Opaak: Using Mobile Phones to Limit Anonymous Identities Online

Gabriel Maganis*, Elaine Shi†, Hao Chen*, Dawn Song†
University of California, Davis* University of California, Berkeley†

ABSTRACT

Trust and anonymity are both desirable properties on the Internet. However, online services and users often have to make the trade off between trust and anonymity due to the lack of usable frameworks for achieving them both. We propose Opaak, a practical anonymous authentication framework. Opaak enables its users to establish identities with different online services while ensuring that these identities cannot be linked with each other or their real identity. In addition, Opaak allows online service providers to control the rate at which users utilize their services while preserving their anonymity. Hence, allowing the service providers to prevent abuse in the form of spam or Sybil attacks, which are prevalent in such online services that offer anonymity. Opaak leverages the mobile phone as a scarce resource combined with anonymous credentials in order to provide these features. We target two kinds of applications for Opaak and identify their requirements in order to achieve both trust and anonymity. We develop efficient protocols for these applications based on anonymous credentials. In addition, we design an architecture that facilitates integration with existing mobile and web applications and allows application developers to transparently utilize our protocols. We implement a prototype on Android and evaluate its performance to demonstrate the practicality of our approach.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*

General Terms

Security

Keywords

mobile applications, privacy, anonymous credentials, spam, Sybil attacks.

MobiSys'12, June 25–29, 2012, Low Wood Bay, Lake District, UK.

1. INTRODUCTION

Trust and anonymity are often conflicting goals for Internet users. One one hand, users wish to evaluate the trustworthiness of online reviews and recommendations and wish to only establish relations with other trustworthy users. On the other hand, users often wish to remain anonymous and protect their online privacy. For example, users do not want their activities being tracked by the websites they visit or by online advertisement companies.

Our online privacy depends on how we manage our identities. A common approach to remaining anonymous online is to use pseudonyms. However, pseudonymous identities come at the expense of trust i.e., it is difficult to determine whether a pseudonym has been created by a legitimate person or for malicious purposes such as spam. One approach to mitigate this problem is to employ trusted authorities to certify or vouch for the validity of these pseudonyms. Indeed, Douceur showed that Sybil attacks are always possible without a trusted authority to certify the uniqueness of identities in the system (except under extreme and unrealistic assumptions) [23]. Furthermore, a survey showed that trusted certification has been cited as the most common approach to preventing Sybil attacks [28]. However, a trusted certification approach usually comes at the expense of anonymity.

Meanwhile, the mobile phone has become the most personal and most frequently used computing device for people today. As a result, recent trends such as including near field communication (NFC) capabilities indicate a move towards the vision for the mobile phone to become one's universal authentication device i.e., the "mobile wallet" [9]. Mobile phones are also considered to generally be more secure computing platforms than desktop computers [12]. This is partly due to the fact that mobile phones have a more controlled computing environment, where third-party applications are installed through a central application vetting authority (e.g., Apple's App Store). Most importantly, mobile phones can serve as a natural form of scarce resource that can defend against Sybil attacks i.e., people can only finance a limited number of mobile phones.

For these reasons, websites currently employ mechanisms such as SMS verification before users can utilize the services they offer. For example, Craigslist [1] may require a mobile phone number from a user. Craigslist sends a verification code to the number and the user must submit this code before they can post their ad. However, such a mechanism also comes at the expense of anonymity i.e., Craigslist learns the user's mobile phone number.

Fortunately, there is a cryptographic mechanism that can help called *anonymous credentials* [15, 17, 18]. Anonymous credential schemes allow users to demonstrate possession of a credential granted by a trusted authority without revealing anything other than the fact that they own such a credential. We can use this tool to enable a trusted certification approach together with the scarcity of mobile phones in order to provide users with anonymity while preventing Sybil attacks at the same time. In fact, anonymous credentials have formed the basis for privacy-preserving digital identity management frameworks [16]. However, such frameworks have very broad goals and focus on demonstrating users' attributes (e.g., age) to online services. Proving such attributes in a privacy-preserving manner require complex algorithms (e.g., proofs such as [4, 31]) which can become impractical on the mobile phone.

In this paper, we propose a practical anonymous authentication framework, Opaak (OPen Anonymous Authentication frameworK), that provides user's with privacy and anonymity online without the expense of trust. We devise efficient protocols based on anonymous credentials that can be deployed on the mobile platform by targeting specific but practical kinds of applications.

1.1 Motivating Applications

Anonymous, unlinkable accounts across different websites. Single sign-on authentication frameworks such as OpenID [6], allow a user to authenticate to different websites using the same credentials. However, the user's identity is revealed to the websites. Even if the user chooses fictitious identities, these websites can link these identities to the same user. Moreover, the websites where the user has created accounts at can be tracked by the OpenID identity provider.

With Opaak, users can create anonymous accounts across different websites and they cannot be linked with each other. In addition, the Opaak anonymous identity provider does not learn which websites where the user has created these accounts.

Anonymous, unlinkable posts with rate limiting. User opinions — such as votes, ratings, reviews, and recommendations — are important contents in online communities, such as IMDB, Yelp, and Amazon. However, privacy concerns may discourage users from contributing. For example, a user may hesitate to rate a politically sensitive movie, or to review a doctor specializing in HIV treatment. To preserve anonymity, the user could use a different pseudonym for each of her posts, but this would reduce the trustworthiness of her posts, as malicious users could launch spam or Sybil attacks.

Opaak allow users to anonymously contribute votes or ratings in such online services. It guarantees that no two posts from the same user are linkable. To prevent malicious users from manipulating the ratings by aggressively spamming the recommendation system, Opaak allows the site administrator to limit the rate of user posts e.g., a user can contribute at most k posts within a time period T.

1.2 Contributions

We develop Opaak, an anonymous authentication framework that takes advantage of the mobile phone as a natural form of scarce resource in order to limit anonymous identities on the Internet. More specifically, this paper makes the following contributions:

- We demonstrate that applications based on full-fledged anonymous credentials can be deployed on the smartphones with very practical performance. We define two types of applications for Opaak and identify each of their requirements in order to enable trust and anonymity simultaneously. To the best of our knowledge, we are among the first to do this along with [37].
- Opaak emphasizes the use of mobile phones as a scarce resource to solve the Sybil problem on the Internet. Although we may not be the first to make this observation, we are the first to demonstrate how to leverage this fact while simultaneously respecting user privacy.
- We develop a set of protocols related to k-times anonymous authentication (k-TAA) schemes that are efficient and practical on the mobile platform. There are numerous existing k-times anonymous authentication schemes however, they are not suitable for the mobile platform precisely because of range proofs and the state-of-the-art algorithms for doing this do not scale well. We show that we can achieve rate limiting for our targeted applications without resorting to range proofs.
- We design an architecture for deploying Opaak that facilitates integration with existing web applications. Our proxy architecture allows application developers who may not be cryptography experts to utilize Opaak transparently. We believe our architecture is more lightweight compared to existing anonymous credential based systems as a result of our focus on smartphone deployment and our well-defined set of target applications.
- Finally, we implement a prototype of Opaak on Android, evaluate its performance and show that it is feasible for real-world deployment.

2. APPLICATIONS AND REQUIREMENTS

Opaak has three main participants namely the *users*, *relying parties* and *anonymous identity providers*. Relying parties offer some form of service that users wish to use (e.g., Wikipedia [8], Craigslist [1]). People who wish to post advertisements on Craigslist would be considered users. An anonymous identity provider (AIP) is a trusted third party, which vouches for the identity of a user. For example, a cellular service provider might serve the role of an anonymous identity provider and certify that a user is one of its subscribers.

