A Mobile Social Network on ESP: an Egocentric Social Platform

T. J. Purtell Ian Vo Monica S. Lam

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

Abstract. Social networks today are social intranets: the entire social graph and all the information communicated is accessible to and monetizable by the social network provider. The trend is for a monopoly to emerge that owns the world's social graph, which not only puts user privacy at risk, but threatens competition and innovation as well.
This paper proposes an open system called the Egocentric Social Platform (ESP), leveraging on Identity-Based Cryptography (IBC), that enables users to easily communicate securely on their mobile phones with their existing identities using their local address book as a sliver of the distributed social graph. This system is integrated into the Musubi social app platform that enables users to share text and photos, play games, all without a central third-party intermediating all communication.

1 Introduction

The prevalent way to create social applications today is to use Facebook connect, which provides apps with an instant social network of almost a billion users. Apps can write to friends' walls to help make the apps viral; apps can invite friends to join in multi-player games. This service comes with a nontrivial price tag, however, as Facebook demands 30% of the revenues of all purchases through Facebook apps. A case in point is that Zynga reported a loss of 95% of their profit in one quarter partially as a result [21]. As Facebook owns by far the largest social graph, app developers have few options. Even though there are other smaller social networks that consumers can participate in, all such networks today intermediate and monetize all the communication and interactions, which is a huge invasion of user privacy.

This paper proposes an open distributed social network standard we call ESP (Egocentric Social Platform). To demonstrate its feasibility, we have ported Musubi, a fully functional mobile social app platform with numerous games and utilities, to ESP. We are at this point preparing a new release of the Musubi app using ESP to the Android market by April.

1.1 Egocentric Social Platforms

To make a phone call or text-message a friend, we only need to know a friend's phone number–we don't need to first ask our friends to join the same proprietary

network or sign away rights to all of our communication. Even the government requires a warrant, before they are allowed to wiretap our phone conversations. Imagine a world where it is just as private *and easy* for groups of friends to communicate real-time in feeds and play social games with each other. To that end, our goal is to create a platform that enables hundreds of thousands of developers to create apps as easily as and with as wide an adoption (eventually) as Facebook.

This paper focuses on creating a platform for mobile apps where the primary use of Facebook is to get access to one's friends. To create the best ecosystem to attract as many users and partners as possible, we wish to have a solution that guarantees privacy technologically rather than contractually. Our design is founded on the fundamental principle that

1. the social graph is distributed and represented collectively in the users' address books, and
2. data are not stored on any central servers; they are accessible only on the participants' mobile devices, and data are encrypted on transit.

Replacing cloud-based social networks with a distributed and encryption-based design is a daunting task. The vast repository of data that these networks collect centrally are used to provide a myriad of functions managing and distributing users' profiles and data, discovering friends, eliminating spam, and authenticating communication. Distributed systems and encryption are notoriously hard to use. In addition, how do we compete given the size of the centralized social networks that already exist, especially since many people do not care about privacy?

One critical factor is how we can acquire a (decentralized) social graph quickly. Our solution is to leverage the existing contacts in our address book, letting people use their *existing identities* to interact with their friends in their *existing* address book with encrypted messages. There needs to be no sign up and no key exchange a priori.

A promising approach is to add Identity-Based Cryptography (IBC) and encrypted messaging to the services offered by existing identity providers, such as email accounts and SMS phone numbers. There is a healthy competition of free identity providers (Yahoo, Gmail, HotMail, Facebook, Twitter, OpenID) and every corporation and university has an email system. Users have multiple personas, allowing to keep separate their corporate and school identity from one or more personal identities. If users so wish, they can even stand up their own identity service, and thus not be beholden to any one.

With IBC, users can use encrypted communication without requiring the exchange of public keys, and it enables revocation in case the key is lost. However, not only do the major identity providers have little incentive to provide such services, it would take a long time to create and deploy such a standard. Our approach is to build a social app platform that is built upon a standalone IBC and encrypted messaging system that can be used with any existing identity.

1.2 Contributions

This paper proposes ESP, an Egocentric Social Platform, where the social graph and its contents are distributed and owned by the users themselves. This system enables the use of any existing identity, including proprietary ones such as Facebook accounts. Our design does not require the cooperation of existing identity providers; it is lightweight and practical, making it possible for us to create a fully functional system. In this way, we can experiment and pave the way to creating an open standard that allows identity providers to participate if they wish. The contributions of this paper include the following:

The ESP API. The ESP API (Application Programming Interface) provides a concise yet comprehensive collection of functions on which distributed social apps can be easily built. The system is carefully designed to balance privacy preservation with ease-of-use. The design takes advantage of the social graph embedded in the native address book, makes the process of onboarding as frictionless as possible, and deters spamming. For privacy, all data sent through the API is only readable to the specifically noted contacts.

Encrypted Messaging with Existing Identities. We propose creating a global, standalone IBC service that provides identity specific keys to an individual user whenever they prove their ownership of an existing account. The service itself can be distributed, implementing a threshold scheme so no one server has access to users' private keys. As phones cannot talk directly, we introduce an IBR (Identity-Based Router) service. Messages are encrypted using the recipients' hashed identity's as public keys and sent to the IBR; recipients authenticate themselves, retrieve the messages and decrypt the messages using the private key.

A Complete Mobile Social Network. To validate our design, we have ported the Musubi mobile social network [4] to the ESP platform. Musubi was also based on encrypted messaging between mobile devices. However, it did not work with existing identities; a device generated its own public-private key pair, and the users shared public keys out-of-band via emails or NFC. Unlike ESP, it was not possible to access the messages from different devices and loss of a key would stop the user from being able to communicate with all contacts. ESP greatly improves the adoptability of Musubi by leveraging existing identities and relations.

