answer that is growing in popularity is that of creating a network of trust between individuals who know one another and have good reason to trust their estimations of others. However, these systems can subject to problems; suppose someone impersonating a trusted party provides incorrect data boosting the reputation of an untrustworthy party. Ratings system for reputation within certain domains[1] may be of some limited use. However, unless some there is a system to verify the raters, they may also be susceptible to malicious users who manipulate ratings. Even if such systems can be guarded against such attacks, why should someone base whether they trust another person on opinions given by others that they neither know nor trust?

In this paper, we present a system that combines the a trust network with the PGP web-of-trust. We describe some difficulties in integrating such reputation-based trust networks with the PGP web-of-trust, and how it might be possible to overcome them. We then describe our structure for representing trust data, and our methods for making trust inferences on this data. Finally, we discuss the our proof-of-concept software for putting this trust to use.

2. RELATED WORK
Rather than creating an entirely new system from the ground up, we have leveraged a number of existing technologies designed to serve various purposes, perhaps in ways compatible with but unanticipated by the original creators. We introduce them here, and explain later in the paper how we have integrated them. We also discuss here related academic research on the relevant topics.

2.1 Representing Trust Relationships
There seems to be a general lack of psychological research on ways of representing trust relationships between individuals and procedures for inferring unspecified trust values. We found no recommendations for a particular scheme for modeling trust relationships or networks mathematically. Most work on this topic in the fields of mathematics and computer science adopts an arbitrary model appropriate to the algorithm under consideration. Guha points out [**?**] that there are compelling reasons for a trust representation scheme to express explicit distrust as well as trust.

2.2 Trust Networks and Inferences
There are a number of papers about different propagation strategies for weighted, directed graphs[2]. For the most part, however, they are concerned with describing the networks mathematically, and do not have much in the way of practical application. While they are of interest and relevance, they concern only the subsystem and do not discuss the design of a larger infrastructure.

Jennifer Golbeck, at the University of Maryland, is doing work on trust and reputation systems [**?**] that is similar to our work on this project. Like us, she uses an extension

Konfidi is the Esperanto term for trust. A universal concept in a universal language seemed appropriate for what we hope will become a universal system.

[1] such as eBay online auctions

[2] See such-and-such, and so-and-so, for example

of FOAF to represent trust relationships and a rating system[3]. She has created TrustMail [?], a modified email client that uses her trust network. She is more concerned with an academic approach than a pragmatic one, since this field is still growing rapidly and she prefers to help research other applications and implications of semantic social networks.

Golbeck suggests an important distinction between belief in statements and trust in people [?]. While networks of both kinds can be created, the latter are usually smaller and more connected. Golbeck argues that in a combined network of trust in people and of belief in statements, a path composed of trust edge and terminating with a belief edge is equivalent to one composed entirely of belief edges, and is on average smaller. Thus, a trust network comprising mostly trust edges allows for simpler traversal.

2.3 The Semantic Web

In addition to Golbeck's work, a number of others have explored the usefulness and implications of expressing trust relationships in the Semantic Web.

The FOAF project [?] is a Resource Description Framework (RDF) vocabulary used to represent personal data and interpersonal relationships for the Semantic Web. Users create RDF files describing Person objects which can specify name, email address, and so on, but more importantly, they can express relationships between Person objects. There are a number of tools in development for processing FOAF data and traversing references between FOAF RDF files. These tools can aggregate information because RDF often uses uniform resource indicators (URIs) to describe an individual object.

Dan Brickley has made a practical attempt to investigate the use of FOAF, particularly the mbox_sha1 property, to automatically generate email whitelists. By hashing the sender's address using SHA1, privacy is protected (and the address cannot be gathered by spiders), and so users can share whitelists of non-spam emailers. Then for all incoming mail, the sender's address is hashed and the whitelist searched for the resulting value, and then is filtered accordingly. This use of FOAF is promising, but since it is decentralized, it is difficult for updates to propagate [?]. No effort was taken in this project to verify the sender's identity.

2.4 Email Filtering

Filtering email to reduce unsolicited email has received lots of attention in many areas. Domain-level solutions, such as SPF[?] and DomainKeys[?], are designed mostly to prevent phishing and also assume that a domain's administrator can control and monitor all its user's activities. Greylisting and blacklisting often have too many false positives and false negatives. User-level filtering, which Konfidi does in the context of email filtering, is not very common. Challenge-response mechanisms to build a whitelist are tedious for the sender and reciever and do not validate authenticity. Content-level testing is the most common, but bayesian filtering and other header checks are reactionary and must be continuously updated, and are becoming less effective as spammers create emails that look more and more real.

