Musubi: Disintermediated Interactive Social Feeds for Mobile Devices

Ben Dodson   Ian Vo   T. J. Purtell   Aemon Cannon   Monica S. Lam

Computer Science Department
Stanford University
Stanford, CA 94305
{bjdodson, ianvo, tpurtell, aemon, lam}@cs.stanford.edu

ABSTRACT

This paper presents Musubi, a mobile social application platform that enables users to share any data type in real-time feeds created by any application on the phone. Musubi is unique in providing a disintermediated service to end users; all communication is supported using public key encryption thus leaking no user information to a third party.

Despite the heavy use of cryptography to provide user authentication and access control, users found Musubi simple to use. We embed key exchange within familiar friending actions, and allow users to interact with any friend in their address books without requiring them to join a common network a priori. Our feed abstraction allows users to easily exercise access control. All data reside on the phone, granting users the freedom to apply applications of their choice.

In addition to disintermediating personal messaging, we have created an application platform to support multi-party software with the same respect for personal data. The SocialKit library we created on top of Musubi's trusted communication protocol facilitates the development of multi-party applications and integrates with Musubi to provide a compelling group application experience. SocialKit allows developers to make social, interactive, privacy-honoring applications without needing to host their own servers.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*; E.3 [**Data**]: Data Encryption—*public key cryptosystems*

General Terms

Design, Security

Keywords

Mobile, Social, Privacy, Platform, RSA

1. INTRODUCTION

Smartphones are fast becoming the device of choice when it comes to personal and social computing. We take pictures on the phone, we play music on the phone, and we can even remotely control our TVs. Through Bluetooth, NFC (Near-Field Communication), and QR codes we can build ad-hoc personal device networks. Eventually we will have all our credentials on the phone so we can unlock paid content anywhere without logging in.

As the gateway to everything personal and digital, the phone can be viewed as an extension of ourselves. Being constantly on and connected, the phone has the ability to let us share and celebrate all our daily moments with our friends and loved ones. For those who do not wish to give up the totality of our digital selves to a for-profit third party, the phones can allow us to interact with others without intermediation. This paper presents Musubi which allows users to share virtually anything with our contacts without sacrificing data privacy or ownership.

1.1 Disintermediation of Social Interactions

Practically everything social is intermediated today, from full-blown social networking to simple games like Tic-Tac-Toe. While earlier social networks would acquire their own social graphs, now almost all social software shares through existing social networks using protocols like Facebook Connect. Users interested in owning their data and controlling access beyond what is made available by such networks have few alternatives. As more information is aggregated and owned by the provider of the proprietary social graph, a powerful monopoly that owns the proprietary social app platform may emerge to the detriment of competition, innovation, and welfare of the consumers.

The closed proprietary social networks we have today do not comply with privacy protection regulations such as COPPA and HIPAA [1]. They do not serve the communication needs of individuals or corporations who wish to own and control their data. Even among those less concerned about privacy, they need to exercise caution using such networks. They need to protect their image, which is defined by not just what they post, but what their family, friends and friends-of-friends post as well.

Personal smartphones make possible the notion of disintermediated social services. Disintermediation is attractive not just for sharing confidential information, but for friends to share anything with each other freely from silly pictures, random thoughts, to dark secrets. However, while phones have persistent connections to the Internet, mobile IP networks disallow incoming connections to the devices. We created the Trusted Group Communication Protocol, or TGCP, to enable phones to contact each other through a cloud relay while providing information confidentiality. Data is pro-

WWW 2012, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1229-5/12/04.

tected using public key cryptography and the data is decrypted only on end users' devices.

1.2 A Decentralized Trusted Social Graph

In the Musubi system, there is no single central social graph. Instead, our friends' contact information is stored purely in our address book; we can interact with any of our friends directly, without requiring them to sign up on an external social network. Individuals are identified by public keys. PKI systems for personal identity are notoriously hard to use [32]. Our goal is to hide the security mechanisms so that even a child could use them. We leverage smartphone technologies such as NFC and Bluetooth to allow users to seamlessly connect with each other as they interact in the physical world. We provide a group-centric user interface that aggregates application activity and communication into interactive private social feeds.

1.3 Sharing Everything Interactively

Musubi provides a *social feed* that lets groups of friends share everything on the phone frictionlessly with each other in real time, from status updates and photos, names of installed apps, to real-time multi-party app sessions. We made the Musubi disintermediated communication primitives available to app developers in a library called SocialKit. The library helps connect participants, exchange messages among participants by reading and writing to feeds, and maintain distributed state by providing an automatically replicated social database. No personal friend information is leaked to the social apps on Musubi, nor do these apps need to host a server.

1.4 Contributions

This paper presents the design of a complete social sharing platform for smartphones. The contributions of this paper include:

- Musubi, which allows users to interact with their friends right out of their address books. Users can share just about everything on friends' feeds. The Musubi feeds let collaborative apps advertise their progress on the same feed; users can launch any of the shared apps easily by clicking on an entry. Musubi also helps friends establish trust with one another by hiding key exchange in common friending interactions.
- TGCP: a trusted communication protocol for exchanging encrypted messages within groups on the mobile phone. This is the backbone of our disintermediated services.
- SocialKit, an API of a social application platform for creating multi-party apps with disintermediated communication.
- a large collection of Musubi applications that range from simple sharing of to-do lists to engaging multi-player games. All these disintermediated apps can be developed quickly using our application platform and require no servers.

The organization of the rest of the paper is as follows. We first present the user experience by way of an example. Sections 3, 4, 5 and 6 present key concepts in our design: the decentralize social graph, TGCP, the Musubi feed, and the SocialKit library. We describe our experience with our system and report on the apps developed on Musubi in Section 7. We then discuss related work and conclude.