Performance and Feasibility of ESP. This provides an analysis as well as microbenchmarking results to show that our proposed system is feasible. The Musubi app with ESP is efficient on mobile devices, and the ESP system is easily scalable.

1.3 Paper Organization

The rest of the paper is organized as follows. Section 2 shows how users of the Musubi app built on top of ESP can quickly start interacting with friends in our existing phone book. We present the overall design and the rationale behind the API in Section 3. Sections 4 and 5 describe two underlying services in the

platform: the IBC private key service and the routing service. We then describe our prototype and performance results in Section 6. Sections 7 and 8 present related work and conclude.

2 Frictionless Interaction with Existing Identities

Social apps all face the same bootstrapping hurdles: engaging users to download the apps, encouraging viral distribution, initiating friend interactions, and getting apps to share information. ESP provides social apps a set of functions addressing these hurdles, without sacrificing users' private information, and without any server infrastructure. The key to adoption is to make onboarding frictionless, which is complicated by the encryption and distributed design involved.

2.1 The Musubi App Platform

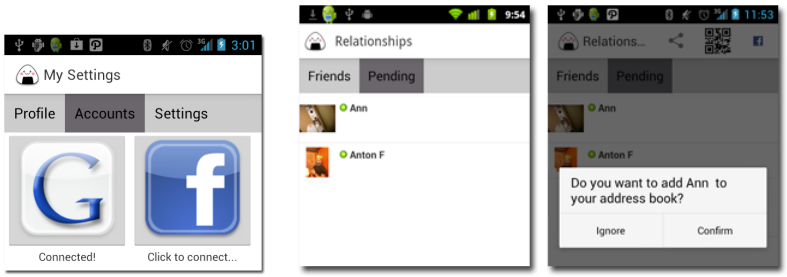
ESP is designed to support a variety of social apps, such as a common calendar for family members or a social browser that allows users to share favorite links, all without disclosing the information to a third party. Instead of building a brand new social app, we have chosen to port Musubi to ESP as a demonstration.

Musubi is attractive because it itself is a *privacy-conscious social app platform*. It provides the abstraction of *disintermediated social feeds*, that allow users to share status, photos, voice messages, and arbitrary data types. Apps built on top of Musubi can communicate via the feed, thus individuals' identities are hidden from the apps themselves. A large number of apps have been built, including games like Tic-Tac-Toe, a Scrabble variant, Texas Hold'em poker, a real-time group finger-painting app, as well as utility functions such as common to-do lists and event invitations. By sharing apps over the feeds, Musubi provides an important function of helping users discover apps their friends like.

While Musubi also used encryption to implement disintermediated communication, it was not integrated with existing networks, requiring users to exchange keys a priori to interact. This lack of an initial user base was a major deterrence to Musubi's adoption. Building on top of ESP, Musubi can leverage existing identities and greatly improves its virality. No extra exchange action is needed on the user's part to communicate with a friend they already know.

2.2 Musubi User Experience as Enabled by ESP

Let us first illustrate how users get started with Musubi on ESP using a scenario. Ann reads about Musubi on some blog that it stores no user information in the cloud and decides to try it out. When she launches Musubi, it invites Ann to establish an identity (suggesting that she might already have messages waiting for her). She decides to link her Google account (although she has the option to link to other accounts such as Facebook, as shown in Figure 1(a)), and is directed to Google's page to authenticate. Her Google username is pre-populated by default from her phone.



(a) Linking ESP to an existing identity.

(b) Approving pending friendships.

Fig. 1. Managing accounts and the address book.

She wants to try Musubi out by playing a game with her friend Bob (Figure 2). She starts typing Bob's name into the recipients field, and sees Bob's address book entry auto-suggested to her. As she starts a game of WordPlay with Bob, Musubi notices that Bob is not yet a Musubi user and suggests sending Bob an email in addition to tell him about Musubi. Musubi shows Ann a composed letter that she can send to Bob with just one click.

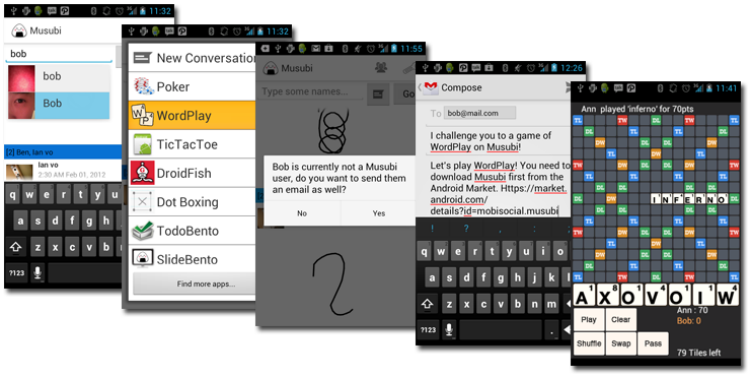


Fig. 2. Ann starting up a game of WordPlay with Bob

Ann's message contains a link that takes Bob to the Musubi app on the Android market. After Bob launches Musubi and signs into his identity, Ann's invitation is waiting for him (Figure 3). Note that if Ann is not already a friend and therefore not in Bob's address book, Musubi will inform Bob that there are messages pending from Ann and will present the messages if Bob acknowledges Ann as a contact (see Figure. 1(b)). This step is useful as a deterrent for spammers to reach Bob on Musubi.