Figure 1 shows a high-level overview of the interactions between the participants. First, users join the system by registering with an AIP and thus acquiring an anonymous credential. The anonymous credential forms the basis for users to create cryptographic proofs that enable them to anonymously authenticate to relying parties so they can use the services they offer. At the same time, relying parties are able to verify properties about the user that allow them to prevent abuse (e.g., spam) while respecting the user's privacy.

Our main goal is to design an authentication framework with privacy-preserving and single sign-on (SSO) properties. However, since anonymity necessarily introduces an avenue for abuse, we must go a step further than existing SSO frameworks (e.g., OpenID [6]) and provide mechanisms to

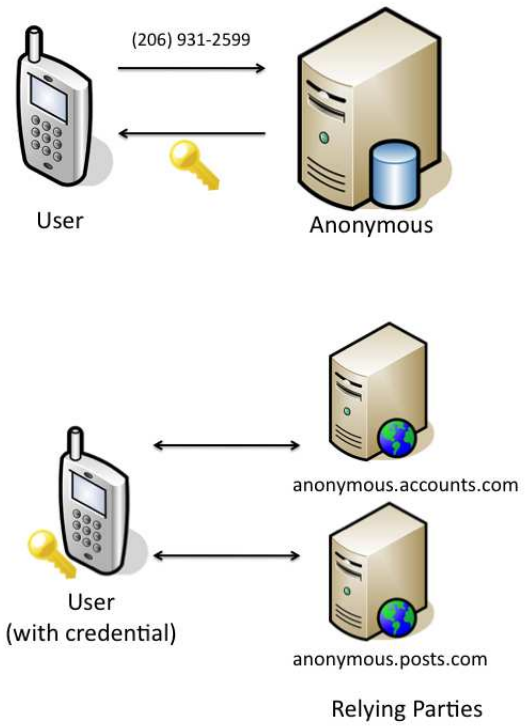

Figure 1: Overview of Opaak. A user is issued a credential by the anonymous identity provider upon presenting their phone number (top). The user can then anonymously utilize services offered by relying parties like posting messages (bottom).

protect relying parties from spam and Sybil attacks. To describe our goals, we logically separate them using two kinds of applications.

First, Opaak supports *anonymous single sign-on authentication*, where users can register anonymous accounts at relying parties then login to them afterwards. An example would be a service like Wikipedia where users would like to edit entries anonymously but at the same time, Wikipedia would like to maintain some form of accountability for edits from the users.

Second, Opaak supports *anonymous message boards* where users can post messages, rate a product, vote on a poll etc. These types of applications offer services but don't necessarily need the users to create an account with them. An example would be a service such as Craigslist where users can post advertisements. As shown by some of the mechanisms they have employed such as mobile phone number and e-mail address verification, spam and Sybil attacks are a concern for Craigslist.

We note that the two applications are not mutually exclusive. A relying party can adopt both anonymous SSO authentication protocols where a user has one or more unlinkable accounts, as well as the anonymous message board protocols where a user may anonymously post messages without being logged in.

In addition to the goals we detail below, we also make it our goal to make these applications accessible to developers who may not be experts in cryptography. Our framework must be designed such that developers can transparently utilize our protocols and easily integrate them into their applications.

2.1 Anonymous Single Sign-On Requirements

Some examples of existing SSO frameworks include OpenID and Facebook Connect [3]. In these frameworks, there are one or more identity providers such as Google or Facebook by which users consolidate their identity or login name. When users wish to authenticate with a relying party, instead of having a different usernames and passwords at each relying party, the user can simply login to Google's OpenID service, or use Facebook Connect. Hence, the identity provider (Google or Facebook) vouches for the user's identity to the relying party. Unfortunately, these SSO frameworks were not designed with user privacy in mind.

2.1.1 Privacy issues

Today's SSO frameworks raise important privacy concerns both for users and relying parties. A user's accounts with different relying parties can be linked with each other, and sometimes even linkable to their real identities. Consider users who use the Facebook Connect feature to login to different relying parties, their accounts and activities at these relying parties could be easily linked to their Facebook identity. For example, relying parties could collude to find out which users they have in common. Furthermore, since the majority of users use their real names on Facebook, these accounts can then be linked to their real identities.

The identity provider can track which relying parties a user is receiving services from. This presents a privacy concern both for the user and for the relying party. Clearly, users sometimes wish to remain anonymous when they visit certain kinds of relying parties (e.g., pornographic websites, or alcohol self-help groups). The relying party may also wish to hide from the identity provider how much traffic or users they have, as this can be sensitive business information.

Therefore, we believe that there is a need for an anonymous SSO framework. Such a framework should offer the following privacy and security guarantees.

2.1.2 Privacy requirements

Unlinkability across different relying parties. We wish to guarantee complete unlinkability for a user's accounts at different relying parties. Given an account at one relying party, there should be no way to determine whether an account at another different relying party belongs to the same user or not. In this way, a user's account at an alcohol self-help group website cannot be linked to his professional profile on sites such as LinkedIn [5].

Transactional privacy from the identity provider. The identity provider should not learn which relying parties a user receives service from or when a user logs in to a certain website. This protects the privacy of both the user and the relying party.

Unlinkability between different accounts with the same relying party. If a user registers more than one account at the same relying party, these accounts should be completely unlinkable. For example, if a user wishes to register two ac-

counts with Wikipedia for editing, one under his real name, and another under a fake name, these two accounts should not be linkable to the same user.

2.1.3 Misuse prevention requirements

While enabling anonymity, Opaak must also prevent abuse in the form of Sybil attacks [23]. These attacks occur when a user is able to create multiple accounts with a relying party thus creating Sybil identities with that relying party. More specifically, we identify our requirements for preventing misuse below.

Limited number of anonymous identities per user. When a user joins the system and registers with an AIP, the AIP must request that the user demonstrate possession of a scarce resource, such as a valid phone number or a credit card number. In this way, a user can only have as much anonymous identities in the system as the resources they actually possess and effectively limiting the capability of a user to create Sybil identities.

Limited number of accounts with the same relying party. Relying parties may wish to limit how many accounts each user can have. Currently, relying parties achieve this through various approaches. Some common approaches include the CAPTCHA [39], the verification of phone numbers via SMS, or the verification of credit card numbers.

We recognize that in some cases, multiple accounts might be desirable like in the Wikipedia example above. In Opaak, we do not prevent users from creating multiple accounts but seek to do it in a controlled fashion with strong guarantees about identities in the system. Opaak allows each user to have up to k accounts at a relying party, where each relying party can choose its own value of k. In addition, if an account is found to be misusing or abusing the service, the relying party can simply freeze that account.

2.2 Anonymous Message Board Requirements

The second type of applications we wish to support are anonymous message board like applications. In these types of applications, users perform actions at a relying party where accounts are not necessary. For example, a user may wish to post a message anonymously, anonymously rate a product or cast a private vote.

2.2.1 Privacy requirements

Unlinkability for any two actions. In anonymous message board type of applications, we wish to ensure that any two actions (e.g., messages posted, ratings, or votes) of the same user, at the same or different relying parties, are completely unlinkable. Given an action at one relying party, there should be no way for the same relying party or another relying party to determine whether another action was performed by the same user or not.

Transactional privacy from the identity provider. Similar to the anonymous SSO application, we also wish to achieve transactional privacy against the identity provider, so that the identity provider does not learn the relying parties utilized by a user.

2.2.2 Misuse prevention requirements

Similar to the anonymous SSO application, our second type of application must also prevent abuse in the form of Sybil attacks. In addition, Opaak must include a mechanism to throttle the usage of services offered by the relying parties to prevent spam.