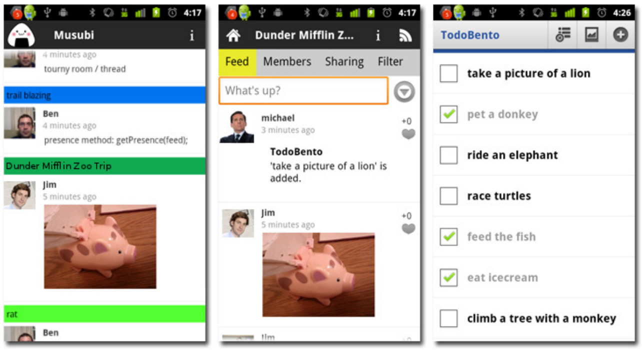
2. USER EXPERIENCE

Our goal is to make the Musubi experience feel as natural and integrated as possible by reducing the frictions in starting up, friend interactions, and the app sharing experience. Here we provide some sample scenarios to show how Musubi is integrated into the social experience on mobile devices. Note below that the user is unaware of the underlying cryptographic operations involved.

Acquiring new friends. Michael meets new customers every day. With Musubi's help, he now manages all of his business associates on his phone. Instead of exchanging business cards with them, he simply touches his phone to theirs to become friends over Musubi and he can now message them easily and securely.

Easy access control. Michael has been using Musubi for some time now and belongs to a few groups: "College Buddies", "The Scotts", and "Dunder Mifflin Paper." He likes this separation of friends, family, and coworkers, as he can act naturally in each group without concerns of how other groups would perceive him. He meets Bob, a college friend from out of town at a bar. He shows Bob a picture shared by Ann, a mutual friend, on the "College Buddies" feed. He brings up a camera with a click and snaps a picture of Bob and himself. The picture is shared immediately on the same feed, Michael can rest assured that his mother, who is opposed to alcohol, will not see the picture.

Group sharing. Michael takes his sales team to the zoo in celebration of a good past quarter. To keep a record of the event, he creates a new group "Dunder Mifflin Zoo Trip." He broadcasts the group locally using GPS and password protects it. His employees check "Nearby Groups" on their phones, enter in the password, and join Michael's new group. Michael creates a to-do list of goals for the day with the TodoBento application, as seen in Figure 1. Everyone turns on the "Share photos" mode, and all of the pictures they take on their phone while in this mode are automatically shared with just the "Dunder Mifflin Zoo Trip" group.



4 minutes ago
tourny room / thread

trail blazing

Ben
4 minutes ago
presence method: getPresence(feed);

Dunder Mifflin Zoo Trip

Jim
5 minutes ago

rat

Ben

michael
3 minutes ago

TodoBento
'take a picture of a lion' is added.

Jim
5 minutes ago

☐ take a picture of a lion
☑ pet a donkey
☐ ride an elephant
☐ race turtles
☑ feed the fish
☑ eat icecream
☐ climb a tree with a monkey

Figure 1: A list of Musubi feeds, a single Musubi feed, and the TodoBento application.

Non-Musubi applications. Michael sees that Ann posted a picture of a cat in the "Dunder Mifflin Paper" feed and can't help but think that it would be hilarious to see the cat with a monocle. He clicks the cat picture, chooses to edit, and loads the cat picture into PicSay, a free third-party photo-editor he downloaded from the market. After he puts the finishing touches on his cat, he presses the back button and his well-dressed cat automatically appears in the feed.

Easy multi-party applications. Ann starts a poker game in the Musubi feed as an invitation for Michael to join. Michael has not heard of that app before; he clicks on the game and is automatically brought to the Android market page for the poker app. After he installs it, he joins the poker game in session and begins playing with Ann. He notices that this free app has no ads because it is an open-source distributed app that requires no server infrastructure.

The above scenarios demonstrate how natural it is for new and old friends to share information and play games with different groups on the phone. The user can discover new apps and install them easily. Existing apps get shared socially without modification and free lightweight multi-party apps are enabled, all without losing any personal or social data to a third party.

3. A DECENTRALIZED SOCIAL GRAPH

Musubi is built upon the familiar address book model. Once friends have exchanged contact information, which include public keys in this case, they are able to contact each other without needing to sign up with a common third-party provider. Thus, Musubi has a decentralized social graph, with only users knowing who their friends are.

3.1 Key Management

By focusing on mobile, we take advantage of the fact that the phone is primarily a single-owner device. Accounts can be bound to a device eliminating the need for account registration. Upon installation, a public/private key pair is generated on the user's behalf.

Users have the option of associating their public keys with their web identities to facilitate key distribution and revocation. If users lose or leak their secret key, they can simply update their public keys. Public keys can be stored with web finger providers [34] such as Google. There has also been a proposal to place public keys on Facebook profiles using SocialKeys [17]. Users can import keys of their friends into their address book just as they would import phone numbers and addresses.

We note that some users, such as peace activists in authoritarian regimes, may not wish to link their accounts with any known identities. They can open an anonymous account with some identity provider or they can simply handle the key distribution and revocation themselves.

NFC-enabled mobile devices offer an easy and direct way to exchange public keys through physical contact. Musubi takes advantage of the natural handshake interaction that people perform when they introduce themselves. By touching phones, we can utilize NFC to directly exchange public keys, thereby establishing friendship. For phones without NFC, we use QR codes that embed the public key.

3.2 Friends and Groups

Public keys are well-known, so anyone can send a message to a Musubi user. For Musubi to be a successful social network, it is important to eliminate spam. Our design is based on the premise that friends should be able to exchange keys with each other. In Musubi, we define *friends* as people who have exchanged public keys and the keys are in their respective address books. Messages received from a non-friend are summarily discarded.

Musubi groups allow for the mass acquisition of friends. Any member in a Musubi group can invite their friends to join the group, and public keys are automatically shared among group members. There is no central group server; the membership is replicated on all the members' devices, and messages to the group are simply addressed to all the members directly. Each group has a public and private key pair which is used for group membership control messages.

A group invitation message contains the group's public-private key pair. This key pair is used to validate that people claiming to be part of the group are actually trusted. Once the user clicks on the invitation, they accept the key-pair as a means for validating new friends. Once the group's public key is entered into the address book, messages that include a signature based on the group key will be accepted even if the sender is unknown. This allows friends to extend the social circles of their peers in a seamless fashion.