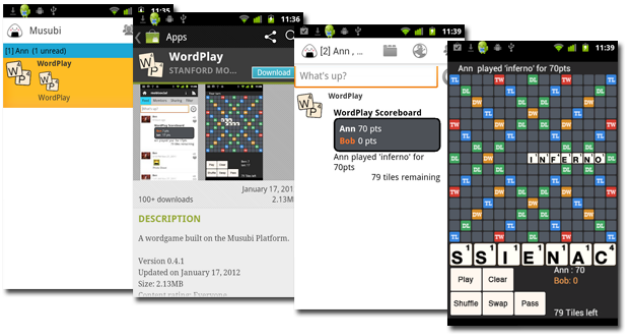


Fig. 3. Bob opening Musubi for the first time with Ann's game invitation already waiting for him. Clicking on the invitation brings him first to the Android Market.

Bob clicks on the invitation and is brought to the WordPlay app in the Android market. He installs the game, sees that Ann has already made her move, and starts working on his word from the letters he was dealt. Bob is pleasantly surprised that the free WordPlay game has no ads, and is highly responsive, unlike other word games he has played that require a constant connection and both players to be online. Ann, who is more privacy conscious, is particularly happy about the fact that WordPlay does not know anything about Bob even though they are playing together.

This example illustrates that the work flow is simple – the users do not see any details regarding key management or encryption. From start to finish, it only takes a few minutes before friends are playing a new game together. Games like WordPlay have no central server and instead use the underlying friends messaging primitives of ESP. That's why it can be provided free of charge, with no ads, and no loss of privacy at all.

3 Design Rationale of the ESP API

This section describes the overall architecture of ESP, an overview of the API followed by a discussion of the rationale.

3.1 Overall Architecture

A high-level architecture of ESP is shown in Figure 4. ESP consists of a service running on the device, which helps connect users to their existing identities and social circles. Once this is done, the friends and groups information are made accessible to all apps built upon this platform. ESP is integrated into the native address book, which is available to all apps. ESP apps use the native address interface to pick the names in the contact book and invoke functions from the ESP API for communication. Each application has an isolated message namespace which allows ESP apps to keep data private from other ESP apps.

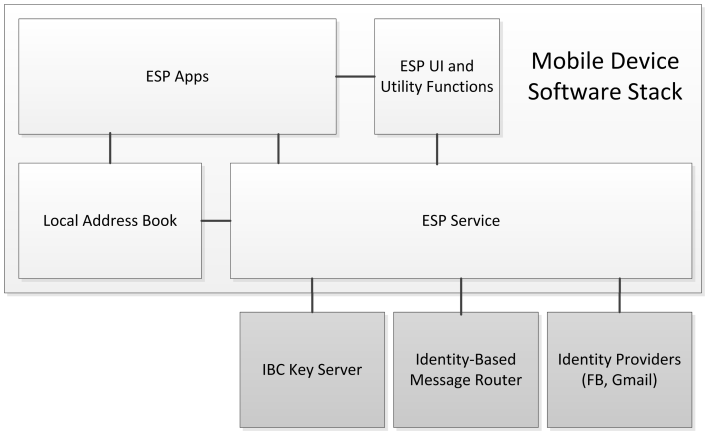


Fig. 4. Architecture of ESP

To provide encrypted messaging with existing identities, the ESP service on the device interacts with two new services in the cloud: the *IBC Private Key Service* (IBC) and the *Identity-based Message Router* (IMR). The IBC service provides the secret key for an identity once the individual proves he is the holder of the account. The IMR service enables mobile devices to communicate securely using their hashed identities–messages are delivered in real time if possible and queued when necessary. Details about these services are provided in Sections 4 and 5, respectively.

3.2 Overview of the ESP API

We now provide a summary of what ESP provides to its applications so apps can easily provide the experience as described in Section 2.

Identity Management.

- Ask for the identities of the user. The ESP service is responsible for the dialog to determine the user's identities. If the user has not registered any identities, the ESP service will provide a UI and prompt him to connect to an existing identity. If identities have already been registered, then it asks the user which identity they want to use in the context of the application. This selection is stored and future queries for the owned identities by an application will not cause a user interface to be displayed. The only exception arises if ESP detects that the user's credential has been revoked.
- Import common social networks, such as Facebook and Gmail, into the address book, upon user's approval. This is a helper function, not a core part of ESP, provided as many apps could benefit from it, in case users have not already populated their address book.

Group and Contact Management.

– Choose from contacts that the user has identified for use with a specific personal identity. Depending on the method the user went through to populate their address book, the native address book may not have a logical separation between the contacts for the users' different identities. This mechanism is provided to allow applications to support multiple personas.
– Trigger a user interface to manipulate the groups declared to ESP.
– Show a dialog to allow a user to block messages from a specific identity.

Social Graph Query.

– Check if a given identity is an ESP user.
– Check if a given identity has used the app.

Messaging.

– Send a message to a list of friends
– Receive a notification if a message is received from people in the address book.
– Receive a notification of messages pending from people not in the address book.
– Reject or accept a person into the local address book. Messages from rejected people will be deleted automatically.
– Contact a user out of band with a given message. This is a simple helper routine provided to perform a common function.

3.3 Identity Management

Today, as many users have already invested a significant amount of time creating online social networks, it is critical that we leverage this investment. As a device-centric design, we have the opportunity to aggregate all our connections in one place and engage in disintermediated communication, without concerns of loss of privacy or confidentiality. While one network may have objections to its connections being imported to another, a user should have the right to import all the relationships into their own device for their own personal interaction. We can consolidate our contacts into one place and use the same tools for our corporate and personal relationships, for confidential interactions related to financial or health matters, for sharing among friends and family, and for frivolous social interactions. An ESP app can easily be isolated to interacting with a specific persona. Users may have a default persona to be used for an application, unless he overrides it.