Limited number of anonymous identities per user. We also wish to defend against Sybil attacks and ensure that each user can only have a limited number of anonymous identities. As discussed above, this can be achieved by requiring that the user demonstrate possession of some scarce resource such as a valid phone number or credit card, when the user registers with the AIP.

Spam prevention. Anonymity is seemingly contradictory to the notion of spam prevention. If the relying party does not learn the identity of the user, how can it throttle the rate at which a user casts votes or posts messages? This question has been answered by earlier work which demonstrates how to realize k-times anonymous authentication [15,33]. Opaak also provides spam prevention and rate throttling features through its *rate limiting pseudonyms*. We elaborate on this technique in Section 3.

2.3 Non-Goals

We consider the following goals to be orthogonal to our efforts, and outside the scope of this paper.

We do not consider network-based attacks on the anonymity of users such as correlating the IP addresses from which requests come from. When requests are generated from mobile phones, today's cellular infrastructure provides some natural protection against IP correlation attacks, as many mobile phones share the same public IP address of the Internet gateway at their cellular provider. Furthermore, existing mechanisms to mitigate these attacks such as Tor [22] can be easily composed with our system so that relying parties do not trivially see the originating IPs of user requests.

Device fingerprinting is another potential approach to identify the origin of network packets. This class of attacks is very challenging to defend against especially when they are based on physical properties of a device (e.g., clock skew [27]). We do not consider such attacks.

3. OPAAK PROTOCOLS

In this section, we describe the protocols we developed in order to realize our target applications.

3.1 Preliminary

First, we briefly describe zero-knowledge proofs and the CL-signature scheme. For a more comprehensive and formal description of these cryptographic techniques, we refer the reader to the specification document for the Idemix cryptographic library [26].

3.1.1 Zero-knowledge Proofs

A zero-knowledge proof of knowledge (ZKPK) is a two-party protocol where a prover convinces a verifier knowledge of a secret value without disclosing the secret value itself. More concretely, consider a prover Alice with a secret key x and public key $y = g^x$. A verifier Bob only knows the values (g, y) but through a ZKPK, Alice can convince Bob that she knows x such that $y = g^x$ without revealing x itself.

In describing our protocols, we use the notation by Camenisch and Stadler [19]. Specifically, to denote the ZKPK of integers α and β such that $y = g^\alpha h^\beta$ and $u < \alpha < v$ holds, where y, g, h are elements of a mathematical group

$G = \langle g \rangle = \langle h \rangle$ we would write the following:

$$PK\{(\alpha, \beta) : y = g^{\alpha} h^{\beta} \wedge (u < \alpha < v)\}$$

We use the various ZKPK techniques available in the Idemix library implemented using a common three-move zero-knowledge protocol [32]. As the name suggests, these protocols require three rounds of communication. Fortunately, they can be made non-interactive (reduced to one round) by means of the Fiat-Shamir heuristic [24], and we denote these with SPK for "signature proof of knowledge" instead of PK. In addition, there are techniques implemented in the Idemix library which allows us to compose multiple ZKPK into a single proof for efficiency. We omit the details of these techniques in the description of our protocols for simplicity.

3.1.2 CL-signatures

The signature scheme proposed by Camenisch and Lysyanskaya [18] is a powerful technique that allows a user to prove possession of a signature to a verifier *without revealing* the underlying message or even the signature itself using efficient ZKPK. In the scheme, the public key of a signer is of the form (n, R, S, Z), where n is a special RSA modulus of length l_n and $R, S, Z \in QR_n$ are generators of QR_n, the group of quadratic residues modulo n. A CL-signature over a message m is of the form (A, e, v). The value A is computed such that $A^e \equiv R^m S^v Z \pmod{n}$ where v is a random number and e is a random prime number with bitlengths l_v and l_e, respectively.

The scheme includes the signing and verification algorithms i.e., CL-Sign and CL-Verify, respectively. In Opaak, we utilize the variant of the CL-signature protocols where the signer and the verifier do not learn the value of the message m. Instead, the signers and verifiers can use ZKPK to ascertain m. In addition, the scheme also includes a CL-Show algorithm. This algorithm allows the signature holder to prove to a verifier that it has obtained a signature from a signer without revealing the message or the signature itself i.e., the *proof of possession* mentioned above. Note that implicit in the CL-Show algorithm is verification of the signature with CL-Verify. We denote a (non-interactive) proof of possession of a CL-signature over a message m as follows:

$$SPK\{(e, m, v) : A^e \equiv R^m S^v Z \bmod n\}$$

3.1.3 Rate Limiting Pseudonyms

At the foundation of the Opaak protocols lies our scheme for forming authentication tokens, which we call *rate limiting pseudonyms*. Simply put, a rate limiting pseudonym (rnym) is globally unique identifier that must belong to a single user i.e., a user can create multiple rnym but an rnym cannot be the same for two users (or even the same user, given different parameters). In the following sections, we show how this simple property enables us to develop protocols that enable anonymity while preventing spam and Sybil attacks at the same time. We discuss the details of how an rnym is formed below.

First, we define some public parameters of the scheme. We denote the group of non-negative integers $\mathbb{Z}^*_\Gamma$ with order $\Gamma - 1$. The public parameters consists of (Γ, ρ, g), where g is a random generator $\langle g \rangle$, which is the subgroup of $\mathbb{Z}^*_\Gamma$ of prime order ρ. Accordingly, the bitlengths of Γ and ρ are given by l_Γ and l_ρ. We also employ a cryptographic hash function H mapping $\{0,1\}^* \rightarrow \mathbb{Z}_\Gamma$.

Now, given a user's secret key uk and an arbitrary string s, we can form an rnym as follows:

$$g_{\mathsf{rnym}} = H(s)^{(\Gamma-1)/\rho}$$
$$\mathsf{rnym} = g_{\mathsf{rnym}}^{\mathsf{uk}} \bmod \Gamma$$

Rate limiting pseudonyms have the following useful properties we take advantage of: (1) an rnym is unique with respect to uk and the string s hence, two rnym formed with different (uk, s) pair, cannot be linked with each other, (2) two rnym formed with the same uk and s pair will always be the same. (3) following from these properties, other users who do not possess uk can form the set of rnym of the user who holds uk as a secret.

Next, we introduce the notion of *rate limiting strings* (rls). Rate limiting strings are strings which encode some form of counting, the very basic of which could be "1", "2", or "3" etc. In our approach, we use a concatenation of the domain name of a relying party, time period identifiers and counters to form the rate limiting string, depending on our target application. For the relying party at *anonymous.posts.com*, during time period T, a user posting for the i-th time would form the following $\mathsf{rls} =$ "$anonymous.posts.com||T||i$" to be allowed to post a message. Indeed, an rnym as defined in the equation above is the same as the "domain pseudonym" feature of the Idemix library [26], where s corresponds to the domain name of a website. However, by defining the meaning of the input to the hash function H more specifically and using rate limiting strings instead, we are able to create the protocols necessary for our target applications.

This way of forming identifiers is very similar to the authentication tokens or tags used in anonymous blacklisting schemes [34–36] and the first k-times anonymous authentication (k-TAA) scheme by Teranishi [33]. The key difference is that in our protocols, we do not hide the base g and how it was formed i.e., it is a public parameter. Hiding the base g would introduce more complex ZKPK as in the anonymous blacklisting and k-TAA schemes. Indeed, not hiding the base weakens our definition of anonymity slightly and the keen reader might have noticed that an rnym is not fully anonymous. We discuss this issue further in Section 6.

