DSNP: A Protocol for Personal Identity and Communication on the Web

Dr. Adrian D. Thurston

Vancouver, British Columbia, Canada
thurston@complang.org

Abstract

The social web is emerging as a pervasive communication tool. The systems in use provide us with online identity. They give us tools for efficiently communicating with people we know. While the emergence of this new paradigm is to be appreciated, there are improvements to be made. The existing tools force us to consider who we are able to communicate with when we decide who our communication service provider is. As we collectively discover the value in the social web, we must lift this constraint. We must go from pockets of social activity in closed systems, to global social activity over open protocols. Distributed Social Networking Protocol (DSNP) one such protocol. In this position paper we cover the most important elements of DSNP. These elements are tools for taking the user experiences of the social web into an open space were everyone is free to contribute to an ecosystem of software and techniques.

1 Introduction

Social networking sites are becoming essential communication tools. The paradigms are becoming ingrained in our day-to-day communication, but we are behind the needed state of affairs because the de facto tools are closed systems. Our identities are becoming fused with the companies that provide us with the software. Our data is not owned by us. We have little choice and therefore little control. The good news is that the existing systems represent the first generation of this technology. The next generation will be built on a foundation of open technology. This paper focuses on the design of one candidate – Distributed Social Networking Protocol (DSNP).

When we look closely at existing social networking systems we find that underlying all the features are two basic elements, identity and message broadcast. When a user takes an action, such as uploading a file or writing a blog post, they do it with their own identity as the host of the content. At other times, content is created on a friend's identity. In both cases, notification of the activity is broadcast to the user's contacts. Once received by contacts, it can be filtered and displayed appropriately.

In a closed system this is an easy problem. We can have a collection of users with some internal encoding of identity. User-created content is encoded in the same database using the same schemes. Displaying broadcasted information is simply a problem of selecting the appropriate data from the common database. So long as the system is programmed securely, no user can break the rules by injectng messages to appear as though they come from someone else. No user can or snoop on messages they are not entitled to see.

Once we start moving to a decentralized sys-

tem, we gain badly needed freedom, but at the same time we introduce a number of security issues. How does one stop people from sniffing activity? How about forging messages? The integrity of our identity is important to us, and without ground rules and some basic protections, any proposed system will quickly become unusable.

In this paper we reduce social networking activities to a small number of functions. We do not need to compromise on security or reduce the quality of the user experience when we break from a central database. We give an overview of our techniques and we present a software architecture designed for rapid uptake of these ideas.

2 RSA-Based Identity

If we consider the Pretty Good Privacy system (PGP) as a social network, one function it does really well is declare and protect identities. Each identity gets an RSA asymmetric key. It is used as the foundation of the identity. In DSNP we do the same; we allocate key pairs, distribute public keys, and use the key pairs for signing and encryption. We take out the email address from the public portion and put in a user's URI.

We take a different approach to managing private keys. It is unlikely that we can expect an average user to manage their own keys, so we store keys on the server. This is something of a security risk that we must mitigate. We utilize not a single private key, but several keys, with varing privilege levels.

The most secure key is allocated at identity creation time, encrypted with a password, and decrypted for use very infrequently. It is reserved for proving intent of significant operations, such as identity movement or deletion.

Then next most secure key is stored encrypted, is decrypted on login, but is immediately discarded from memory after use. It can be used to sign login tokens that prove recent login activity.

The third most secure key is stored encrypted and retained in memory unencrypted, but only while the user is logged in. It is used for any signing activity that requires a currently logged-in user. This key is required most often, as much of the protocol covers activity of logged-in users.

The final key, which is the least secure, is stored unencrypted and is always available for use, regardless of whether the user is logged in or not. This is used for activity that does not require a logged in user. It is typically in response to other user's queries or requests, for example, submitted friend requests.

If we know our peers will accept signatures at certain privilege levels for certain actions, and we are diligent about encrypted storage, we protect ourselves from some forms of unauthorized access to our identity. For example, we can anticipate recovering from a server theft under this scheme. The most secure key won't be accessible by anyone who has acquired the server hardware, even if they manage to keep it running during the theft. Assuming adequate backups are available, the identity can then be safely moved to another site by notifying all contacts of the change.

We use asymmetric keys to protect identities, encrypt messages, sign messages. Next we need to build out a protocol that supports friending, passwordless login, message passing, distributed agreement.