Having a single service that all apps can use to establish credentials improves the protection of users' identities. First, the user will not be inundated with requests to log in by different apps. Second, as will be described in Section 4, careful consideration is taken to safeguard one's privacy. Finally, the ESP software is available as open-source to facilitate user trust in the service.

While ESP is designed to help users with existing relationships, it does not preclude users who want to have no connections with existing social networks. Such users could benefit from the privacy-friendly design of ESP. They can

simply create a pseudonym account with an identity provider such as Gmail and use that exclusively in ESP interactions. As long as his friends all do the same, nobody will be able to connect their activities with any identifiable individuals.

Platforms based on encrypted communication must also allow law enforcement entities to gain access to the communication in case criminal activities are suspected. In our case, law enforcement can get the identity credentials from the identity providers.

3.4 Social Graph Management

Currently, ESP only provides a small set of core functions to help manage the egocentric social graph and to aid with onboarding. This functionality is likely to grow in the future to provide higher-level social functions.

Deterrence of Spam. As an open messaging system, ESP allows anyone to message anyone using the system. The obvious downside is that this platform could become another vector for spam, like email today. While we allow users to message without first requesting to be friends, we do not display any of the messages without the receiver's approval. In other words, only messages from an approved list (the whitelist) are displayed. To keep user involvement minimal, everybody in the current address book is considered to be on the whitelist. Messages from unknown identities are withheld and such identities are placed on a gray list. The first of the messages from the same unknown identity is decrypted and the ESP app receiving the message will be notified of the individual's identity. If the user approves the individual, he joins the white list and his contact information is entered into the address book. Anybody denied will be placed on the black list, and their messages are summarily discarded.

Apps like Musubi which are designed for small groups of close friends can include a low-key UI element so that the user can deal with them at their leisure, rather than accenting it to push users into accepting requests more than they would otherwise.

We expect spam not to be an issue in ESP. First, all messages are authenticated, thus eliminating impersonation. Second, we do not show any messages from anyone not in the address book. If there is a substantial amount of undesirable message traffic, our design makes it possible to offload this function to the routing service. As will be explained in Section 5, identities in message headers are not the actual identities but a hashed version, thus the exposure of private information is limited.

Reducing the Friction in Onboarding. Until ESP becomes as widespread as SMS on the mobile devices, we cannot assume that our friends are using ESP. It is thus desirable to keep track of friends who are ESP users so that we can send out-of-band messages to the non-users to tell them about ESP.

ESP keeps track of its users in a distributed fashion, all without a central server. When a new user installs an ESP app, the application can automatically

send each friend in the address book a state change message indicating that the user is now an ESP user. Notice that this is just an internal message not visible to the end user. The ESP service keeps track of whether or not each contact has ever sent a message using ESP. It also keeps track of what applications triggered the tranmission of messages so apps can detect if they are installed on friends' devices. If the user is not an ESP user (yet), the information will sit in the friend's queue until it is delivered when the friend installs an ESP app. Once ESP captures this information from the users' message queue, it stores it in a local SQL database.

We expect that many more distributed graph properties will be made available in the future. The key takeaway is that the underlying platform must provide the ability to make state changes efficiently and securely. We need to send a single encrypted message to all our friends, informing them of the state change, without them learning the identities of all our friends. We refer to this messaging mode as *blind*, as in blind carbon copy.

3.5 ESP App Communication

To provide apps with different namespaces, the ESP message format is defined such that the header includes the hash of the local application identifier. Because the uniqueness of application namespace is enforced by the platform market, message data can be isolated into separate application namespaces while reusing the same identity. Additionally, this application identifier can be used for network traffic control.

Applications can choose to send messages to a group with either an open or blind recipients' list. Furthermore, an application can request ESP to deliver all application messages to it in the order they were sent by devices or it can ask for the messages to be delivered as they arrive. This allows more sophisticated applications to deal with potential out of order message delivery without latency overhead. An application can be registered to listen for new events. Because the application ID is part of the message format, it is also possible for the ESP daemon to launch the already installed application to whom the message belongs.

4 IBC with Existing Identities

We standardize around IBC thus allowing users to send encrypted messages to one another without requiring the exchange of public keys. IBC is concept proposed Shamir where the public key for an individual is their human readable identifier [17]. A server hands our private keys to users that are mathematically derived from their identifier string. A method for performing these techniques using the Weil pairing was later discovered by Boneh and Franklin [2]. IBC also enables a signature scheme where peer-to-peer messages can be validated without contacting the private key service [9]. IBC affords us the frictionless experience where a user can start playing games using encrypted communication even before her friend has downloaded any ESP software. IBC also supports revocation as the keys implicitly expire periodically. In our case, we have chosen a phased

refresh granularity of one month.

In this design, the authority to read messages sent to an identity is granted by the original identity provider. We use an intermediate IBC secret server to provide delegated authentication across a variety of sources of real world identity. Ideally, each identity provider also provides an IBC secret service. But until that happens, our system runs a default IBC service that can be used with any existing identity. To avoid giving up secrets to the global IBC service, we can adopt a threshold cryptography scheme so no single IBC server knows any complete user secret [6]. Here we focus on the relationship between the identity provider and the IBC service.

4.1 Private Key Distribution

When a client wants to claim an identity that can be verified using a public authentication protocol such as Facebook auth or OAuth, the client communicates first with that service to get the appropriate token. Then it submits that token to the IBC server along with a requested key timestamp. After checking that the OAuth token is valid and that the request timestamp for the key is legal to request, it sends the secret to the client.