3.2 Setup

Recall that Opaak has three main participants which include the users, relying parties and the anonymous identity provider (AIP). In the setup phase, the participants generate cryptographic keys and agree on system parameters.

First, the system parameters are generated and agreed upon by all participants. The AIP generates the public group parameters (Γ, ρ, g) necessary for the rnym scheme. In addition, it is necessary to define the bitlengths $l_n, l_e, l_v, l_\Gamma, l_\rho$, which determine the security of the system. In the case of multiple AIPs, the federation must agree on these values. For a more comprehensive list of these parameters, we refer the reader to the Idemix specification [26].

Next, the participants generate their respective cryptographic keys. The AIP generates its keypair $(\mathsf{sk}, \mathsf{pk})$ and publishes its public key necessary for the CL-signature scheme $\mathsf{pk} = (n, R, S, Z)$. Users generate a master secret uk upon joining the system perhaps via downloading the credential service smartphone application of an AIP. Relying parties do not require cryptographic keys and must simply obtain (securely) a copy of the system parameters described above.

3.3 Registration

The registration protocol is executed between a user Alice and an AIP as she joins the system. This process is analogous to the part of the login procedure in conventional SSO frameworks when a user logs in to the identity provider instead of the relying party. We note that unlike existing SSO frameworks, this procedure is only done once in Opaak. In fact, this is key to achieving transactional privacy.

As described in Section 2, we would like to limit the number of anonymous identities that a user can assume in Opaak. This both protects against Sybil attacks and assures relying parties that the users they are serving correspond to real human beings, or at least a close approximation. To achieve this, Opaak requires users to demonstrate possession of some scarce resource (ResourceName). Some examples of the scarce resource include the user's mobile phone number or the user's International Mobile Subscriber Identity (IMSI).

The protocol enforces that each user (with one unit of resource) can only register up to k times with an AIP. Of course, ideally $k = 1$ but we allow for flexibility depending on the policies an AIP would like to implement.

1. Alice initiates the protocol by sending the AIP a registration request and ResourceName.
2. The AIP maintains a database of ResourceName it has seen. If the ResourceName submitted by Alice has been $> k$ times, then it terminates the protocol and returns `failure` to Alice.
3. The AIP must verify if ResourceName is valid. For example, with mobile phone numbers, the AIP could send an SMS message with a verification code. If the verification fails, then it terminates the protocol and returns `failure` to Alice.
4. Next, the AIP adds ResourceName to its database or updates its count.
5. At this point, the AIP has some assurance that Alice is a real person via ResourceName and proceeds to grant Alice a credential. Now, Alice and the AIP execute the CL-SIGN algorithm. At the successful termination of this protocol, Alice obtains a CL-signature over her secret key uk of the form (A, e, v) such that $A^e \equiv R^{\mathsf{uk}} S^v Z \bmod n$. For the rest of the paper, we may refer to this CL-signature as Alice's anonymous credential or credential for brevity.
6. Alice saves (A, e, v) and keeps it secret.

3.4 Anonymous SSO: Registering accounts

Most relying parties require users to create an account on their site before they can give them access to their services. This registration protocol is executed between a user Alice and a relying party whenever Alice wishes to create a new account. We emphasize that the AIP is not involved in this protocol unlike existing SSO frameworks. In fact, the AIP is not involved in the any of the remaining protocols to be described i.e., it is only needed when Alice obtains her anonymous credential.

Similar to the AIP, relying parties may have different policies regarding users creating multiple accounts however, Opaak seeks to prevent Sybil attacks while simultaneously enabling anonymity. Taking this into consideration, the protocol ensures that each user can only create up to k anonymous accounts with a the relying party i.e., a malicious user who tries to create more than k accounts can be detected.

The relying party defines its parameters *dom* for its domain name and k for the maximum allowable anonymous accounts per user.

1. Alice downloads the parameters *dom* and k.
2. Alice chooses a fresh i such that $1 \leq i \leq k$ i.e., an i value that she has not used to to create an account with yet. This means that Alice must keep track of the i values that she has used so far.
3. Alice forms the corresponding rnym as follows:
$$g_{\mathsf{rnym}} = H(dom||i)^{(\Gamma-1)/\rho}$$
$$\mathsf{rnym} = g_{\mathsf{rnym}}^{\mathsf{uk}} \bmod \Gamma$$
4. Alice sends an account creation request to the relying party along with an optional FriendlyName.
5. Alice creates a proof of possession of her credential from the AIP and a ZKPK of her master secret used to form rnym as follows:
$$SPK\{(e, \mathsf{uk}, v) : A^e \equiv R^{\mathsf{uk}} S^v Z \bmod n \wedge g_{\mathsf{rnym}}^{\mathsf{uk}} \bmod \Gamma\}$$
Alice sends the proof to the relying party. As a result of this proof, the relying party learns rnym, and the values *dom* and i used to form it.
6. The relying party verifies the proof and checks that the parameters are valid i.e., *dom* is the correct domain name and $1 \leq i \leq k$. If the proof verification fails or the parameters are invalid, then the protocol is terminated and the relying party returns `failure` to Alice.
7. The relying party now checks the database it maintains of rnym it has already seen. If the rnym is valid, then it adds it to the database together with FriendlyName associated with it and returns `success` to Alice. Otherwise, the relying party returns `failure`.

3.4.1 Friendly names

An rnym is actually a very large integer value with a long string representation and might not be user friendly. For this reason, Opaak allows users to specify human-readable, friendly names (i.e., username) for their accounts. For example, a Wikipedia user may want to be known under a pen name. Similarly, a friendly name can also be an email address, or the user's real name. The relying party saves the mapping between each account's rnym and friendly name. Of course, this has direct privacy implications on the anonymity of the account. We discuss the privacy issues related to enabling friendly names in Section 6.

3.5 Anonymous SSO: Logging in

This authentication protocol is executed between a user Alice and a relying party. When Alice wishes to login to an account she has created with the relying party (using the previous protocol), she proves possession of her anonymous credential and shows a valid rnym along with a ZKPK of her master secret used to form the rnym. We describe the step by step process below.

The relying party defines its parameters *dom* for its domain name and k for the maximum allowable anonymous accounts per user.

1. Alice downloads the parameters *dom* and k.
2. Alice keeps track of the i values corresponding to the accounts she has created with the relying party. She looks up the correct i optionally corresponding to FriendlyName.

3. Alice forms the valid rnym as follows:

$$g_{\mathsf{rnym}} = H(dom||i)^{(\Gamma-1)/\rho}$$
$$\mathsf{rnym} = g_{\mathsf{rnym}}^{\mathsf{uk}} \bmod \Gamma$$

4. Alice creates a proof of possession of her credential from the AIP and a ZKPK of her master secret used to form rnym as follows:

$$SPK\{(e, \mathsf{uk}, v) : A^e \equiv R^{\mathsf{uk}} S^v Z \bmod n \wedge g_{\mathsf{rnym}}^{\mathsf{uk}} \bmod \Gamma\}$$

Alice sends this proof to the relying party along with a login request and the correct FriendlyName. As a result of this proof, the relying party learns rnym, and the values *dom* and i used to form it.

5. The relying party verifies the proof and checks that the parameters are valid i.e., *dom* is the correct domain name and $1 \leq i \leq k$. If the proof verification fails or the parameters are invalid, then the protocol is terminated and the relying party returns `failure` to Alice.
6. The relying party has authenticated Alice and must simply look up the rnym entry in its database (and optionally Alice's FriendlyName). If rnym is not in the database then Alice has made an error and has tried to authenticate with an i value that she has not used to create an account with the relying party. The relying party may proceed with the previous registration protocol and add the rnym entry to its database or simply terminate the protocol and return `failure` to Alice.
7. After the relying party has retrieved the account information pertaining to rnym, then the relying party's application protocols can proceed as regularly. For example, the relying party could send back a unique token to establish an HTTP session with Alice.