Figure 2 shows the protocol for sharing a group and maintaining the group membership. First, a user creates a group, which consists of a name and a public and private key pair. Then he sends an invitation message to all of the desired participants. Users who want to join the group respond to this request (at the behest of the user) with a join notification that indicates their participation in the group feed. The originator of the group then responds with a message that contains the complete membership list of the group.

This message is signed with the group's public key. When the invitee eventually receives the join response, he will send a notification to all of the other members with his perceived membership list for the group. Whenever a participant receives a membership list signed by the group key, Musubi extends the membership of the group to include all of the asserted members. Any member can extend invitations to new members in the same way.

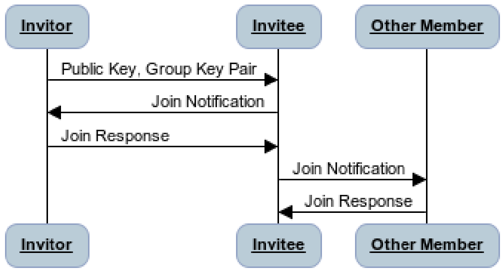


Figure 2: Any group member can invite new participants by sharing the group key pair and a partial list of group members with another Musubi user.

3.3 Invitation through External Channels

Musubi users can extend group invitations using external channels. A group invitation can be encoded as an HTTP link. The link encodes the group's public and private keys, sender's public key, and a friendly name for the group as parameters. The domain of the link is that of Musubi's home page, allowing the app to register itself to handle such links directly.

The link can be distributed through any medium, for example NFC, email, a GPS-based group broadcasting server, or on a (access-controlled) website. If Musubi is not installed, the link leads to a webpage prompting the user to install it. Once installed, a second click of the link is intercepted by Musubi to initiate the group joining process.

4. TRUSTED GROUP COMMUNICATION PROTOCOL

Mobile phones are the primary platform of a modern social network. These devices are always on, often connected, and intimately involved with our daily lives. Unfortunately, the 3G network service provided for these devices usually does not allow for incoming connections to be accepted by the device. This limitation mandates the use of an external service to buffer messaging between smartphones. Furthermore, aggregating non-critical message delivery allows for significantly reduced power consumption [19].

To preserve the principle of disintermediation, the Musubi social application platform relies on end-to-end encryption of messages addressed to public keys. We propose the Trusted Group Communication Protocol, or TGCP, for the required message transport primitives for the application platform. TGCP is a multicast messaging system that provides public-key based routing, server-hosted message queues, and connection aggregation. Each user, and hence each public-key, has a message queue associated with it on the TGCP server. Devices receives new messages from their corresponding message queues. When a user sends a message to a friend, it sends it through the TGCP server. Thus each users' device only needs to maintain a connection to the TGCP server to interact with their friends' devices.

4.1 Public-Key Based Messaging

Instead of using human readable names such as an email address, TGCP specifies that a public key be used as a global identifier. Messages are addressed to individuals by their public keys; the content of the messages are encrypted and signed with the senders' private keys. A user can hand the encrypted message off to any server to forward the message to a recipient, as only the recipients can decrypt the messages with their secret keys. We require all messages be signed so the recipient can authenticate the sender. Servers can even cache large amounts of data for clients without imposing any risk to privacy. This design preserves the disintermediation principle because the messaging subsystem is not privy to the content of the messages. However, the server is given a pseudononymous view of the structure of groups and frequency of messages.

Figure 3 shows the message format of a TGCP message. Each message consists of a header and a payload. The payload is encrypted with a 128-bit AES key using CBC mode with PKCS5 padding. Each message is encrypted with a different key, which itself is encrypted with the RSA public key for each of the recipients of the message using ECB mode with PKCS1 padding. The sender's public key and an RSASSA-PKCS1-v1_5 signature using the SHA1 hash function for the full message are also included in the header to prevent tampering [11].

signature	from	to$_0$	key$_0$	...	to$_n$	key$_n$	payload

Figure 3: A TGCP packet is encrypted with a per-message key and signed for authenticity.

4.2 Federation

One possible implementation of TGCP is a centralized cluster that handles the routing of message between all mobile phone users. We have implemented this design to explore the higher level challenges inherent in implementing a decentralizable application platform. For the purpose of geographic distribution and choice of different providers, a federated design is necessary. In a federated model, each public key is associated with a server, or home agent, that serves as the designated contact point for a mobile device.

Federated TGCP aggregates all social activity into a single stream of messages to and from the home agent. The protocol implies a network topology where many clients are connected to a single server, and each server is connected to many other servers. The servers participate in the peer-to-peer network that routes traffic between devices, while the client simply pushes and pulls messages to and from its self-designated home agent.

TGCP is intended to operate using semi-trusted networks of home agents, such as those that might be provided by telecom operators. Messages are addressed to a set of individuals by specifying their public keys with an encoded TGCP packet. Once more than one server is available for message routing, it is necessary to designate which server is pertinent to the specific recipient of a message. The expected home agent for a particular keyed identity is also encoded in the TGCP packet. This allows the common case of messaging to happen without requiring servers to consult a global database of designated home agents. The client can easily keep track of the home agent for its contacts because the home agent designation changes infrequently.

5. MUSUBI FEEDS

Musubi is designed to be a highly engaging application platform. Users can share everything, not only status updates and photos, but arbitrary content generated by any application on the phone as well as interactive application states that solicit active and ongoing participation from group members.

5.1 The Feed Abstraction

All objects shared within a group are organized in the abstraction we call a *feed*. A feed can be a long-lasting connection with a mostly fixed group of people. A feed can also be associated with a physical place such as someone's house, with its members being the people and devices present there. A conference room can associate with several feeds throughout the course of a day, one for each meeting, allowing applications and data to be shared easily with the people and devices nearby.

The feed is, in a sense, an extensible semantic web consisting of members and objects with access protection, as illustrated in Figure 4. Only members in the group can submit objects to the feed, and everything is replicated on members' phones. Data objects can be of well-known, renderable types such as photos or text, or they may be of a type without an associated view, useful for storing internal application state. The meta-information associated with the objects provides the semantic relationships between them. It includes information about the user who submits the object, the application that creates the object, the time of creation, whether the object is a response to another, and so forth. Musubi displays renderable data in order of its arrival time, and all data is stored in Musubi's *social database*, which can be accessed by applications in controlled ways.