If a client wants to claim an identity that can receive messages but lacks any well-known delegated authentication scheme, the secret request process proceeds as follows. The ESP service submits the publicly reachable address to the identity secret servers along with a one time use symmetric key. The server sends a message, email, SMS, etc, as appropriate, to that identity with the requested partial user secret encrypted with the specified one time key. ESP locally uses the users credentials to access the contents of the IBC secret messages (e.g. read from phone SMS database or download from email server). It then decrypts the secrets and stores them.

4.2 Key Updates

The design of IBC is that all secrets are associated with a validity time period. This implicit refresh requirement is designed so as to prevent the use of a stolen key beyond that period. Our IBC keys have an issuance time associated with them that has a granularity of seconds. Signatures with and IBC key are considered valid only for up to 30 days from the issuance time. Therefore every 30 days, the ESP service will contact the IBE server for a new private key. On the encryption side, messages are encoded using the last regularly scheduled IBC secret update time. The exact date of the key expiration is a well-known function of the hashed identity that causes all clients to request key updates in an evenly distributed traffic pattern.

4.3 Revocation

Revocation may be necessary under a variety of conditions. In a corporate setting, a company may revoke access by an employee upon termination. This does

not apply to free accounts such as Gmail or Facebook. For personal group socialization, revocation is necessary when a user loses his device or if malware on the device has stolen the user's credentials or the secret key. Our system supports revocation at several levels, while not inconveniencing the user unnecessarily.

First, our system requires the user maintain a valid credential to his identity. Requiring users to sign in every time would not be acceptable. Instead, we cache the credential to his identity provider, and refetch a new one whenever it expires (if it is supported). In the meantime, the credential is checked periodically (every 24 hours) to ensure that it has not been revoked. Identity providers like Google and Facebook allow users to remotely revoke the access token for a specific application or device . If the credential is revoked, we also delete the private key.

If a device or key is suspected to be stolen, our system also makes it possible for a user to request a new key before the standard expiration time. Once the user sends a message to his contacts using this newer key, the contacts will no longer accept messages signed with any older key. This allows for users who are already known to each other to revoke keys with a much shorter latency. When a user reports to the client that he need to revoke his key, it will send a message to all known individuals to ensure that the defunct key is no longer accent.

A final level of revocation can extend low-latency revocation to new contacts. When a user revokes a key, he can sign his new minimum allowable key time and hashed identity using his replacement key. This bundle can be stored at a public location and checked by the ESP before it sends a message to a previously unknown identity. This requires storage of at most a single small revocation record per user.

5 Hashed Identity-Based Router (IBR)

Once clients have claimed ownership of an identity they can send and receive messages to each other. Rather than relying on typical internet routing which fails to allow incoming connections on mobile phones, we provide the Identity-Based Router (IBR) service. On most non-WiFi wireless networks deployed today, network address translation and/or firewall restrictions block a mobile device from receiving arbitrary incoming connections. This excludes a design where messages are sent directly between network participants, at least if the system is meant to be usable for the average Joe. Additionally, devices often temporarily lose network access or run out of battery power.

ESP apps post communications to this routing service, which then buffers the messages to their peers. In order to hide the real identities of the users, the service relies on hashed-identities as the routing key. In this way, the IBC secret service never needs to see any user data, and the message routing system never needs to see the user keys. This allows for strong separation of services which protects users' private data.

Once the ESP service has proven the user credential to the IBC secret server, it acquires a signature key and an encryption key, both tied to a specific expiration time. The client authenticates with the message routing service via challenge response using this secret. Authenticating allows ESP to download the messages

queued for the user and allows ESP to send new messages originating from that user. When the routing service sees a newer signature used for authentication, it records the timestamp of that key and enforces that no future connection attempt succeeds without proof of a signature key that is at least that new.

5.1 Message Format Design

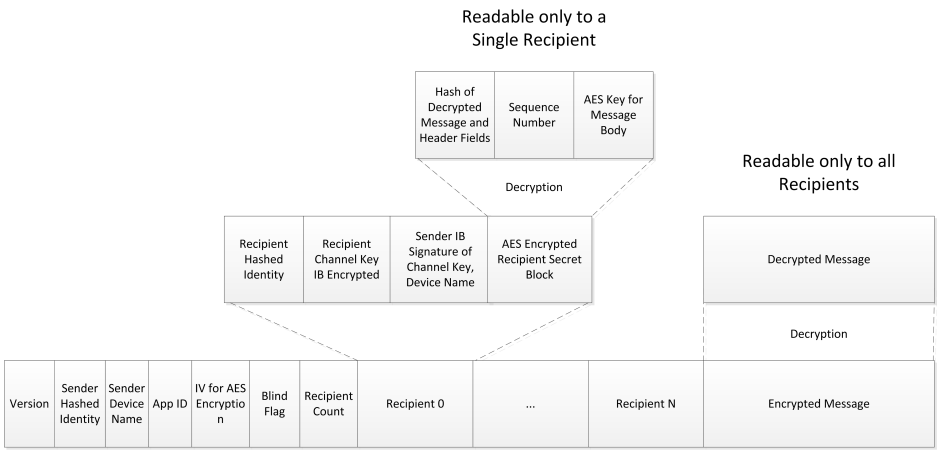


Fig. 5. Encrypted Message Format

Our message format is designed to protect the real identity of the user from direct exposure and to maximize performance on mobile devices. ESP messages are similar to emails. They have a single sender and multiple recipients. The sender is identified by an authority type flag (e.g. email, SMS) and the hash of the identity name (e.g. SHA256("foo@example.com")).