There has been some work to bring authentication to email through efforts like DomainKeys, SPF, and IIM[?]. It seems that their primary goal is to prevent phishing[4], not to stop spam. They store cryptographic data in DNS records and the granularity is normally per-domain not per-address. These approaches limit their applicability to domain-related data (email, webpages) and do not address any issues of trust (DNS records must be assumed to be authentic).

2.4.1 *Inferred Trust*

Boykin and Roychowdhury discuss ways to infer a relationship based on existing data [?]. They suggest scanning the `From:`, `To:` and `Cc:` headers and building a whitelisting database based on relationships indicated by the recipients. This seems to work fairly well, but there is often not enough data to make the spam/not-spam decision because it is based only on the user's own previously received messages. They clearly state a cryptographic solution would be ideal to verify identity.

2.5 Trust Inference Using PGP

There are some Mail User Agent (MUA) plugins, such as Enigmail[?] use PGP to sign emails and validate any emails that are recieved with a PGP signature. This is done entirely in the MUA, fetching keys from the keyserver when necessary. If there is a short enough path of signatures[5] from the recipient to the sender, the signature is considered "trusted". It does not fetch keys in an attempt to find such a path; you must already have the keys locally that form the path.

This approach requires that most users digitally sign email messages, and it depends on users to be aware of known spammers and avoid signing their keys. However, the recommended PGP keysigning practices require only the careful verification of the key-holder's identity, and a signed key does not entail anything about trustworthiness in other areas. Furthermore, if the identification requirements for keysigning are met, even by a spammer, it would be unfair to refrain from signing that spammer's key[6]. Whether a user should be trusted to send good email, and not spam, is information over and above that expressed in the PGP web-of-trust itself, so the web-of-trust would be inadequate to encode such information.

A more serious flaw in this approach is this: because information about key signatures is stored with the key that was signed, and not that of the signer, paths between users can only be constructed from the sender backward to the recipient. When a key is retrieved from a keyserver, all of the signatures on that key are included with it, showing which keys have signed that key, and providing a number of possible links in a chain. However, using the existing

[3]Though both our ontologies and ratings are different in significant ways, which we will address later.

[4]Emails with a forged From: address

[5]In the web-of-trust, nodes are PGP keys and edges are key signatures. Paths are made when the recipient had signed someone's key, who had signed another key, and so on until a signature is found on the sender's key

[6]In fact, such positive identification might be of use.

keyserver infrastructure without a transformation of some kind, there is no easy way to tell which other keys a particular key has signed. If these paths are built backward using a breadth-first search from the sender to the recipient, a spammer or other malicious user could generate a large number of fake keys that are inter-signed, and then use these keys to sign the sender's key. By adding this artificial information, the client's searching capabilities would be crippled, and the web-of-trust would be filled with fabricated keys, users, and signatures, quickly becoming useless. The PGP system of key-signing and verification was designed to be robust against this sort of impersonation, by requiring photo-identification and fingerprint exchange before any key-signing, but a deluge of false information would put undue strain on the keyserver infrastructure, and would amount to a denial-of-service, of sorts.

2.5.1 Wotsap

Wotsap[**?**] is a tool to work with the PGP web-of-trust. From a keyserver it creates a data file with all the names and signature connections, but no cryptographic data. A python script is available to use this data file to find paths between keys, generate graphics, and generate statistics.

2.6 Summary

These related works form many of the building blocks, both technically and theoretical, for our work. The system should determine authenticity through a decentralized network and determine trust in a topic through a similar network topology. We leverage PGP, RDF and FOAF, and design ideas from Golbeck, Guha, and others.

3. KONFIDI

Konfidi refers to the trust network design, the ontology used to encode it, and the software to make it usable. Figure 1 shows the components of the Konfidi architecture and how they relate to external components and one another. The numbered paths indicate the steps in the process. First, a client makes a request to the Konfidi server, indicating the source and sink, usually the person making the request and the person to which the path leads, respectively.