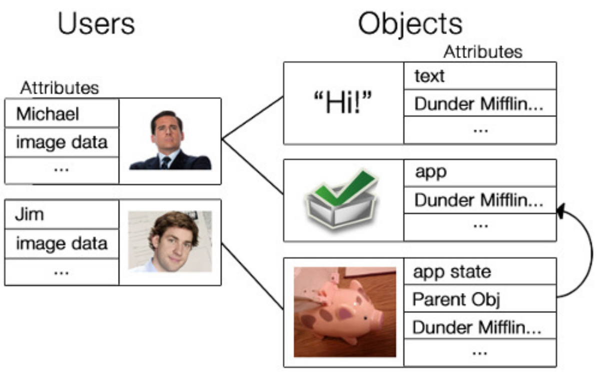


Figure 4: The Musubi API builds a semantic web of objects and users with the feed defining the context.

5.2 Interactions on a Feed

Besides letting users type in status updates or post pictures, users can interact with content on the feed in several novel ways.

Data import and export. Musubi supports the basic kind of sharing available on today's mobile devices. For example, a link can be shared from a web browser with a few clicks: the user elects to share a link, chooses Musubi, and then chooses a set of recipients which can be either friends or feeds. Later, the link can be exported from the feed to email, or launched in the browser.

In-place editing. Users can edit objects in the feed and reshare them easily. They click on an object, pick among a list of applications installed on the phone that support editing that object, and the data is shared upon exiting the program. As seen in Figure 5, two clicks allow a user to invoke their favorite photo editing application to edit a picture that has been shared in a feed, and when he is done, the content is instantly shared with the group. This basic interactivity is enabled through the "edit" intent on the Android platform. The Musubi feed provides the context of how the data is shared.

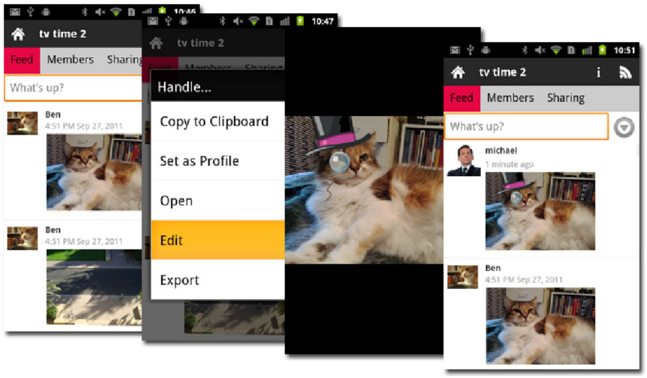


Figure 5: Using the PicSay photo editor to modify a picture in a feed.

Virtual presence. Musubi takes advantage of being "mobile-first" by supporting ongoing engagement that extends beyond Musubi's core application. The phone is an important personal accompaniment and is increasingly privy to our daily goings-on. Musubi makes it easy for friends to continually share activities through the phone, giving them a persistent sense of companionship.

With a click on the feed of their choice, users can elect to share a variety of ongoing activities from their phone as "presence updates" to a feed. For example, users can enable GPS location sharing. This feature is similar to several other services such as Loopt or Google Latitude, but without intermediation.

The "camera presence" shares any newly captured images with a group whenever a photo is captured with the built-in Android camera application. A group of friends can enable camera presence during a night out to automatically share their captured experience with each other without requiring a centralized service. We can also broadcast the music we listen to and even the TV shows we watch. Sharing such data with feeds provides fresh context for us to interact with friends while limiting the scope of what we share.

Multi-party applications. All the modalities described so far show how users can share information of existing applications on mobile devices, provided that these applications support the proper intents. Musubi also exposes an API, described in the next section, which further enables new kinds of disintermediated interactions built by third-party developers. Interactions with these applications are analogous to the above; an app adds a visual representation of itself to a feed, and clicking on the feed entry launches the application for further interaction.

6. MULTI-PARTY APP PLATFORM

Writing multi-party games for mobile phones remains a daunting task for developers today. Consider the exercise of writing a basic game of Tic-Tac-Toe for phones, played across two remote devices. In the standard model, we must (1) choose an API for pairing friends, such as Facebook Connect; (2) set up a server to handle the pairing; (3) manage push notifications from server to client; (4) write the mobile client. Musubi reduces the task to only writing the mobile client.

We expose the platform's interaction primitives through an API allowing application writers to simply write the mobile client code. They do not have to set up a server. Note that we do not preclude the development of server-client apps; we are simply providing an alternative model that was not available before.

6.1 Overview

A decentralized multi-party application needs:

1. a group of participants,
2. a sequence of objects that represent interactions between members,
3. distribution or replication of these objects across members' devices.

The feed is a basic example of such an application. We further extend the feed primitives to multi-party application developers so apps can be developed as a *subfeed* within a feed. The main feed serves as a form of a "bulletin board" of arbitrary posts from a mix of applications, where users can issue and accept invitations to new app sessions. Users can easily join these sessions even if they do not have that app installed. Moreover, they do not have to monitor the status of independent applications, users are alerted of state updates from any applications on the general feed.

In this model, an application is bound by a parent feed and communicates purely with feed primitives. Musubi provides an *identity firewall* that enables applications to send messages to a users' friends without learning their identities.

Our Android version of Musubi provides an API that is accessible to other native appicatitions through the use of built-in interprocess communication mechanisms including Content Providers and Intents. On other mobile platforms like iOS, applications are sandboxed and we cannot provide this level of communication. We thus provide API access to HTML5 applications which execute within a web container embedded inside the Musubi application.

The following describes the application platform we have built for the Android OS. The API is accessible directly as a Content Provider representing feed data. We also provide a library called SocialKit to further simplify the development of multi-party applications.