3.6 Periodic k-Times Anonymous Authentication

The second class of applications targeted by Opaak are those where the relying party offers services where users don't necessary need to create accounts are their site such as posting messages, reviews, ratings, or casting votes. Enabling anonymity for these kinds of applications means that we must prevent spamming behavior from malicious users as well. Below we describe how Opaak reconciles between the seemingly conflicting goals of anonymity and spam control.

The relying party publishes its parameters *dom* for its domain name and k for the maximum allowable requests per time period T. For example, if Craigslist would like to allow users only one advertisement post per hour, then $k = 1$ and the time period identifier T would be updated every hour. The only requirement for T is that it is a unique identifier the simplest example of which is a timestamp. The protocol ensures that a user Alice can only create one valid rnym per time period T.

1. Alice downloads the relying party's parameters *dom*, k and current time epoch identifier T. Note that a k is only valid for the current time epoch i.e., a relying party can change its policy at every time epoch if it wishes. For example, if the $k = 1$ for the current time period, the relying party can easily change it to $k = 2$ for the next time period without any problems whatsoever. This means that Alice cannot cache these parameters.
2. Alice must keep track of the i values she has used so far for the current time period. Alice chooses a fresh i such that $1 \leq i \leq k$ i.e., an i value that she has not used to issue a request so far.
3. Alice forms a valid rnym as follows:

$$g_{\mathsf{rnym}} = H(dom||i||T)^{(\Gamma-1)/\rho}$$
$$\mathsf{rnym} = g_{\mathsf{rnym}}^{\mathsf{uk}} \bmod \Gamma$$

4. Alice creates a proof of possession of her credential from the AIP and a ZKPK of her master secret used to form rnym as follows:

$$SPK\{(e, \mathsf{uk}, v) : A^e \equiv R^{\mathsf{uk}} S^v Z \bmod n \wedge g_{\mathsf{rnym}}^{\mathsf{uk}} \bmod \Gamma\}$$

Alice sends this proof to the relying party along with her request (e.g., a message for an anonymous message board application). As a result of this proof, the relying party learns rnym, and the values *dom*, i and T used to form it.

5. The relying party verifies the proof and checks that the parameters are valid i.e., *dom* is the correct domain name, $1 \leq i \leq k$ and T is the current time epoch. If the verification fails, then the protocol is terminated and the relying party returns `failure` to Alice.
6. The relying party searches its database for the current time epoch and checks whether it has seen rnym before. If rnym is not in database then it can proceed to grant Alice's request (e.g., add the message to it's message board) and return a `success` status to Alice. Otherwise, Alice is attempting to reuse the rnym and the protocol terminates and a `failure` status is returned to Alice.

Our technique can be thought of as a combination of the schemes proposed by Teranishi et. al. [33] and Camenisch et. al. [15]. However, it is more efficient since we don't require complex ZKPK (e.g., proving a number lies within a given range [4, 31]). Moreover, the penalty for misbehavior is less severe i.e., we do not de-anonymize (by revealing the public key of) a misbehaving user. We discuss the privacy implication of this tradeoff for efficiency in Section 6.

4. OPAAK ARCHITECTURE

4.1 Components

Figure 2 shows the architecture of Opaak and an overview of how the components interact with each other. There are three major components (each corresponding to the participants in the system): a relying party (RP) application, a mobile phone (users), and the AIP application. An RP application (or app) is composed of a mobile phone app and a web app. Our approach allows RP apps to utilize our protocols transparently. The Opaak components serve as a proxy for the requests carried out between the RP mobile phone app and the RP web app. In other words, we augment the existing communication between the RP app components with the Opaak protocols by sitting in between the RP phone app and the RP web app. This approach lends itself to compatibility with current systems and incremental deployment since existing RP apps would require minimal modification in order to use our protocols.

We now describe the function of each component. First, the AIP application is a web service consisting of the Registration Service, which executes the AIP side of the reg-

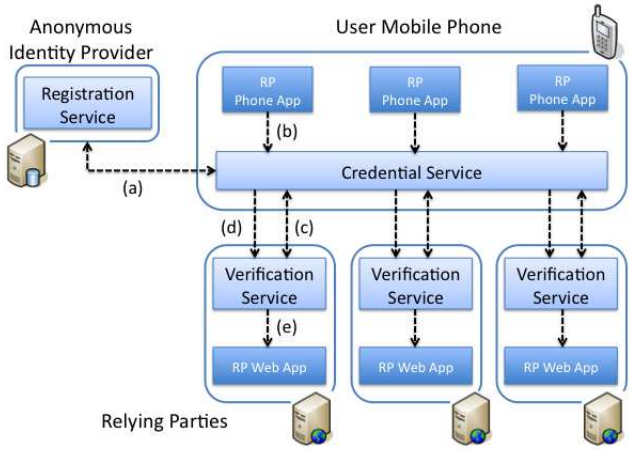

Figure 2: Architecture components. (a) User registers with an AIP through the Credential Service. (b) Phone app sends a request. (c, d) Credential Service runs Opaak protocols with the Verification Service. (e) Verification Service forwards the request to the web app.

istration protocol. It is also responsible for publishing the public parameters of the system (e.g., the RSA modulus n). Next, the mobile phone is responsible for interacting with the other components on behalf of the user. It consists of RP mobile phone apps which can utilize the Opaak protocols transparently through the Credential Service. The Credential Service is the main software component on the phone running all the protocols on behalf of the user. It registers the user with an AIP and obtains an anonymous credential by communicating with the Registration Service. It is also responsible for controlling access to the Opaak cryptographic material and keeping them privy to the other components. When using relying party services, it takes the requests from RP phone apps and executes the protocols with the Verification Service on the RP web app side. Finally, the Verification Service is responsible for maintaining the database of rnym and forwarding requests (upon successful execution of the protocols) to the RP web app. In the case of the anonymous SSO, the rnym could be exported since the RP app would need to use them as identifiers for its own database.

As a concrete example, consider a simple online message board RP app. It has a mobile phone app that takes an input message from the user, sends it as an HTTP POST request to a web app, which then takes care of displaying it on a webpage. From Figure 2, the RP app can be easily modified to allow anonymous posts (e.g., one post every five minutes) as follows:

(a) To participate, a user first registers with an AIP Opaak through the Credential Service. At the end of the protocol, the Credential Service saves the user's master secret and anonymous credential. This is only done once, for example, a cellular service provider serving as an AIP could distribute phones with the registration protocol already completed.

(b) The RP phone app (instead of sending the POST request directly to the RP web app) sends the message to the Credential Service. Of course, the RP phone app must let the Credential Service know the Internet location of the Verification Service. For example, the request to the Credential Service might look like the following: Request("Post this message!", `http://anonymous.posts.com/verify`).

(c) The RP web app publishes $k = 1$ and the current time period identifier T (updated every five minutes) through the Verification Service. The Credential Service authenticates with the Verification Service using the periodic k-times anonymous authentication protocol described in Section 3.

(d) Upon successful execution of the protocol, the Credential Service can proceed to send the message from the RP phone app to the Verification Service.

(e) Finally, the Verification Service forwards the message to the RP web app so it can proceed as originally intended.

We note that the RP phone app does not need to know about the first step, (a). It simply discovers the Credential Service on the phone and uses it.

4.2 Practical Considerations

In this section, we describe several practical and usability considerations that result from our mobile phone based architecture.