Then, the frontend passes the request to the PGP Pathfinder, which verifies that some path exists from the source to the sink in the PGP web-of-trust. After this, the frontend passes the request to the TrustServer, which checks the Konfidi trust network that is built from data kept up-to-date by the FOAFServer. The TrustServer traverses the trust network and constructs a response with the inferred trust value or an appropriate error message. The Frontend combines the responses of the Pathfinder and the TrustServer, and sends it along to the client. In the remainder of this section, we discuss the underlying data structure for representing trust, how it is implemented in these steps, and the rationale for the system design.

3.1 Trust Ontology

In the current research on trust inference networks, there seem to be two general kinds of representations: one that uses discrete values for varying levels of trust, and one which uses a continuous range of trust values. Both return an answer in the same range as their domain. Now, either kind of representation could be roughly mapped onto the other, however, a continuous range would allow more finely-grained control over the data. Further, the inferred trust values returned by searches would not have to be rounded to a discrete level, which would lose precision.

3.1.1 Distrust

The choice of representation is closely related to another important concern, that the representation give some account of distrust. If the trust network contained values ranging from neutral trust to complete trust, then everyone in the network is trusted, explicitly, or by inference on some level at or above neutral. If the system makes a trust inference between Alice and Bob at one level, but Alice really trusts Bob at a different level, she can explicitly state this previously implicit trust to have a more accurate result (for herself and for others who build inference paths of whom she is a member). But, suppose that Alice feels strong negative feelings about Bob. In this case, she would still only be able to represent this relationship as one of neutral trust. So, the trust network must account for distrust in some reasonable way.

One of the difficulties of using explicit distrust in an inference network, however, is that it is unclear how inferences should proceed once a link of distrust has been encountered. Suppose Alice distrusts Bob, and Bob distrusts Clara. As Guha points out [**?**], there are at least two interpretations of this situation. On the one hand, Alice might think something like "the enemy of my enemy is my friend" and so decide to put trust in Clara. On the other hand, she might realize that if someone as scheming as Bob distrusts Clara, then Clara must really be an unreliable character, and so decide to distrust Clara. Further, suppose Bob expressed trust for Dave. At first consideration, it might seem reasonable to simply distrust everyone that Bob distrusts, including Dave. But suppose there were another path through different nodes indicating some minimal level of trust for Dave. Which path should be chosen as one which provides the correct inference?

One solution to this problem is simply to not traverse beyond a relationship of distrust, and to seek an alternate path. There is no clear answer regarding the statements a distrusted person makes, so it is best not to consider them at all.

We decided on a model that corresponds to another intuition about how trust works between people, that it is more of a continuum of both trust and distrust than a measure of just one or the other. For example, if Alice trusts Bob at some moderate level (say, .75 of a scale of 0 to 1), then it seems that she also *distrusts* him at some minimal level (say, .25). If Alice trusts Bob neutrally, then she trusts him about as much as she distrusts him. If she distrusts him completely, then she doesn't trust him at all. But in all of these cases, there is a trade-off between trust and distrust. Only in the extremes is either of them eliminated completely. So, we decided that our trust model should represent a range of values from 0 to 1, treating 0 as complete distrust, 1 as complete trust, and 0.5 as neutral, and calculate trust

Figure 1: The Konfidi Architecture

inferences accordingly[7].

3.1.2 Data Structure

With these considerations in mind, we found that a relationship-based model would meet our needs while avoiding some of the disadvantages of other representations. According to this system, each trust relationship is an object, and the trusting person and the trusted entity (typically a person) are specified as such. Thus, each relationship is one-way, but since the truster is responsible for the accuracy of the information, that is fine. Trust relationships also have trust items specified.

Because the trust relationship is represented as its own object, other attributes may be added later, such as the dates the relationship began, annotations, etc. as the need arises.

3.1.3 Trust Topics

If other attributes about a trust relationship could be expressed, in addition to the rating system, then a system like Konfidi would be useful in many wider scopes than email spam prevention. The most important of these is the trust topic, or in other words, what the trust is about. A natural feature of interpersonal trust relationships is that there can be many different aspects of the same trust relationship.

(you can make an Alice-Bob-Clara-Dave diagram for this business)

For example, suppose Bob is a master chef, but is terribly gullible about the weather forecast. Alice, of course, knows this, and so wants to express that she trusts Bob very highly when he gives advice for making souffle, but she does not trust him at all when he volunteers information about the likelihood of the next tornado. Suppose she only knows Bob in these two capacities. Any trust inference system should not average the two trust values and get a somewhat neutral rating for Bob, for that would lose important information about each of those two trust ratings, the only information that made these ratings useful in the first place.