6.2 Subfeeds

The abstraction of a subfeed is used for grouping messages from the same application session as well as for grouping user responses to messages within a feed. Technically, every object in a feed is the head of a subfeed. Each object has a universal identifier generated from the TGCP's packet signature which we reference in our semantic graph. The subfeed is then a collection of objects that lists the head as a "parent."

Independent subfeeds allow different app instances (such as two different games of poker) to maintain separate state. Musubi supports a mechanism for updating the view within a feed that is associated with an app instance by sending a specially typed object to the subfeed. The rendering is available on devices even if the application is not yet installed.

Figure 6 demonstrates how a poker application interacts with a Musubi feed. The feed has two different application instances, each representing a game played with different people. The poker app associates internal state information with each session which is shared across all participants. It also periodically updates the visual representation of the game so it can be rendered in Musubi as seen in Figure 8.

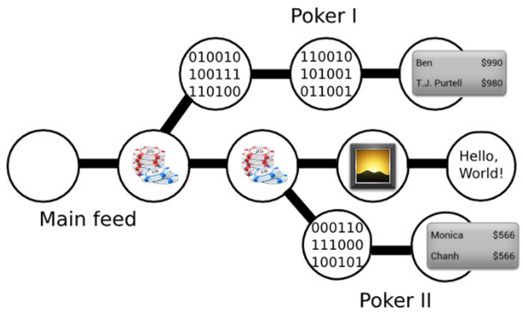


Figure 6: A feed showing two sessions of a poker application, an image, and a text object. Each poker session has a subfeed of non-rendered session data as well as thumbnails representing the current state.

There are two ways in which an application can attach itself as a subfeed to a feed. The user can first pick the people with whom he wishes to interact and then the application, or he can pick the app first and then choose participants.

In the first case, the user launches an application within a Musubi feed. When a user views a feed, a button press allows him to choose an application to launch, adding it to the feed. The application is automatically handed a subfeed identifier, which it can use for subsequent communication. In the second case, an application can call into Musubi to prompt the user for a feed. On Android, this is achieved by having the app send an intent to Musubi requesting that Musubi ask the user to pick a feed; Musubi ultimately responds to this intent with the feed identifier. During the process, a user may create a new feed for the application.

6.3 Messaging

Once an app is connected to a feed, it can send arbitrary data objects to it. Musubi wraps the object with additional meta information as shown in Figure 7, signs it with the user's private key and sends it to the TGCP network.

The message contains the application's data and also includes a timestamp, an identifier for the sending application, and a sequence number indicating how many messages the user has sent to this feed, and the name of the feed. Objects are composed of properties that address various needs of application development. An object must contain a type (a short string identifier), and can optionally contain JSON-formatted data as well as raw binary. The size of a message is limited to 1 MB due to constraints on the devices. When an object is received, the timestamp, sending application's identifier, and user signature are also made available to an application.

timestamp	app_id	seq_id	feed_name	type	json	raw

Figure 7: A data object to be encrypted and inserted as the body of a TGCP packet.

6.4 Social Database

Objects entered into the social database are available to applications in controlled ways. In Android parlance, the Musubi social database is implemented as a Content Provider, allowing apps to run queries over the feeds and objects it maintains. The Content Provider also allows us to enforce access control, as we can determine the application that issued a query.

If an application is given access to a feed, it can obtain limited information about the users of that feed, including their public key, nickname, and profile picture. This is enough information to visually represent that user and also to establish communication with him via Musubi. More sensitive data such as phone numbers, email addresses, and physical addresses are by design not accessible. We believe a user should not have the ability to share this data on behalf of a friend, preferring instead for this data to be opted-in for sharing from that friend's device.

Currently, applications are restricted to seeing only data generated by their application from any member in the feed. We are actively exploring models that allow willing applications to exchange data.

6.5 SocialKit

Even with the friend management and connectivity provided by Musubi, writing distributed applications remains a challenge. We must also help developers manage distributed state, tackling transportation and state management details while providing a straight-forward, usable API.

To help develop applications over the distributed datastore, we provide APIs to manage common application styles in a library called SocialKit. For example, the *TurnBasedGame* API maintains consistency for applications in which only one user at a time is allowed to update the application's state. The API allows a developer to set a JSON object representing the application's state, to specify whose turn is next, and also to provide a text, HTML, or bitmap thumbnail representing the application for display in the main feed. This greatly simplified API can support a wide variety of applications.

The universal identifier given to the app instance's feed object can also be used to set up data connections outside of Musubi. We have demonstrated how this identifier can be used to run application sessions using the Junction API [7], which is bundled with SocialKit.

Junction allows applications to communicate in real-time as if they were in a chatroom. A unique session identifier acts as a capability for joining that session. Junction supports communication over a variety of channels, including XMPP, Bluetooth, and a local LAN. In each case, a device acts as a "switchboard," routing messages and establishing a global ordering. While objects in Musubi are implicitly persisted, Junction messages are transient. Junction also provides its own system for managing distributed state with its Props abstraction, which may be preferable to Musubi's APIs for some styles of applications.

The app instance identifier can also be used to maintain state on a central server. Musubi is still useful for establishing the application's membership and embedding it in a social feed.

7. EXPERIMENTAL RESULTS

7.1 Implementation

We have developed a prototype of Musubi on the Android platform and have made Musubi available in the Android Market since July of 2011. Our codebase is open source and available on GitHub [16].

Our experimental implementation of TGCP uses a stock distribution of the popular AMQP server RabbitMQ [23]. Each individual owns a message queue on the server named according to his public key. When a person sends a message, he creates a fanout exchange that multicasts messages to their intended recipients. Then, they send a message to the fanout, which causes it to be distributed to all of the recipients' message queues. The message queues are durable, meaning that the server guarantees message delivery once it accepts a message. A receiver acks messages from the server as it consumes them to ensure that each message is processed by the receiver at least once. The receiver itself keeps track of the signatures of messages it has received and fully processed to avoid repeatedly handling the same message. Our single server instance was able to route about 30,000 small messages per second. The architecture of the RabbitMQ server allows for it to scale up nearly linearly versus CPUs in a clustered configuration [22].