4.2.1 *Credential Access Control*

The Credential Service on the phone must control access to the user's master secret and anonymous credential. We can leverage several features of modern mobile operating systems (OS) such as Android. We describe them below assuming a deployment of the Credential Service as an Android app.

First, Android apps execute in a sandbox and are isolated from other apps on the phone. By default, only the Android app itself can access its resources such as files. We can rely on Android to enforce this mechanism so that no other third party RP phone app can read or write the files containing the user's master secret and anonymous credential.

Second, Android allows apps to communicate with each other through well-defined interfaces i.e., Android apps can call other Android apps. Moreover, it allows passing parameters to callee applications through generic "bundles". This feature actually lends itself well to our proxy architecture. RP phone apps can encapsulate their web requests in bundles and pass them along to the Credential Service to be forwarded to the RP web app.

Third, Android apps can declare custom permissions that they could require from other applications. For example, a Credential Service Android app can declare a custom permission `android.USE_CREDENTIAL_SERVICE` and require that other apps that would like to call it to declare this permission in their manifest. Android can enforce this permission check during installation of the RP phone app hence, only apps that the phone owner has allowed to use this custom permission would be able to use the Credential Service.

4.2.2 *Registration using SMS messages*

Recall that an AIP must require users to demonstrate possession of scarce resource such as a valid mobile phone number as a way to defend against Sybil attacks. To make the registration process more usable on a mobile phone, we propose the following method which minimizes user involvement.

Recall that the user and AIP execute the CL-Sign algorithm during registration. One possible method is for the AIP to piggyback the mobile phone number verification onto the CL-signing protocol. Without going into much detail, in the last step of this protocol, the AIP sends the user a part of the CL-signature necessary for the user to execute further steps of computation before the actual CL-signature is obtained. The AIP can split this last part of the CL-signature C into two random shares, a small number s, and a larger share t, such that $C = s + t$. The smaller s will be sent over an SMS message, while t is sent directly over a data connection where the Credential Service receives it directly. We assume that the Credential Service is granted permission to read SMS messages, such that it can read the share s directly from the received SMS message, without having to prompt the user. In this way, the Credential Service can compute $C = s + t$, and then to proceed to computing the final CL-signature.

Alternatively, the AIP can simply send a verification code over SMS which the Credential Service reads and sends back to the AIP for verification. However, this approach introduces one more round trip than the above.

4.2.3 Defense against Device Theft

When storing the credential on the mobile phone, a major concern is device theft. Absent of any protection, if a legitimate user's device is captured by a malicious party, the malicious party will be able use the credential and impersonate the user when logging onto websites. The following methods can help mitigate the device theft problem.

First, we can employ conventional mechanisms such as master passwords. We may require that the user enter a master password on the phone during any authentication request. In addition, to prevent a malicious party from trivially reading credential from the phone's storage, we can also encrypt the credential under a secret key generated from the user's master password.

Second, we can employ credential revocation mechanisms. The Idemix library supports protocols for updating an existing credential i.e., the user simply re-runs parts of the CL-Sign algorithm. This feature can be used to implement credential revocation. We can add an expiration date to the basic anonymous credential used by Opaak. When the date expires, the credential is rendered invalid and the user must re-run the registration protocol with the AIP. Clearly, there is trade-off between mitigating the problems of device theft effectively and efficiency. The smaller the time window for expiration is, the better for defending against device theft. Otherwise, it would require more resources on the AIP's part to enable frequent re-runs the registration protocol to re-validate the user's credential. Indeed, one of the advantages of our approach is that the user only needs to register with the AIP once.

5. IMPLEMENTATION AND EVALUATION

To evaluate the practicality of our approach, we implemented a prototype which includes both mobile and web applications. We used the Identity Mixer cryptographic library version 2.3.2 for Java to implement the ZKPK and CL-signature cryptographic operations. To implement our web applications, we used the Django framework. However, to carry out the actual cryptographic operations on the server side, we simply wrote Java programs that were called by Django applications. A more ideal solution would be to leverage projects like Jython but as a first step, we opted for less engineering overhead but even our unoptimized implementation performs well.

5.1 Protocol Parameters

We developed a Java program to generate the necessary parameters discussed in Section 3. First, we set the bitlengths for the rnym parameters to be $l_\Gamma = 1632$ and $l_\rho = 256$ and the bitlengths for the CL-signature parameters to be $l_n = 2048, l_e = 120$ and $l_v = 2724$. We follow the recommendations for strong security parameters in the Idemix specification. The program computes the group parameters (Γ, ρ, g) and generates the the public-private key pair for an AIP where the public key is (n, R, S, Z).

5.2 Setup and Registration

We developed a web application that implements the Registration Service for an AIP. Our AIP assumes all users requesting a credential are legitimate i.e., it does not require the user to demonstrate possession of a scarce resource such as a phone number. We believe this is reasonable for an experimental prototype. However, in a real world setting, this is absolutely necessary, otherwise Opaak does not have a basis for its guarantees against spam and Sybil attacks.

5.3 Anonymous Posts Website

We developed an example relying party, which lets users anonymously posts messages to a public forum using our mobile phone application. The website demonstrates the periodic k-times anonymous authentication protocol described in Section 3. It allows up to 1000 posts per hour i.e., $k = 1000$ and the time epoch identifier T is updated every hour. It can be viewed live at `http://trident.cs.ucdavis.edu/website/posts`.

The website includes a `posts` and a `verify` web application. The `posts` application is a standard database driven website which displays messages posted by users. The `verify` application unsurprisingly implements the Verification Service. Upon successful execution of the protocols with the Credential Service (running on the user's phone), the message string is inserted into the `post` application's database by the `verify` application.

5.4 Anonymous Post Android Application

We developed a mobile phone application on the Android 2.2 platform, which implements the Credential Service. The Android app can be downloaded at `http://trident.cs.ucdavis.edu/app/MobiCredAndroid.apk`.

The app has two Android activities corresponding to the two main responsibilities of the Credential Service. First, it includes an activity to run the registration protocol with the AIP registration service application we developed. This activity also downloads the necessary files containing the group parameters and AIP public key described previously. Second, an activity that runs the periodic k-times anonymous authentication protocol. It retrieves the T and k parameters from the above website, forms a valid rnym for the current monitoring period then submits the message entered by a user to the Verification Service via HTTP POST. Note that instead of developing a separate example RP phone app, we simply include the message post functionality into the

Credential Service in order to facilitate our benchmark experiments.

5.5 Performance

	Min	Median	Mean	Max
Setup (GP)	1.122	23.549	57.266	586.902
Setup (AKP)	2.041	30.362	39.662	191.088
Register	3.541	4.459	4.490	8.654
Post	2.076	2.238	2.553	7.385

Table 1: Wall clock times (in seconds) to perform 100 trials of the setup phase operations, registration protocol, and message post operation.

We ran several benchmarks to measure the performance of our implementation. The system that hosted our experiments for the setup operations was a quad-core 2.67 GHz Intel Xeon processor with 6GB of RAM. The system that hosted our website was a standard VMWare Workstation virtual machine hosted by the previous system and configured with 1 2.67 GHz processor, 1GB of RAM and a bridged network adapter. Both systems were connected to the Internet via ethernet on a university network. The smartphone that hosted our Android applications was a Motorola Droid running Android 2.2.

Table 1 gives the wall-clock time in seconds (calculated via the `java.util.Date.getTime` API call). For the setup operations, GP refers to the time to generate (Γ, ρ, g) while AKP refers to the time to generate the AIP's keypair where the public part is (n, R, S, Z).