Multi-device support. The sender also identifies the device from which they send. The device name is simply a random 64-bit integer that is used to discriminate between messages that come from different devices. Because there is only a single user secret per time period when using IBC, the message format must allow for devices to under the same identity to be discernable. This allows for proper implementation of higher level primitives needed to implement distributed protocols based on the devices participating in an interaction.

Separate app namespaces. An application identifier is also included in the message header. This identifier allows for the ESP service on a mobile device to route messages to specific applications on the same device using the system.

Efficient encryption. Each pair of participants shares a pair of unidirectional channel secrets, with each derived from the sender's private key and the receiver's public key. The channel secret has to be recomputed whenever either partys' key expires. Assuming that the participants' keys expire at different

times, the secrets are recomputed twice a month, which is the twice the frequency of the IBC private key updates. One channel secret is used to encrypt the shared message key and authenticity information for each recipient.

The unidirectional channel secret concatenated with the device identifier is signed using the IBC signature secret, allowing the contents of the personal secret block and the message headers to be checked for authenticity. The encrypted and signed channel secret can be reused as long as the IBC expiration time has not been reached for the sender or the recipient. Because IBC operations are time consuming particularly on mobile devices, caching this secret significantly limits the CPU overhead of communicating using ESP when there is substantial communication between users.

Recipients can use the channel secret to decode the AES key, which is used to encrypt the bulk of the per-recipient encrypted block. The secret block contains the absolute sequence number of the message sent to this recipient from the sender's device and a SHA256 hash of the decrypted main message body and message header. The hash is use to verify that the message was not tampered with in transit while the sequence number allows the ESP services to retransmit potentially lost messages.

The main body of the message is encrypted with a single key which is randomly generated for each message. An initialization vector (IV) is included in the message header to facilitate secure AES encryption in the message encoding process.

Protecting the recipients' list A "blind" flag on the message can be set inidicating that it is a widely distributed state update message. When the message router receives a message with this flag, it will trim the recipient list to just a single identity before delivering the data.

5.2 Privacy

Hashed identities are sufficient information for the router to do its work, so the router need not discover the real identities of its clients. The router could potentially track the groups of hashed identities which communicate with one another as well as the volume of communication amongst them. Enough information is available to the message routing system to be able to construct an anonymized global social graph. Combining this with a dictionary attack based on known email address could result in the routing system having at least a partial understanding of which real world identities relate to each other. As discussed in Section 3, users and their friends can choose to create new pseudonyms not used any where else to preserve their anonymity. Regardless, the message router is absolutely not able to access the contents of the communication between clients.

6 Experiment

6.1 Prototype Implementation

Our prototype of ESP is built on the Android platform. The account management and other user interfaces are invoked by sending Android intents to the ESP

service. Message are sent and received by interacting with the local database. The ESP service watches the database for changes and encodes and transmits new messages added to the database. When a message is received from the IMR, ESP inserts it into the database and notifies applications that new content is available.

The IBC encryption and signature primitives are built using an open-source library, called the Pairing Based-Crypto Library [12]. The implementation for the encryption is native code, while the ESP service is implemented as Java code running on the Dalvik VM. The crypto primitives are about 1k lines of code and the ESP service itself is about 9k lines of code. We use the BSON format to encode our messages and secret blocks through the Jackson JSON library combined with a plugin called bson4jackson.

The IBC key server is implemented using a Python server that calls the IBC primitive library. We did not implement a distributed IBC threshold scheme. Our timing measurements only consider the cost for the single server implementation.

Finally, the message router is built using the AMQP protocol. This flexible messaging system allows clients to define routes handled on a remote cluster [1]. It provides an abstraction that allows for the definition of message queues and routing of message through exchanges to multiple queues. Each identity in our system has an exchange on the server. Each device that connects to the server attaches a private message queue to the identity exchange, thus each device sees all messages sent to the identity. When a user sends messages, they also address the messages to themselves to ensure they are replicated across devices. An initial identity queue is created for a user if they have not added themselves to the system. When the user first connects, he drains that queue and detaches it from his identity exchange. The list of recipients a message is being sent to is temporarily cached on the server to lower the overhead of sending a stream of messages to a larger group.

Musubi had to be modified significantly to adapt it to ESP. Predominantly, these changes consisted of removing the unnecessary key-exchange and storage mechanisms from the code. The data model for Musubi also needed to be updated so that it interacts with the local address book. The entire set of API functionality provided by the Musubi social platform, SocialKit, remains the same. Minor modifications were required the global unique identifier generation code.

6.2 ESP Service Performance on Mobile Devices

One major concern of using encryption on mobile devices is the potentially significant CPU usage impact. Identity-based cryptography requires big number exponentiations and difficult elliptic curve pairing procedures. To ensure privacy, each message requires an encrypted block per recipient in addition to the encrypted message body. This brings into question if IBC can be used for social messaging, especially when considering the possibility of high frequency in communication between a large number of contacts. We use our implementation to provide estimates on the performance of the ESP system.

Microbenchmarks. We first measure the cost of the individual operations. The numbers are collected on a 1GHz ARM processor. Figure 6 shows a breakdown of the cost of receiving and sending messages on the mobile device, assuming the following parameters:

m_{S}: Number of messages sent.
m_{R}: Number of messages received.
r: Number of recipients.
s: Number of senders.
t: IBC expiration window size, which is 1 month in this design.
l: The length of the message (in kilobytes).

(a) Receiving messages.