7.2 User Experience

Our research group has been using our prototype continuously for over half a year. On a larger scale, Musubi and our whiteboard application have been deployed at a charter school in New Jersey to 150 K-3 students. Teachers have responded favorably to the ability to share a private, collaborative drawing space with their students, using it for situations such as helping a student practice their cursive.

We have also conducted a study with two groups of 15 students each, aged 10-12, at a local Montessori school [30]. Students were all joined to one big group ahead of time to simulate the scenario where a student directory is made available by the school. Musubi was exceedingly popular with the children, so much so that parents were concerned about how to limit usage. The children had little trouble adapting to the system, making excessive use of picture taking and photo editing with PicSay, while remaining completely unaware of the underlying cryptographic operations. These students had mostly positive feedback, with a number of them proclaiming that "Musubi is AWESOME!"

7.3 Application Experience

The Musubi application provides a suite of functionalities including group management, app communication, and distributed state management. We describe below a sample of the different kinds of collaborative apps built for Musubi.

7.3.1 Interactive Sharing

Musubi makes it trivial to build interactive sharing applications among friends. **TadPoll** allows members to conduct quick polls. A user runs TadPoll in a feed and enters a question he would like people to answer. The question is rendered as an HTML thumbnail in the feed, and clicking it prompts users to respond to the poll. Each answer is persisted in the social database, and when a new answer is inserted, the HTML thumbnail updates to show current answers from the aggregated results. The basic TadPoll implementation is approximately 175 lines of code. In the same vein, **TodoBento** helps friends keep track of a common to-do list; friends can put up new items and check them off when completed. These apps will see more usage if they are frictionlessly integrated into the regular communication flow.

7.3.2 Turn-Based Apps

Many casual games are turn-based apps. Instead of requiring developers to manage scalable web services, Musubi lets all these apps be written as peer-to-peer software. **Tic-Tac-Toe** is implemented using SocialKit's TurnBasedGame API. Tic-Tac-Toe sessions are launched from within a feed, with Musubi prompting the user to select an opponent. Tic-Tac-Toe can be written in less than 200 lines of code, leveraging the TurnBasedGame API to maintain state and display players' names and images, as well as to represent the game's state as an HTML thumbnail in the encapsulating feed. Tic-Tac-Toe is particularly easy to write because its state is known in full to all players; other compelling games share this property, including Chess, Checkers, Connect 4, and many more.

weHold'em and **WordPlay** are two fancier turn-based games implemented using SocialKit. weHold'em is a game of Texas Hold'em poker for 2 to 8 players, and WordPlay is a Scrabble variant for up to 4 players. In weHold'em, each player joins with a fixed amount of money, and the game is played in rounds until a player's money runs out. WordPlay runs a single game per application instance. Both games publish HTML thumbnails to the feed to update their state.

Unlike Tic-Tac-Toe, both weHold'em and WordPlay in-

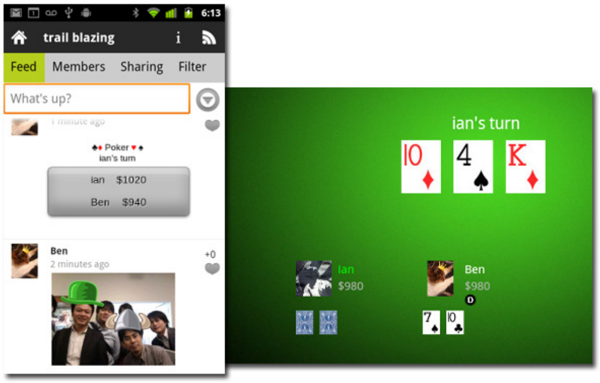


Figure 8: A game of Poker and its feed view.

volve application state that should be kept private to each player. We keep state private by simply not rendering it on devices that do not have access, however a curious user could determine other player's private state by exploring their local database. Because Musubi is designed for use with friends, this is not a grave concern. It is possible to run these games with cryptographically verified privacy [18].

7.3.3 *Real-Time Collaboration*

weTube and **wePaint** are two real-time applications we had previously built using Junction. WeTube is a collaborative playlisting application. A weTube session is started by invoking the application from a feed. Within the app, users can choose media from the web to add to a playlist, including YouTube videos or links to music tracks. Links can be voted up or down and are sorted based on their popularity. WePaint is a collaborative whiteboard app. Friends can draw with each other in real-time, and a thumbnail of the sketch is displayed in a Musubi feed. Junction's Props abstraction manages consistency of the distributed whiteboard state across devices.

Junction applications support ad-hoc groups by design, and Musubi provides the group context in which they may run. Integration took approximately 20 lines of code per app, allowing each to be shared directly from a feed. WePaint can also be launched atop an existing image in the feed, making itself available using the "edit" intent. Musubi passes the object's universal identifier as an argument to the editor, which allows wePaint to establish a unique, shared editing session.

7.3.4 *Making Existing Apps Social*

Jinzora is an independently developed media streaming application that connects to a personally-hosted cloud-based service to browse and stream a music collection. Jinzora integrates with Musubi in two ways. First, it can publish playback information to feeds as a form of presence sharing, letting friends know what you are currently listening to. Since Jinzora's music is hosted in the cloud, a friend can click on a music entry to play it back immediately.

Second, Jinzora can be launched as a feed app, sharing a session across devices. When Jinzora detects that it has been launched as such, it allows the client to be put in "jukebox" or "remote" mode. When in jukebox mode, Jinzora listens for messages to update the current playlist or to control

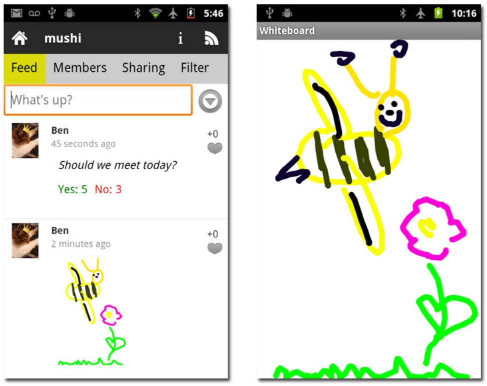


Figure 9: The TadPoll and wePaint applications.

playback, turning the device into a media renderer. Remote mode shares playback requests over Musubi rather than handling them on the local device.