Suppose also that, given only the above trust ratings, the system tried to make an inference on a subject that was not specified. Perhaps Alice has some general level of trust for Bob that should be used when there is no specific rating for the topic in question. See the discussion in Future Work for our proposal for a hierarchical system of topics that might account for this situation.

3.1.4 OWL Schema

As the FOAF project grows in popularity, an infrastructure is growing to support it, as mentioned in 2.3. Since there would be many advantages to tapping into this infrastructure, and since the specification of trust relationships fits in naturally alongside the existing `foaf:knows` property, Konfidi also uses the Resource Description Framework (RDF) [**?**] for representing trust relationships. In addition to the FOAF vocabulary, there is a vocabulary called WOT defining and describing signing and assurance, which provides the necessary structure for associating persons and organizations with key fingerprints [**?**]. By designing Konfidi's vocabulary to make use of FOAF and WOT vocabulary elements, then, we can take advantage of the established standards and make our extensions compatible with existing FOAF-enabled tools.

Konfidi uses the Web Ontology Language (OWL) [**?**] to define the RDF elements that make up the Konfidi trust ontology. OWL builds on the existing RDF specification by providing a vocabulary to describe properties and classes, and their relations. The Konfidi trust ontology provides two objects and five properties, which, in conjunction with the existing FOAF and WOT vocabularies, are sufficient to describe the trust relationships that Konfidi requires.

[7]This also makes many propagation algorithms simpler, as we'll discuss later.

The primary element is `Relationship`[8], which represents a relationship of trust that holds between two persons. There are two properties required for every `Relationship`, `truster` and `trusted`, which indicate the two parties to the relationship. Both `truster` and `trusted` have `foaf:Person` objects as their targets. These `Person` objects should also contain at least one `wot:fingerprint` property specifying the PGP fingerprint of a public key held by the individual the `Person` describes. This property is required for verification; if no `fingerprint` is available, then Konfidi cannot use the relationship. In general, any object described in RDF with a resource URI can be the `trusted` party, such as specific documents or websites, but for simplicity in our examples, we will focus on persons. which may be defined in the same file, inline, or in external documents indicated by their resource URIs. Because it does not matter where the `foaf:Person` data is stored, users may keep files indicating trust relationships separate from main FOAF files. However, to prevent spoofing, any file containing one or more `Relationship` objects must have a valid signature from a public key `fingerprint` corresponding to each `Person` listed as a `truster` in that file. As described in 4, flexibility in data location can have a number of advantages.

In addition to `truster` and `trusted`, each `Relationship` requires at least one `about` property, which relates the trust `Relationship` to a trust `Item`. A `Relationship` is not limited in the other properties it can have, so auxiliary information about the relationship, such as when it began, who introduced it and so on may be recorded without having an effect on the requirements of Konfidi. Each `Item` has two properties belonging to it. The `topic` property specifies the subject of the trust according to a trust topic hierarchy[9] and the `rating` property indicates the value, according to the 0-1 scale of trust (specified in Section 3.1.2) that is assigned to the relationship on that topic.

A `Relationship` may have more than one `Item` that it is about. For example, remember the example given above, in which Alice trusts Bob highly about cooking, and distrusts him somewhat about the weather. This might be represented in our ontology as something like the following[10]. For a more in-depth example, see Appendix B.:

```
<Relationship>
  <truster rdf:resource="#alice123" />
  <trusted rdf:resource="#bob1812" />
  <about>
    <Item>
      <rating>.95</rating>
      <topic rdf:resource="#cooking" />
    </Item>
  </about>
  <about>
    <Item>
      <rating>.35</rating>
      <topic rdf:resource="#weather" />
    </Item>
  </about>
</Relationship>
```

See Appendix A for the full OWL source code.

3.2 TrustServer

This part of the design is the core element of Konfidi. Storing and managing the data would not be of much use in this context unless we did something with it. So, the TrustServer would handle requests for trust ratings, verify that a PGP connection exists, and traverse the internal representation to find a path. Since these three tasks are so distinct, all of Konfidi is divided into three parts, a frontend which listens for requests and dispatches them, and two backend components, one to search the PGP web-of-trust and another to query against Konfidi's own trust network. This separation, in addition to simpilifying the design by encapsulating the different functions, also allows for increased flexibility and scalability. Each part is loosely coupled to the other parts, with a simple API for handling communications between them.