3 Friending Protocol

The first step in a social network interaction is to establish contacts. We have three computers involved int this action: the browser initiating the request, the server representing the browser's identity, and the server representing the recipient of the friend request.

It is critical that the recipient's server be assured the browser is the actual owner of the identity they are claiming. To satisfy this we must challenge the browser to return a token that only the browser's server is privy to. The browser must go home to their own server to retrieve this token, then submit it as part of the final friend request. Once the request is received, it is stored on the recipient's server to later be accepted or denied. This final step involves a notification message to the requestor.

This challenge-response pattern is repeated throughout DSNP. We often have some browser

on another user's site and the browser needs to prove ownership of their identity. A token is encrypted to the identity's appropriate RSA key, the browser goes home, and if logged in correctly, fetches the token and returns it to the site that issued the challenge.

4 Passwordless Login

The pattern used for friending is also used for passwordless login. If a user wishes to login to a friend's site, a login token is allocated and encrypted to the browser's identity. The browser must go home, prove she is logged in, decrypt the token, then return it to the friend's site. From there the browser is granted access the identity in a role that indicates she is a friend.

5 Broadcasting

The basic element in social networking is the broadcasted message. All activity by a person is broadacasted to others, then displayed in an aggregate news feed. These feeds are filtered and restricted by the recipient's viewing preferences.

Considering a user's contact list can grow to a large number of recipients, perhaps on the order of thousands, we must consider the cost of RSA-based encryption to contact lists of this size. Several thousand RSA-based encryptions can easily take up seconds of CPU time. If we wish to develop systems that host thousands of users, we are creating for ourselves a performance problem.

To solve this problem, DSNP introduces a broadcast key. This is a pre-shared symmetric key that is given to contacts ahead of time. All broadcast activity can be encrypted once and the single message copied to all contacts. The key has a generation associated with it, so if it needs to be replaced because a group member is being removed, it can be delivered one user at a time, then the new generation made active once it has been fully replaced.

6 Message Signing Keys

Broadcast and direct friend-to-friend messages must be signed to prove they come from a valid contact. Signing is both good and bad. It is good because it eliminates the possibility of attackers injecting messages into our news feed. It is bad because we give people proof that we have said something. While not expected and always unfortunate, we have to consider the possibility that something we say can be used against us. We must introduce some deniability.

To solve this, we never sign the things we say with our permanent identity keys, instead we distribute signing keys and use those. Signing keys are less valuable to us and can be revealed should we absolutely need to deny something we have said. We can do this while preserving our most valuable keys and therefore preserve our identity.

Signing keys are given directly to every contact. We compose a unique message that contains the recipient's identity and the public portion of the disposable signing key, then sign that message with one of our permanent keys. If someone makes public something we have said and they wish to prove it, they must also reveal their own identity along with the proof.

Next, we are able to divulge the signing key and create a situation such that anyone is able to produce the proof that the attacker produced. Once a key has been revealed, the attacker then faces the problem of proving that the signed message they received was produced before the key was revealed. This is difficult because it is easy to cast doubt on any such assertion.

Before we reveal a signing key, we must first inform all of our remaining friendly contacts that the key is no longer good, otherwise those contacts may receive forged messages and believe them. We therefore have to track who we have given signing keys to. This applies to friends, and friends of friends who have received notifications of our activity.

Key revealing should be considered a last line of defense, as it is indeed cumbersome to ensure that a signing key is no longer trusted by the contacts who have previously received it. It is a nice option to have, nonetheless.

7 Remote Broadcasting

Many social social activities can be considered messages with several actors. For example, a "board post" is a hosted message with a publisher and an author. The publisher is the owner of the board and the author is the friend making the post. Notification of the post is broadcast to the contact list of both parties.

As another example, a photo tag is a message containing co-ordinates and a reference to a hosted photo. It can involve up to three actors. The publisher is the owner of the photo. The subject is the person being tagged, and the author is the person doing the tagging.

In DSNP we generalize multi party activity into a function called 'remote broadcasting' and we propose reducing social activity to this function. The kinds actors that can be involved are the publisher, the author, and some number of subjects. Any of the actors are able to broadcast the message to their contact list, and when they do so, they are also called a broadcaster.

We require all actors to agree on every remotely broadcasted message. On some message types we want the actor to be logged in to authorize the message, and on others, the message can be automatically authorized. Authors should always be logged in. For the remaining combinations of message type and actor type, these decisions can be based on each user's preferences. For example, a user can deny a signing request on a photo tag until they have had a chance to review the tag, or they can automatically accept all photo tags by certain users. Regardless of the particular policy in place, once agreement has been reached by all parties, the message can be broadcast.