For the register and post message operations, the time given is measured from the user's perspective and includes all the operations on the mobile phone, network requests, and all the operations on the server. More precisely, for the registration protocol, the time is measured starting from the first web request and ending after the credential has been saved to a file on the device. Similarly, for posting a message, the time is measured starting from the initial web request, including the proof computations on the mobile phone, the verification and database storage operations on the server, then ending after the user has received a status code from the server indicating a successfully posted message. Each operation was timed for 100 repetitions and given in the table are the min, median, mean and max times.

5.6 Discussion

Notice that the max execution time for GP is almost ten minutes long. Moreover, the standard deviation for the GP times is 107.104 while for the AKP times is 35.190. Compare these to the register and post operations which are both under one. The wide variation is due to the algorithms used to generate the random numbers required in the protocols.

For example, the GP operation involves generating the random generator g, and the algorithm for this is to simply keep generating random numbers until a number satisfying the constraints is found i.e., the order is prime and lies within a given interval. In addition, the AKP operation involves generating the "safe RSA modulus" n in the public key i.e., n is the product of two safe primes. This means that $n = pq$ where $p = 2p' + 1$, $q = 2q' + 1$ and p', q' are prime numbers as well. Here, the algorithm is the same, generate random numbers and loop until the constraints are satisfied. Indeed, there are optimizations to speedup the GP and AKP operations by taking advantage of multi-core processors. However, these operations are only performed once for the lifetime of the system hence we deemed the running times from our experiments reasonable and did not explore the optimizations further.

6. SECURITY ANALYSIS

The security of Opaak relies heavily on the security of CL-signatures and the proofs of knowledge used. In this section, we discuss the resistance of our approach to various attacks with respect to our goals described in Section 2. Recall that we are not concerned with network based attacks i.e., we are concerned with misbehaving participants in the system.

6.1 Linking User Identities and Tracking

6.1.1 Identifying Different Users

Recall from Section 3 that an rnym is of the form $\mathsf{rnym} = g^{\mathsf{uk}} \bmod \Gamma$ where $g = H(s)^{(\Gamma-1)/\rho}$ and s is an arbitrary string. Ideally, full anonymity means that given two different rnym, a relying party *cannot determine whether it belongs to the same user or not.* As previously mentioned, having a public base (g) has a privacy implication. However, in Opaak, two different rnym with the same input to the hash function means that they *must belong to two different users.* More concretely, consider users Alice and Bob with master secret keys uk_a and uk_b, respectively. If they try to create their first accounts at a relying party with domain name *dom*, then Alice would form $\mathsf{rnym}_a = g^{\mathsf{uk}_a} \bmod \Gamma$ and Bob would form $\mathsf{rnym}_b = g^{\mathsf{uk}_b} \bmod \Gamma$ where $g = H(dom||0)^{(\Gamma-1)/\rho}$.

Indeed, this is a slightly weakened definition of anonymity. A consequence of this is that a relying party would be able to count the number of unique users it has seen. We emphasize that the above example does not break our definition of linkability i.e., the relying party knows that two users have created accounts but it cannot identify which user. Moreover, if Alice chooses to create a second account where $g = H(dom||1)^{(\Gamma-1)/\rho}$, the relying party cannot tell whether it is Alice, or Bob. Hence, even if relying parties collude, the most information they can glean is limited to comparing the number of unique users they have seen.

Hiding the value of g_{rnym} as some existing schemes have done would require us to utilize ZKPK techniques [4, 31] that can hamper performance significantly in practice. We believe this tradeoff is reasonable in order to keep our protocols efficient on the mobile platform.

6.1.2 Offline Identity Providers

Our main concern with AIPs is that they attempt to track the user's online transactions with different relying parties. Unlike existing SSO frameworks, the user's transactions need not be brokered through the AIP in Opaak. This means that the AIP learns nothing about the transaction. Furthermore, the AIP can be offline and is never involved during transactions between users and relying parties.

During registration, the AIP can learn some identifying information from users because they must present some form of scarce resource such as a mobile phone number. However, the AIP is limited to knowing that it has certified such a phone number and nothing else.

6.1.3 Optional Friendly Names

If a user does not associate a FriendlyName with an account, then that account is an anonymous account. However, if a user associates the same FriendlyName to her accounts at different relying parties, then these accounts can be linked. Similarly, a user might willingly reveal her real identity to the relying party e.g., posting a message containing their real name, or using the same profile picture on different websites. To benefit from the Opaak's unlinkable accounts, a user must choose a different FriendlyName at an alcohol self-help group than her professional LinkedIn account. A user should also refrain from sending any data that may expose their identity to the alcohol self-help group website.

6.1.4 Collusion

Relying parties, users and AIPs might collude to attempt to de-anonymize an rnym they have seen, or link a user's proofs of possession of CL-signatures, with the goal of tracking the relying parties it has visited. Provided that the registration protocol was carried out honestly between the user and an AIP, then AIPs cannot do this. This is also true even in the case that an AIP is also a relying party. This is because of the property of CL-signatures that two proofs of possessions cannot be linked with each other.

At most, colluding participants could compare the number of unique rnym a relying party has seen. It is not possible for two relying parties to link the users they have seen hence, colluding participants could not even count the total number of unique users in the system.

6.2 User Impersonation

One of our concerns is that users might try impersonate other users in the system. They might do this by forging a CL-signature or by taking over another user's mobile phone. Provided that the registration protocol was carried out honestly between the user and an AIP, then this is not possible. CL-signatures are resistant to forgery even if a user colludes with other participants in the system. On the other hand, a malicious user could simply take over another user's mobile phone to either use it directly or perhaps copy the stored master secret. We describe ways to mitigate device theft in Section 4.2.

6.3 Spam and Sybil Attacks

As it enables user anonymity online, Opaak's main concern come in the form of spam and Sybil attacks. Users could attempt to spam service requests to relying parties. They might also attempt to register to an AIP multiple times and obtain multiple credentials.

Provided that the AIP requires the user to demonstrate possession of a scarce resource, then the protocols outlined in Section 3 prevent a user from creating an uncontrolled number of accounts with a relying party or spamming requests to a relying party by using multiple credentials. Furthermore, the periodic k-times anonymous authentication protocol which makes sure that a user can only create k valid rnym for a specified time period per credential they possess.

7. RELATED WORK

A number of digital identity management (DIM) frameworks have been proposed with the most notable ones being OpenID [6] and Microsoft's CardSpace [20]. OpenID and CardSpace focus on allowing users to consolidate and manage their digital identities and don't offer users anonymity from relying parties. In contrast, Opaak's main goal is to provide users anonymity from relying parties (as well as the identity providers) while simultaneously preventing spam and Sybil attacks. On the other hand, a suite of cryptographic techniques called U-Prove technology [2] developed by Stefan Brands has been integrated into CardSpace to offer similar capabilities as PRIME. Presumably, Opaak can also use U-Prove to achieve its goals, however we opt to focus on Idemix which provides better support and documentation. Similarly, PseudoID [21] augments the OpenID framework with blind signatures that allow users to anonymously authenticate with identity providers. The addition of a blind signature service (BSS) gives PseudoID users transactional privacy. However, identity providers can still log the relying parties users are visiting since users (although it doesn't know which user) are still redirected by relying parties to identity providers. Opaak eliminates this communication flow completely by using anonymous credentials and the AIP is never involved when a user authenticates with a relying party. Furthermore, Opaak enables relying parties to rate limit users while PseudoID must completely rely on the BSS to prevent Sybil attacks.