3.2.1 *Frontend*

Like the FOAFServer, the TrustServer's frontend is a web service, using the REST architecture to receiving and answering queries. It also runs on the Apache web server, using the mod_python framework. Queries are passed in using HTTP's GET method, and responses are returned in XML, which a client application may parse to retrieve the desired data.

When a query is received, the Frontend passes the source and sink fingerprints to the PGP backend, and, if a valid path is found, to the trust backend[11]. The Frontend then builds the response document in XML, and passes the result back to the client[12].

3.2.2 *PGP Pathfinder*

As mentioned in Section 2.5, the PGP web-of-trust is not sufficient for determining trust. However, it is necessary for the proper operation of Konfidi because it plays a very important role in verifying the identity of the sink. If the author of a document were not identified correctly, someone might forge the trust data, and Konfidi would return a deceptive result. Verifying that the document's signing key matches the key of the sink in the Konfidi trust network ensures that when Konfidi finds a topical trust inference path from source to the sink, it is valid.

Since one purpose of Konfidi is to provide trust information over and above the PGP web-of-trust, it may seem natural to keep the Konfidi trust network tightly coupled to it. In other words, every trust link specified using the Konfidi

[8] According to RDF standards, the names of objects are capitalized, while the names of properties remain lowercase.

[9] yet to be developed

[10] That is, supposing that the objects `alice123`, `bob1812`, `cooking`, and `weather` are all defined elsewhere in the same file.

[11] Strictly speaking, either query is optional. The PGP backend may be skipped to run tests on large sets of sample data, and the trust backend may be skipped if the system is to be used as an interface to the PGP web-of-trust only.

[12] The client may also request for simplicity a single value, the trust rating value.

trust ontology corresponds to a link in the PGP web-of-trust made by signing the appropriate key. However, there are compelling reasons not to do this. First, the set of people one might wish to indicate trust for in Konfidi will likely not be the same as the set of those whose keys you are able to sign. For example, a researcher in Sydney may work closely with another in Oslo, and so trust that person's opinion highly in matters relating to their research. But unless they are able to meet at a conference some time, or are willing to fly halfway around the world, it is unlikely that they will be able to sign each other's keys directly. However, a valid path in the PGP web-of-trust may already exist connecting them.

Second, requiring users to sign the key of each person they want to add to their Konfidi trust networks adds additional difficulty which should otherwise be avoided. In keeping with the recommended practices for PGP, two individuals must meet in person and verify photo identification before they are to sign each other's keys. This extra hassle might entice users to grow lax in their keysigning policy, failing to properly complete such requirements. This attitude, when widespread would substantially weaken the web-of-trust. By keeping the PGP web-of-trust separate from the Konfidi trust network, the strength of the web-of-trust will not be weakened needlessly.

An additional advantage of separating the two trust networks is usability. Aunt Sally can use Konfidi to indicate trust if she and one other person (say, a more technically savvy nephew) sign each other's keys. She will then probably be connected to the PGP web-of-trust within a reasonable distance of other family members which she is likely to include in her trust network. Now there is no need to teach Aunt Sally the requirements for key signing, and explaining why they must be done for each person she wishes to add to her Konfidi trust network. The system is easier to use, and the web-of-trust is less likely to be compromised[13].

The frontend uses a number of drivers in a Strategy pattern [**?**], so that different subsystems for doing PGP pathfinding can be interchanged as they are developed. The current version utilizes a pathfinder known as Wotsap [**?**], which downloads the complete web-of-trust from a keyserver and stores it in an intermediate format for searching.

3.2.3 *Trust Backend*

The Konfidi trust backend is responsible for storing the internal representation of the Konfidi trust network, incorporating updates into the network, and responding to queries about the nodes in the network.

The backend can register with a FOAFServer as a mirror to receive notification whenever a FOAF record with trust information is added or altered. It needn't ever notify the FOAFServer of any updates to its records, for the FOAFServer would be the only source of new information. This allows also for it to syncronize with the FOAFServer after a period of down time in which new records have been added. The backend currently assumes that the FOAFServer has verified the signatures of the FOAF records it stores, freeing it from the computational burden of fetching the signing keys and verifying the signature.

When the TrustServer updates a record, the backend parses the RDF input data and adds the relevant information to its internal representation of the trust network, which is a list of all `foaf:Person` records indexed by fingerprint and links to each `Person` marked as trusted, along with topic and rating data. The updated data will then be available for subsequent queries. This scheme accomplishes the goal of having trust links available in the proper direction, from source to sink, and avoiding one species of bogus data attack, as discussed in Section 2.5.