Like the friending protocol and passwordless login, remote broadcasting involves a challenge and response to the author. For example, a "board post" requires that the browser prove they are in control of the author identity and intend to compose the message. They most go home to obtain the challenge response. This also gives them the opportunity to register the composition of the message and produce the signature that will be distributed to others.

8 Software Architecture

It is clear that new protocols for social networking are needed. Proliferating an implementation of a new protocol is a daunting task. There is no canonical language of the web. We have Ruby, Python, Perl, PHP, and many other languages that are used to write the supporting frameworks. We cannot favour any single language or framework.

For the federated social web to flourish, we need web-based systems that communicate asynchronously. There will be encryption involved. There will be a desire for carefully engineered systems that support high-volume, low-latency message processing. It is therefore natural to move protocol implementation into a long-running daemon, separating the user interface from protocol implementation details in a language-agnostic way.

The reference DSNP implementation has been developed this way. It is separated into a content manager, which provides the web-based UI, and a daemon that deals with the protocol specifics. The daemon is reusable by other systems. It is is responsible for server-server communication, but also serves the web-based content managers.

The content manager is responsible for providing all aspects of the user experience. It can be written much the same way non-distributed social networking systems are written. It should be designed to support a collection of users and contain numerous functions for managing content. How it should differ from existing systems is that it must defer to the daemon for all communication between identities. If it follows the protocol, as implemented by the daemon, identities can communicate not only with other users on the same site, but with users on other sites, including those running software written by other people.

8.1 User Representation

In DSNP a user is identified by her IDURI. This is a URI with scheme `dsnp`, followed by a host and path, and optional arguments. Since this is an arbitrary URI, we allow any user identification plan (eg `?uid=N`, or `/user.name/`). When an IDURI is written using the `https` scheme

it is called an IDURL and it is expected to be accessible from a web browser.

8.2 Command/Notification

The DSNP daemon exports a set of commands for content managers to use and makes them available over a local socket. It expected that the content manager will make available a program that the DSNP daemon can use to notify the content manager of various events, such as message arrival, or friend acceptance. Both of these languages are text-based and are designed to be easily constructed and parsed from common web application languages.

The relid request is an example of a command for use by content-managers. This is the command that should be called when a friend request is received. It begins the friend request process.

```
"RELID_REQUEST" SP
        user-iduri SP
        friend-iduri CRLF
```

The DSNP daemon responds with a request identifier. The content manager then redirects the browser to their home identity where the identifier is submitted using a command similar to this one (relid-response).

When the DSNP daemon has something it needs to tell the content manager about, such as a friend request being accepted, it uses the notification interface. It will fork and execute a program that is owned by the content manager and send it the notification using a syntax similar to the above. The following is the syntax for the friend-request-accepted notification.

```
"FRIEND_REQUEST_ACCEPTED" SP
        user-iduri SP
        friend-iduri SP
        accept-reqid CRLF
```

The content manager can then register in its database that the event occured and appropriately notify the user.

8.3 Standard HTTP Args

DSNP requires that browsers be redirected from site to site. Redirection is necessary in the friending, passwordless login, as well as remote broadcasting portions of the protocol. To ensure interoperability, DSNP prescribes a set of standard HTTP args that must be added to IDURIs to indicate certain functions, or to provide associated information. The following is an example of the arguments added to the destination identity URI when the browser is sent home in the second step of the friend process.

```
addArgs( $destIduri,
    "dsnp=relid_response",
    "dsnp_iduri=$userIduri",
    "dsnp_reqid=$reqid"
)
```

8.4 Implementation

Most of the ideas the ideas expressed in this paper are implemented in the reference implementations. The reference daemon is called DSNPd and the content manager is called Choice Social. The protocol is described in the DSNP specification, all of which are available from the DSNP home page.

`http://www.complang.org/dsnp/`

This protocol is still being developed and is expected to change. The intent is to continuously maintain a working reference implementation. As of this writing, the protocol is at version 0.6.

9 About the Author

Adrian D. Thurston holds a Ph.D. in Computer Science from Queen's University, where he studied and developed source transformation systems. Adrian is the author of Ragel, a unique software development tool for producing very fast parsers. He works in the field of network security where he designs and develops real-time parsing systems that analyse network traffic for evidence of security events.