The basic jukebox functionality took roughly an hour to implement, and required about 50 lines of code. Presence sharing required about 10 lines. Note that presence updates occur on a main feed, while jukeboxing occurs in a subfeed.

7.4 Discussion

We have shown that a large range of applications, from simple polling among friends to complete media browsing and streaming apps, can be integrated into the Musubi social experience seamlessly. We found the uniformity in the user interface to be very helpful because users know how to interact with new apps instinctively. Unlike other collaborative apps, these Musubi apps have no access to friends' contact information. And except for Jinzora, which streams from a media server, none of these apps have a server component. weHold'em and TodoBento are written by non-Musubi developers, who succeeded in building these apps without our involvement. This suggests that the API is usable and complete enough for at least these apps.

Our exploration of Musubi and TGCP highlight several challenges in deploying such a system in the real world. First and foremost, public keys are not widely associated with existing user accounts today. If we wish to import them from existing social networks, we must first persuade our friends to put up public keys in their profiles. Also, our technique of generating the keys on the device restricts us from extending TGCP messages to a user's proliferating number of devices.

Musubi focuses on the exchange of small data blobs up to 1 MB in size. Sharing larger content is out of the scope of TGCP, however we envision a companion service for sharing data such as HD photos and videos. This service would store encrypted copies of large data for only a short period of time (say, 1 week). Such short-lived data storage would keep deployment costs low while supporting common use-cases of mobile data consumption.

8. RELATED WORK

8.1 Groups as a Primitive

Group management in online social networks are a popular feature that enhances community interactions. The burden of getting group functionality tailored for efficacy in each and every social application is daunting, and the primitives need to become a part of a basic social operating system. When properly integrated into a system, group-based communication systems can increase usage by more than double [5]. Systems such as Cluestr, SocialFlows, and Facebook SmartLists all strive to eliminate the overhead of creating groups, yet they do not leverage the possibility of real-time acquisition of groups from mobile sensors nor do they provide a mechanism for ongoing private communication with the group members [9, 14, 28].

8.2 Alternative Social Application Platforms

A few companies, such as Skiller and OpenFeint, have attempted to provide game developers with a managed backend networking service so that developers need not worry about implementing and hosting a server. These companies, however, address this problem with a centralized solution and do not provide the same guarantees of privacy that encryption offers [20, 27].

Many systems have been built that try to enhance the ability of the individual to control the flow of information about them. Decentralization and encryption are the techniques that support preservation of data ownership and information flow control. Diaspora was a newsworthy example of a distributed social network because they were able to rapidly garner the interest and money of a crowd of people who were yearning for a safe social option [26]. There are a plethora of other alternative social network protocols in the open source community, each with a slightly different architecture [2, 31, 33]. SocialVPN suggests that the communication interface can be any IP based protocol and that members of groups should be connected via a virtualized subnet [12]. Distributed social communication systems, such as Contrail and MobiClique, are also designed with the mobile phone in mind, however, these systems do not address the difficulties the architectures present to distributed multi-party application developers [21, 29].

Google's Plus social network emphasizes the need for private correspondence by making users specify the precise groups they wish to share with, similar to how email works [8]. Although this network encourages users to consider the audience of their posts, it does not provide any guarantees about data ownership and privacy beyond what is outlined in the EULA. FlyByNight takes a different approach and uses existing centralized social networks to manage friend relationships, while storing only encrypted content on a central server [13]. Technically it provides disintermediation, but it requires all participants to join the same proprietary social network to establish friendships.

8.3 Identity and Routing

External identities can be authenticated in many ways ranging from standards like OpenID [25] or OAuth [10] to proprietary system such as Facebook Connect. Our system allows for real world identities to have a public key attached using these federated identity systems, but we do not require the Musubi identity to be bound to a “real world” account. This opens up the possibility to have pseudononymous identities, with the caveat that the structure of the social graph is still visible to the network operators. This is distinct from the anonymization provided by services like TOR [6] which obscure the endpoints of internet communication channels.

Because of the difficulties associated with P2P overlay routing schemes [4], the Musubi design depends on reliable traffic relay services. The reliable, but not fully trusted, provider model is more similar to existing sytems for electronic mail and instant messaging than P2P routing techniques [15, 35]. TGCP allows delivery of messages across a variety of network links because it is an identity-based routing system [3] with embeddable routing hints.

Authenticatr is a framework for establishing secured connections between friend on existing social networks [24]. After negotiating a shared key using the messaging primitives of existing social networks, it encodes future messages within the available platform-specific sharable types. It does not address the issue of data sharing between applications, but rather solves the problem of passing a secret message to a friend programmatically using existing channels. SocialKeys also proposes a mechanism for bootstrapping cryptographically secured identities through existing social networks by repurposing existing data sharing APIs [17].

9. CONCLUSION

This paper presents the Musubi social sharing platform which enables users to share and interact with friends on the phone without having to give up privacy to any third-party service providers. We have focused on creating a usable system, making sure the cryptographic operations used in enforcing privacy and security do not get in the way of the usability of the software. We have made the social experience very natural on the phone; we can interact with anybody on the phone's address book, without having to ask users to sign up on an external social network. Our informal user study involving about thirty children suggests that the software is attractive. We have made Musubi and a host of Musubi applications (Jinzora, wePaint, weHold'em, weTube) available in the Android Market, and all the source is open and available on GitHub. Videos showing some of Musubi's features are available on http://mobisocial.stanford.edu/musubi.

Disintermediation of phone interactions is accomplished with the design of the Trusted Group Communication Protocol (TGCP). The data are routed according to the public keys, thus enabling many possible relatively simple implementations. Making TGCP available through our application-level API, we enable easy development of many collaborative decentralized applications that honor privacy and without the need of hosting servers.

As we have demonstrated with the Musubi applications described in this paper, we now have the ability to create small and large decentralized social apps without having to worry about disclosing our friends' information to a third party. Even non-Musubi applications can share information privately through our system using Android intents. We envision one day that there will be hundreds of thousands of Musubi applications, and all of the native applications will share within Musubi feeds.