This representation requires $O(m + n)$ space to store and on average, $O(m * l)$ time to search, and $O(o + p)$ time to update, where m is the number of persons, n is the number of trust links, l is the average length of a path between two persons, o is the number of persons being updated, and p is the number of links being updated. On the other hand, a representation of a completely solved network, storing the trust values between any two individuals[14], requires $O(m^2)$ space, but makes trust queries take a maximum of $O(1)$ time. However, such a representation requires $O(m^2 * l)$ time to solve, which it must do again after every update, since it must recompute the value for every pair.[15]

It may also be advantageous to store trust links going the other direction, perhaps for local representation analysis, or auxiliary information like name or email address. Other information, such as when the record was last updated, could allow for record caching that might improve performance.

Because of the apparent lack of psychological research on trust representations, we have again implemented the Strategy pattern [**?**], for the trust propagation algorithm. This allows additional propagation strategies to be dropped into place as they are developed.[16] Our preliminary algorithm does simple multiplicative propagation over each link in a path. It uses something like a breadth-first search[17] to find the shortest path between source and sink, if one exists:

```
function findRating(source, sink):
```

[13] While the effects of individual keys being compromise on the web-of-trust as a whole would be restricted to the key's neighborhood in the web, as this happened with greater frequency, the usefulness of the entire web would be undermined.

[14] In other words, a lookup table.

[15] The tradeoff between storage space and query time made it hard to settle on a representation. Perhaps a compromise between a "live" system that incorporates incremental updates with slow queries, and a system that updates its network several times a day, rather than on each update, could provide better performance. Most users would not need up-to-date links with every user, since their queries would most likely be over a rather limited subset of the network. Caching of previously computed trust values on the user's end, with periodic updating, could also make a difference.

[16] It also served, unlooked for, as an excellent interface for various special queries we wanted to make, like getting a list of all the people in the network, or a full dump of all people and trust data.

[17] Prioritized to follow whichever path has highest value after each iteration.

```
keep a priority queue of all paths
until the sink is found
  find the path with the highest rating
  find the link not already seen
    concatenate ratings from path and link
    add the path with its rating to the queue
return the rating of the path to the sink
```

One such concatenation algorithm is a simple strategy like the following, which simply multiplies trust ratings along each step in the path:

$r = \prod_{i=0}^{n-1} Rating(i, i+1)$

where *Rating* returns the rating between two nodes in the network.

4. FOAF SERVER

The Konfidi server uses data from PGP keyservers to act on identity trust. To act on topical trust, we need a similar data store. This is not necessarily within the scope Konfidi, but is a necessary prerequisite. We created the FOAF Server to fulfill this need.

The FOAF Server is a web service that stores and serves FOAF files that include trust relationships as specified by our trust ontology. A seperate FOAF file is stored for each person, identified by their PGP fingerprint. All FOAF files must be PGP signed by the owner to prevent false data from being submitted and to prevent unauthorized modification of someone else's data. When a FOAF file is requested, the PGP signature is included also so that it may be verified by a client.

Multiple FOAF Servers will be available for public use and will synchronize their contents. Like the SKS PGP Keyserver[18], anti-entropy reconciliation[19] will be used, rather than the rumor-mongering reconciliation[20] used by traditional PGP keyservers. Synchronization will be PGP signed to maintain trusted secure communication channels everywhere.

Since the primary function of the FOAF Server is data storage, it may hold FOAF files that are not related to trust. A FOAF server may be configurable to act as one that is used for trust relationships, pet information, or rsums. Moreover, RDF features a `seeAlso` tag so a single FOAF file hosted on a FOAF server may refer to more FOAF data hosted elsewhere. This gives the owner flexibility, including encrypting or limiting access to the FOAF file.

Our FOAF Server is built with the Apache HTTP Server and mod_python using principles of REST architecture. Various clients can retrieve and set data using HTTP `PUT` and `GET` methods on URIs of the format `http://domain.org/foafserver/EAB0FABEDEA81AD4086902FE56F0526F9BB3CE70`. `PUT` requests must be of `Content-Type: multipart/signed` and requests are served with a content appropriate to the request's `Accept:` header. A web form for uploading FOAF files and their signatures is also provided.