Operation	CPU Time (ms)	Frequency
Compute channel key	78	$2r/t$
Sign encrypted channel key	340	$2r/t$
Load cached channel key	0.58	$m_{\mathrm{S}} \times r$
SHA256 of message headers	$0.0067 \times r$	m_{S}
SHA256 of message body	$0.026 \times l$	m_{S}
AES encrypt secret block	0.78	$m_{\mathrm{S}} \times r$
AES encrypt message body	$0.42 \times l$	m_{S}

(b) Sending messages.

Operation	CPU Time (ms)	Frequency
Check user signature	590	$2s/t$
Decrypt channel key	522	$2s/t$
Load cached channel key	0.59	m_{R}
AES decrypt secret block	0.85	m_{R}
AES decrypt message body	$0.43 \times l$	m_{R}
SHA256 of message headers	$0.0067 \times r$	m_{R}
SHA256 of message body	$0.026 \times l$	m_{R}

Fig. 6. CPU time required for receiving and sending messages in ESP on the mobile devices

The table shows that there is a high cost in computing the channel key and signing the key when receiving a message, and in checking the user signature and decrypting the channel key when sending a message. These are all the IBC operations. By introducing the use of a channel key, we have made it necessary to perform these functions for each friend only twice a month. Had we used IBC in each message directly, the cost would be prohibitively expensive on the mobile device. We see from this table that the marginal cost per message is low.

Social Network Models. To get an estimate on the cost of ESP on realistic social models, we look to Facebook and Twitter, two of the largest social

networks, for guidance on usage patterns. American users of Facebook have on average 229 friends as of mid-2011 [8]. According to 2010 statistics, the average number of items shared with each friend was five per day [14]. To model an active user's behavior, we use a high estimate of having each person send 100 messages to all his friends everyday. As Facebook is largely a symmetric sharing network, so we assume users will receive 100 messages from each one of his friends also. Thus, we can model a Facebook-like network with the following parameters: $r = s = 229, m_{\mathrm{S}} = 100$ per day, and $m_{\mathrm{r}} = 22900$ per day, and $l = 50$ kilobytes. The large message length is chosen to allow for sharing pictures.

A Twitter-like network has active users that subscribe to hundreds of twitter accounts and popular figures have tens of thousands of followers. Consider a private variant, where users can subscribe directly to each other to receive the tweet stream. We model an active user with many followers in such a network with the following parameters: $r = 10,000$ followers, $m_{\mathrm{S}} = 100$ per day, $s = 100$ people followed, m_{r}=100/day, and $l = 4$ kilobytes. Note that people wishing to reach many followers are expected to use public networks like Twitter rather than an ESP-based system.

ESP enables contextual sharing, where apps can constantly keep our friends up to date without user intervention. This is particularly useful for sharing sensitive information such as health monitoring and real-time location sharing, as there is not currently a social network safe enough for such purposes. Such use of ESP might generate a significant amount of message traffic. We model this usage with these parameters: m_{S}=m_{r}=10000/day, but the information would only be shared with $r = s = 20$ people. To allow for sharing large amounts of data, $l = 50$ kilobytes.

Table 7 shows, for each network model, the responsiveness and overhead of messaging on ESP and the CPU burden imposed on the mobile devices. The first column "min latency" measures the time overhead in encrypting and decrypting a typical message. This value does not include the transmission and the routing time. The second column measures the overhead in size. As a data point, a 1-kilobyte message sent to one receiver has a time overhead of 5 ms and 548 bytes. The third and fourth columns compute the total percentage of CPU time spent on sending and receiving ESP messages for the various models.

Simulated Network	Min Latency	Size Overhead (bytes)	% CPU Send	% CPU Receive
Facebook	360 ms	59266	0.05%	0.70%
Contextual	74 ms	5423	0.4%	5.6%
Twitter	14 s	2589186	1.9%	5.4%

Fig. 7. Costs of ESP for different types of social networks.

The results show that the message overhead is minimal except for the Twitter model, where a power user is sending messages to 10,000 followers. It incurs a 14 seconds latency to encrypt and decrypt the message with a size overhead

of 2.6 MB. It shows that the overhead is high, but is still manageable. The CPU burden overall is manageable across all the models. Note that had we not adopted the design of caching channel keys, even the simplest of the network models (Facebook) would have required continuous 2900% CPU usage on the mobile device.

6.3 Encryption Operations on the IBC and IBR Servers

The server components in the ESP system also need to perform IBC operations, so we also evaluated the server performance. Figure 8 shows the costs and frequencies of the operation the servers must perform, based on these parameters:

p: Number of people using the system.
d: Frequency of network drop outs (reconnects to router).

We choose a dropout frequency of 50 times per day to reflect a particular painful case where a phone is switching between WiFi and 3G/4G networks twice per hour. The results are collected by running the service on a single thread on a 2.8 GHz Core i7 processor. Using these figures we determine that we need 1 CPU per 230 million users to run the key service, and 1 CPU per 370 thousand users to provide the authentication mechanism for the router.

Server	Operation	CPU Time (ms)	Frequency
IBE Key	Compute user signature key	1.5	p
IBE Key	Compute user encryption key	9.8	p
Router	Generate Challenge	0.015	$p \times d$
Router	Check user signature	139	$p \times d$

Fig. 8. CPU time required for primitive operations used in IBC key server and the Identity-Based Router.

6.4 Performance of the Router

Our single server instance of the message router prototype can route about 30k small messages per second. The architecture of the RabbitMQ server allows for it to scale up nearly linearly versus CPUs in a clustered configuration [15]. Storage space will be consumed for messages sent to users who have not yet signed up, but these will typically be state update messages, so the actual storage required for the not-yet-users should be small when using implicit content deduplication.