10. ACKNOWLEDGMENTS

We would like to thank Chanh Nguyen and Kazuya Yokoyama for their help in creating the weHold'em and TodoBento apps. This research was funded in part by NSF Programmable Open Mobile Internet (POMI) 2020 Expedition Grant 0832820, the Stanford MobiSocial Computing Laboratory, and a Stanford graduate student fellowship.

11. REFERENCES

[1] D. Boyd, E. Hargittai, S. J., and J. Palfrey. Why parents help their children lie to Facebook about age: Unintended consequences of the "Children's Online Privacy Protection Act", 2011. http://www.uic.edu /htbin/cgiwrap/bin/ojs/index.php/fm/article/ view/3850/3075.

[2] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta. PeerSoN: P2P Social Networking: Early Experiences and Insights. In *SNS '09: Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, pages 46–52, New York, NY, USA, 2009. ACM.

[3] M. Caesar. *Identity-based routing.* PhD thesis, University of California at Berkeley, 2007.

[4] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. *ACM SIGOPS Operating Systems Review*, 36(SI):299–314, 2002.

[5] S. Counts. Group-based mobile messaging in support of the social side of leisure. *Computer Supported Cooperative Work (CSCW)*, 2007.

[6] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th conference on USENIX Security Symposium-Volume 13*, pages 21–21. USENIX Association, 2004.

[7] B. Dodson. Junction: A Platform for Mobile, Ad-Hoc, Multi-Party Application Development. http://openjunction.org.

[8] Introducing the Google+ project: Real-life sharing, rethought for the web. http://googleblog.blogspot.com/2011/06/introducing-google-project-real-life.html.

[9] R. Grob, M. Kuhn, R. Wattenhofer, and M. Wirz. Cluestr: Mobile Social Networking for Enhanced Group Communication. In *Proceedings of the ACM 2009 International Conference on Supporting Group Work*, pages 81–90. ACM, 2009.

[10] E. Hammer-Lahav. The OAuth 1.0 protocol, 2010.

[11] J. Jonsson and B. Kaliski. Public-key cryptography standards (pkcs)# 1: Rsa cryptography specifications version 2.1. Technical report, RFC 3447, February, 2003.

[12] P. Juste, D. Wolinsky, P. Oscar Boykin, M. Covington, and R. Figueiredo. Socialvpn: Enabling wide-area collaboration with integrated social and overlay networks. *Computer Networks*, 54(12):1926–1938, 2010.

[13] M. Lucas and N. Borisov. flybynight: Mitigating the Privacy Risks of Social Networking. In *Proceedings of the 7th ACM Workshop on Privacy in the Electronic Society*, pages 1–8. ACM, 2008.

[14] D. MacLean, S. Hangal, S. Teh, M. Lam, and J. Heer. Groups Without Tears: Mining Social Topologies from Email. In *Proceedings of the 15th International Conference on Intelligent User Interfaces*, pages 83–92. ACM, 2011.

[15] S. Marti, P. Ganesan, and H. Garcia-Molina. Sprout: P2p routing with social networks. In *Current Trends in Database Technology-EDBT 2004 Workshops*, pages 511–512. Springer, 2005.

[16] MobiSocial. Musubi Source Code. https://github.com/Mobisocial/dungbeetle.

[17] A. Narayanan. SocialKeys: Transparent Cryptography via Key Distribution over Social Networks. *The IAB Workshop on Internet Privacy*, 2010.

[18] C. A. Neff. Verifiable mixing (shuffling) of elgamal pairs. Technical report, In proceedings of Privacy Enhancing Technologies '03 (PET), LNCS series, 2003.

[19] J. Nurminen. Parallel connections and their effect on the battery consumption of a mobile phone. In *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*, pages 1 –5, Jan. 2010.

[20] OpenFeint. OpenFeint. http://openfeint.com/.

[21] A.-K. Pietiläinen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot. MobiClique: Middleware for Mobile Social Networking. In *Proceedings of the 2nd ACM Workshop on Online Social Networks*, WOSN '09, pages 49–54, New York, NY, USA, 2009. ACM.

[22] RabbitMQ: FAQ, 2011. http://www.rabbitmq.com/ faq.html#clustering-scalability.

[23] RabbitQ. http://www.rabbitmq.com/.

[24] A. Ramachandran and N. Feamster. Authenticated out-of-band communication over social links. In *Proceedings of the first workshop on Online social networks*, pages 61–66. ACM, 2008.

[25] D. Recordon and D. Reed. Openid 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*, pages 11–16. ACM, 2006.

[26] R. Singel. Open Facebook Alternatives Gain Momentum, $115K. http://www.wired.com epicenter/2010/05/facebook-open-alternative/.

[27] Skiller. Skiller SDK. http://www.skiller-games.com/.

[28] The Facebook Blog: Improved Friend Lists. http://www.facebook.com/ blog.php?post=10150278932602131.

[29] P. Stuedi, I. Mohomed, M. Balakrishnan, V. Ramasubramanian, T. Wobber, D. Terry, and Z. Mao. Contrail: Enabling Decentralized Social Networks on Smartphones. Technical report, Tech. Rep. MSR-TR-2010-132, Microsoft Research, 2010.

[30] I. Vo, T. Purtell, B. Dodson, A. Cannon, and M. S. Lam. Musubi: A mobile privacy-honoring social network, 2011. http://mobisocial.stanford.edu/papers/musubi.pdf.

[31] The Appleseed Project, 2010. http://opensource.appleseedproject.org/.

[32] A. Whitten and J. Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *Proceedings of the 8th USENIX Security Symposium*, pages 169–184, 1999.

[33] OneSocialWeb, 2010. http://onesocialweb.org/.

[34] WebFinger, 2010. http://code.google.com/p/webfinger/.

[35] Z. Xu, R. Min, and Y. Hu. Hieras: a dht based hierarchical p2p routing algorithm. In *Parallel Processing, 2003. Proceedings. 2003 International Conference on*, pages 187 –194, oct. 2003.