Synchronization has not been implemented yet. Currently the Trust Server listens on a port for filenames that it should load into its memory. When someone updates a file with the FOAF Server, it sends the filename to the Trust Server update listening port and it reloads it. Thus currently the FOAF Server and Trust Server must run on systems with access to the same filesystem.

5. CLIENTS

The PGP, FOAF, and Konfidi servers each have clients which end-users use to view and modify the data.

5.1 PGP Clients

Many clients have already been written to interact with PGP keyservers with the Horowitz Key Protocol (HKP)[21]. The server itself also provides web forms to search for and view keys. It may be useful to integrate a PGP client with other Konfidi clients to provide a more cohesive user interface to the system.

5.2 FOAF Clients

The FOAF Server provides some web forms to allow users to upload FOAF documents and PGP signatures. We plan to develop desktop software for users to create, sign, and upload their FOAF documents. See Section 4 for a summary of the FOAF Server HTTP interface.

5.3 Konfidi Clients

Only the Command Line Email Client has been written yet, but most clients will work similarly, depending on the context in which they are used. We expect that to make Konfidi widely popular as a method of stopping spam, a plugin or extension for every major Mail User Agent (MUA) will need to be written.

5.3.1 Command Line Email Client

This client is designed to be invoked from a mail processing daemon, such as procmail[22]. It reads a single email message from standard in, adds several headers, and writes the message back to standard out. By doing this, a MUA can filter the message based on the value of the added headers.

The client does the following tasks:

1. determine the source's PGP fingerprint (normally from a configuration file)
2. remove any existing X-Konfidi-* and X-PGP-* headers[23]

[18] http://www.nongnu.org/sks/

[19] At each time of synchronization, servers synchronize the entire database no matter what the current states are. There is a trade-off between computation and communication expenses.

[20] Only the most recent updates are pushed to other servers; this does not allow servers to be out of communication for an extended period of time.

[21] A standard, yet undocumented, set of filenames and conventions using HTTP

[22] http://www.procmail.org/

[23] This is done in case a spammer sends an email with invalid headers in an attempt to get past the filter.

3. stop, if the message is not multipart/signed using PGP
4. stop, if the PGP signature does not validate
5. stop, if the `From:` header is not one of the email addresses listed on the key used to create the signature
6. query the konfidi server with the source (recipient) and sink (signer) PGP key fingerprints and topic "email"
7. recieve the computed trust value from the konfidi server

The client addes the following headers to the email:

Header	Value
`X-PGP-Signature:`	valid, invalid, etc
`X-PGP-Fingerprint:`	the hexadecimal value
`X-Konfidi-Email-Rating:`	decimal in [0-1]
`X-Konfidi-Email-Level:`	*s for easy matching e.g., `-Level: *******`
`X-Konfidi-Client:`	`cli-filter 0.1`

If the client stops at any point, it will still add appropriate headers before writing the message to standard out.

6. FUTURE WORK

There are a number of things to be done to develop Konfidi from a proof-of-concept to a useful system. As we've mentioned above, one thing we need most is a good base of psychological research backing up our trust representation and propagation, or suggesting a new one. Unfortunately, we must leave this to the experts in psychology, and so it is hard to say when it will be there. The rest of the system can be developed in its absence, so long as it is understood that we have just approximated how trust might work.

As we've said, a trust system is only as useful as it is trusted. Thus, a system of secure communication between every different components is required, most likely using PGP multipart/signed data. It is hard to say how a user's trust in a system like Konfidi can be represented within itself, but that may have implications, too.

In addition to plugins at the user's email client level, Konfidi could be incorporated into the email infrastructure at the mail transfer agent (MTA) level. Thus, a system could check Konfidi and add query results to every email message that it delivers to the user.

As the scope of Konfidi naturally expands to include things other than email, other clients will be developed. One possible client is an web browser extension to query pages when they are visited. Other extensions might allowing multipart MIME PGP signatures attached to webpages for easy verification.

For trust topics to be really useful, some sort of hierarchy is in order. Topics ought to standardized so that it is clear in what circumstances they apply, and how they relate to one another. So, for example, if Alice trusts Bob about internet communication in general, then if a query is made about email (a descendent of internet communication) and no explicit email rating is given, then Konfidi traverses up the heirarchy until some more general trust rating is found, and applies that.

7. CONCLUSIONS

8. ACKNOWLEDGMENTS

We would like to thank Jim Laing for assisting with test data, Prof. Vander Linden for advising us on this project, and Profs. Fife, Frens and Plantinga for their advice on specific matters.