7 Related work

Private Social Platforms. There are a variety of other social platforms which guarantee privacy through cryptography. Diaspora is an example that is intend to

be run on trusted servers rather than on personal mobile devices [18]. FlyByNight provides private communication on top of the Facebook network [11]. This is in stark contrast to ESP which strives to allow users to select a provider of their choice who is the ultimate arbiter of access to their data.

Musubi is an open-source cryptographically private social platform best suited for creating friendships based on physical encounter [4]. It forces user to manually perform key exchange cloaked within a friendly user interace that leverages mobile sensors and pre-existing communication channels. ESP provides a private messaging layer which can replace Musubi's Trusted Group Communication Protocol while enhancing it usability by instantly granting access to the ego-centric social graph.

Mr. Privacy used existing email identities as mechanism for programatic social sharing [5]. It relies on the existing email transport and storage services to provide the social functionality. ESP does not require permanent storage of messages on a central service at all, instead it only uses the existing identities to bootstrap has its own non-human-readable message channel. The channel is optimized for supporting state updates as opposed to documents for user consumption.

ESP provides a distributed social graph, privacy for the contents of communications, and enforces that messsaging is authenticated against real identities. This makes it usage scenarios very different from notable existing privacy systems, such as TOR [3] which attempts to completely obscure the identities of participants.

Identity Standards. Standards like OpenID [16] or OAuth [7] allow users to establish for ownership of identities to third-party service. Our system builds upon these capabilities to allow encrypted messaging with existing identities. SocialKeys [13] and WebID [19] outline methods for publishing a cryptographic key associated to use with existing identities. ESP's architecture simplifies this model by obviating the need for publishing a key, thus increasing the reach of encryption based privacy.

IBC Applications. Identity-based cryptography is deployed today in commerical environments for corporate email encryption [20]. It generally has not been applied to alternative social networks, because of the need for central key authority. The bootstrap ESP described in this paper avoids this pitfall by having a distributed key server, and proscribes that identity providers directly offer the key services in the future, thus eliminating the presense of an additional party snooping on the social graph data even in scenarios where users host their social data with a different provider. IBC has been explored as a technique appropriate for vehicular ad-hoc networks; it is desirable for the real identities to be proven to peers so that users can make trust decisions based on proven identities without reaching the global internet [10].

8 Conclusions

The ESP infrastructure provides a decentralized privacy preserving alternative to centralized app platform like Facebook. Real identities can be used so that

the user base of the system is not limited by a complicated friending process and the full egocentric social network is immediately reachable within the system. Specific mechanisms in the protocol such as using hashed identities and special handling for broadcast messages allow users to keep the maximum amount of information about their social graph private.

In an ideal world, the two services, message routing and IBC secret services, would be federated and provided by the owners of individual domains. With very small modifications, the addition of a domain field in the recipient blocks, the design can immediately be used in a federated model. So far the standards with regard to social information sharing are not showing broad adoption, in fact there is more standardization on proprietary systems every day. This design has the significant advantage that it can bootstrap a data protecting social network without reliance on the original identity providers for implementation. This allows for ongoing research into the development of higher level privacy preserving primitives necessary to make rich social applications that don't put users data at risk.

References

1. Amqp protocol specifications. http://www.amqp.org/resources/specifications.
2. D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *Advances in CryptologyCRYPTO 2001*, pages 213–229. Springer, 2001.
3. 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.
4. B. Dodson, I. Vo, T. Purtell, and M. S. Lam. Musubi: Disintermediated Interactive Social Feeds for Mobile Devices. In *Proceedings of the 21th International Conference on World Wide Web*. ACM, 2012.
5. M. H. Fischer, T. Purtell, and M. S. Lam. Email clients as decentralized social apps in mr. privacy. In *Hotpets*, 2011.
6. P. Gemmell. An introduction to threshold cryptography. *RSA CryptoBytes*, 2(3):7–12, 1997.
7. E. Hammer-Lahav. The OAuth 1.0 protocol, 2010.
8. K. Hampton, L. S. Goulet, L. Rainie, and K. Purcell. Social networking sites and our lives, 2011. http://pewinternet.org/Reports/2011/Technology-and-social-networks/Part-3/SNS-users.aspx.
9. F. Hess. Efficient identity based signature schemes based on pairings. In *SAC 2002, LNCS 2595*, pages 310–324. Springer-Verlag, 2002.
10. D. Huang, Z. Zhou, X. Hong, and M. Gerla. Establishing email-based social network trust for vehicular networks. In *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*, pages 1–5. IEEE, 2010.
11. 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.
12. B. Lynn. The pairing-based cryptography library, 2010. http://crypto.stanford.edu/pbc/.
13. A. Narayanan. SocialKeys: Transparent Cryptography via Key Distribution over Social Networks. *The IAB Workshop on Internet Privacy*, 2010.

14. Internet 2010 in numbers, 2010. http://royal.pingdom.com/2011/01/12/internet-2010-in-numbers/.
15. RabbitMQ: FAQ, 2011. http://www.rabbitmq.com/faq.html#clustering-scalability.
16. 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.
17. A. Shamir. Identity-based cryptosystems and signature schemes. In *Advances in cryptology*, pages 47–53. Springer, 1985.
18. R. Singel. Open Facebook Alternatives Gain Momentum, $115K. http://www.wired.com epicenter/2010/05/facebook-open-alternative/.
19. M. Sporny, S. Corlosquet, T. Inkster, H. Story, B. Harbulot, and R. Bachmann-Gmür. Webid 1.0: Web identification and discovery. draft.
20. Voltage. Voltage identity-based encryption. http://www.voltage.com/technology/ibe.htm.
21. Wall Street Journal. Zynga dependency on facebook, October 12, 2011.