APPENDIX
A. OWL TRUST SCHEMA

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY trust "http://svn.berlios.de/viewcvs/*checkout*/konfidi/schema/trunk/trust.owl#" >
    <!ENTITY rdf  "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
    <!ENTITY foaf "http://xmlns.com/foaf/0.1/" >
    <!ENTITY wot  "http://xmlns.com/wot/0.1/" >
    <!ENTITY rel "http://vocab.org/relationship/#" >
    <!ENTITY dc "http://purl.org/dc/elements/1.1/" >
    <!ENTITY vs "http://www.w3.org/2003/06/sw-vocab-status/ns#" >
  ]>

<rdf:RDF
    xmlns="&trust;"
    xmlns:owl="&owl;"
    xmlns:rdf="&rdf;"
    xmlns:rdfs="&rdfs;"
    xmlns:xsd="&xsd;"
    xmlns:rel="&rel;"
    xmlns:foaf="&foaf;"
    xmlns:wot="&wot;"
    xmlns:dc="&dc;"
    xmlns:vs="&vs;"
>

<rdf:Description rdf:about="">
    <dc:title xml:lang="en">Trust: A vocabulary for indicating trust relationships</dc:title>
    <dc:date>2005-03-20</dc:date>
    <dc:contributor>Andrew Schamp</dc:contributor>
    <dc:contributor>Dave Brondsema</dc:contributor>
</rdf:Description>

<owl:Ontology
    rdf:about="&trust;"
    dc:title="Trust Vocabulary"
    dc:description="The Trust RDF vocabulary, in OWL and RDF."
    dc:date="$Date: 2005/03/19 11:38:02 $"
    >
    <owl:versionInfo>v1.0</owl:versionInfo>
</owl:Ontology>

<!-- classes first -->
<owl:Class rdf:about="&trust;Item" rdfs:label="Item"
    rdfs:comment="An item of trust">
    <rdfs:isDefinedBy rdf:resource="&trust;" />
    <rdfs:subClassOf rdf:resource="&rdfs;Resource" />
</owl:Class>

<owl:Class rdf:about="&trust;Relationship" rdfs:label="Relationship"
    rdfs:comment="A relationship between two agents">
    <rdfs:isDefinedBy rdf:resource="&trust;" />
    <rdfs:subClassOf rdf:resource="&rel;Relationship" />
</owl:Class>

<!-- for constraints -->
<xsd:element xsd:name="percent" rdf:ID="percent">
    <xsd:simpleType>
        <xsd:restriction xsd:base="xsd:decimal">
```

```
            <xsd:totalDigits>4</xsd:totalDigits>
            <xsd:fractionDigits>2</xsd:fractionDigits>
            <xsd:minInclusive> 0.00</xsd:minInclusive>
            <xsd:maxInclusive> 1.00</xsd:maxInclusive>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>

<!-- properties second -->
<owl:ObjectProperty rdf:ID="truster" rdfs:label="truster"
    rdfs:comment="The agent doing the trusting.">
    <rdfs:domain rdf:resource="&trust;Relationship" />
    <rdfs:range rdf:resource="&foaf;Agent" />
    <rdfs:isDefinedBy rdf:resource="&trust;" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="trusted" rdfs:label="trusted"
    rdfs:comment="The agent being trusted.">
    <rdfs:domain rdf:resource="&trust;Relationship" />
    <rdfs:range rdf:resource="&foaf;Agent" />
    <rdfs:isDefinedBy rdf:resource="&trust;" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="about" rdfs:label="about"
    rdfs:comment="Relates things to trust items.">
    <rdfs:domain rdf:resource="&trust;Relationship" />
    <rdfs:range rdf:resource="#Item" />
    <rdfs:isDefinedBy rdf:resource="&trust;" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="rating" rdfs:label="rating">
    <rdfs:isDefinedBy rdf:resource="&trust;" />
    <rdfs:domain rdf:resource="#Item" />
    <rdfs:range rdf:resource="&rdfs;Literal" rdf:type="#percent" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="topic" rdfs:label="topic">
    <rdfs:isDefinedBy rdf:resource="&trust;" />
    <rdfs:domain rdf:resource="#Item" />
    <rdfs:range rdf:resource="&owl;Thing" />
</owl:ObjectProperty>


</rdf:RDF>
```

B. EXAMPLE TRUST NETWORK

edgecases example here