There are DIM frameworks that consider user privacy as a design goal and these include Privacy and Identity Management for Europe (PRIME) [16] (now PrimeLife [7]), VeryIDX [30], and PriMan [38] PRIME is an ongoing effort involving a consortium of companies and academia aiming to develop a comprehensive framework for privacy for the web and its applications. Indeed, the Identity Mixer cryptographic library [26] was developed as part of the PRIME project and is utilized by the PRIME core to enable anonymous authentication in applications. While both Opaak and PRIME utilize anonymous credentials, the PRIME architecture [10] has much more components as a result of its wide scope of design goals. Opaak has a more lightweight architecture as we focus on smartphone deployment and a well-defined set of target applications. In addition, our architecture allows developers to use our techniques more transparently. PRIME also focuses on managing the identity attributes (e.g., age, sex, marital status) of users and allowing them to selectively disclose these attributes to relying parties [11]. While Opaak could easily integrate such selective disclosure features as well, it opts to focus on giving relying parties the ability to control spam and Sybil attacks as these can be done with efficient protocols. Zero knowledge proofs of such identity attributes (e.g., proving an age value lies in a given interval like "≥ 18") can include techniques that do not scale well [4, 31]. VeryIDX also places an emphasis on proving identity attributes (e.g., credit card number, social security number) to relying parties. They do this with their "aggregate zero knowledge proof" protocol based on an aggregate signature scheme by Boneh et. al.. Similar to Opaak, they also realize the increasing ubiquity of mobile phones and store the secrets on the phone itself. As a result, they utilize a combination of cryptographic techniques (secret sharing) and secure hardware to protect the secrets on the phone. In contrast, Opaak focuses on controlling how third party applications access the credentials and leverage the isolation mechanisms and permission framework available with a modern smartphone OS such as Android. PriMan claims to be a user-centric identity middleware frame-

work. PriMan takes into consideration third party application developers in their architecture design which reconciles different approaches to digital credentials and allows developers to easily switch from one approach to another.

Several applications based on anonymous credentials have also been described in literature. Blanton designed a system that adds anonymous access to online subscription services [14]. During the subscribe phase, users and service providers agree on a subscription type and an expiration date on which the service provider issues a CL-signature. Subsequently, users access the service anonymously until the expiration date by proving possession of the CL-signature. To prevent users from abusing the system, the access phase requires the user to obtain a new signature on a fresh access token where the old access token is stored by the service provider so that it can check whether tokens are being reused. Similar to Opaak, the anonymous ePoll system developed by Verhaeghe et. al. utilize the Idemix library on an Android phone [37]. Anonymous voting applications can be supported by Opaak as well however, their system goes further and provides features such as using verifiable encryption to provide voters with receipts that voters can use to prove participation. Furthermore, they conduct an evaluation and show how range proofs can affect performance negatively and that they do not scale well. This is precisely the reason why Opaak makes the tradeoff for privacy as discussed in Section 6. While Opaak and the ePoll system above show that full-fledged anonymous credentials can be utilized on commodity smartphones, Bichsel et. al. show that it can be done on severely resource constrained standard Java smart cards [13].

A number of cryptographic schemes have been proposed that have similar goals to rate limiting pseudonyms. The original k-times anonymous authentication scheme developed by Teranishi et. al. uses authentication tokens formed very similarly as rate limiting pseudonyms. They allow users to form up to k valid tokens which can allows them to authenticate with a relying party. It is based on group signatures instead of CL-signatures and could potentially be used to implement our protocols. More recently, a periodic n-times authentication scheme proposed by Camenisch et. al. [15] allows users to show up to n tokens anonymously and unlinkably to a verifier. Exceeding n means a user gets de-anonymized i.e., the verifier can compute the user's public key from (reused) tokens. In contrast, our protocols do not have such harsh punishment for misbehavior and only detects it so that the user's request can be denied. Both these schemes could potentially be modified to fit our scenario however, they both require complex ZKPK (e.g., proving a number lies in a given interval) that we opt to avoid in our scheme.

Anonymous blacklisting schemes are protocols that allow relying parties to maintain a blacklist of misbehaving users while preserving user privacy. There are the BLAC family of protocols [34–36] and the Nymble family of protocols [25,29]. In the BLAC protocols, users authenticate to relying parties by downloading the relying party's current blacklist and creating a ZKPK that they are not in the blacklist. Relying parties maintain their own blacklist containing "tickets" which are transcripts of previous authentications from users. Tickets are formed such that during authentication, users prove that they could not have created any of the tickets currently on the blacklist. In the Nymble protocols, users form "nymbles" based on their credentials and relying parties file complaints on a nymble to a Nymble Manager which can then include all other nymbles that a user may form on the blacklist. Since our protocols were tailored with specific applications in mind, they are more efficient than these blacklisting schemes. Moreover, it is unclear how they can be adapted for a periodic k times anonymous authentication scheme while maintaining the same security guarantees.

8. CONCLUSION

In this paper, we proposed Opaak, a framework for anonymous authentication that enables users to participate in online services anonymously while making sure that relying parties have the ability to prevent spam and Sybil attacks. It targets two kinds of applications, an anonymous single sign-on application, and anonymous message board types of applications. Opaak provides efficient protocols for these applications based on anonymous credentials. In addition, we designed a proxy based architecture that allows Opaak to be integrated easily with existing mobile and web applications. Our architecture also allows applications developers who may not be well-versed in cryptography to utilize our protocols transparently. We implemented a prototype on an Android phone, and demonstrated that it can perform anonymous authentications in a matter of seconds. With our work, we hope to encourage application developers to adopt systems like Opaak and motivate the deployment of anonymous credential based systems in the current Internet landscape.

9. ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers and our shepherd Yih-Chun Hu for their insightful comments. Special thanks to John Bethencourt for help with initial work on this project, Patrik Bichsel for advice on Idemix, and Man Ho Allen Au for advice on k-TAA schemes. This material is based upon work partially supported by the National Science Foundation (NSF) under Grant No. 0644450 and 1018964 and by the Air Force Office of Scientific Research (AFOSR) under MURI Grant No. 22178970-4170 and No. FA9550-08-1-0352. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the AFOSR and NSF.

10. REFERENCES

[1] Craigslist. `http://www.craigslist.org`.
[2] Credentica. `http://www.credentica.com/`.
[3] Facebook connect. `http://developers.facebook.com/docs/guides/web`.
[4] Four squares. `http://schorn.ch/lagrange.html`.
[5] Linkedin. `http://www.linkedin.com`.
[6] OpenID website. `http://openid.net`.
[7] PrimeLife. `http://www.primelife.eu/`.
[8] Wikipedia. `http://www.wikipedia.org`.
[9] R. Anderson. Can We Fix the Security Economics of Federated Authentication? In J. A. Malcolm, editor, *International Workshop on Security Protocols*, 2011.
[10] C. Andersson, J. Camenisch, S. Crane, S. Fischer-Hubner, R. Leenes, S. Pearsorr, J. Pettersson, and D. Sommer. Trust in PRIME.

International Symposium on Signal Processing and Information Technology, 0:552–559, 2005.

[11] E. Bangerter, J. Camenisch, and A. Lysyanskaya. A Cryptographic Framework for the Controlled Release of Certified Data. In *Security Protocols Workshop*, 2004.

[12] M. Becher, F. C. Freiling, J. Hoffmann, T. Holz, S. Uellenbeck, and C. Wolf. Mobile Security Catching Up? Revealing the Nuts and Bolts of the Security of Mobile Devices. In *IEEE Symposium on Security and Privacy*, 2011.

[13] P. Bichsel, J. Camenisch, T. Groß, and V. Shoup. Anonymous Credentials on a Standard Java Card. In *CCS*, 2009.

[14] M. Blanton. Online subscriptions with anonymous access. In *ASIACCS*, pages 217–227, 2008.

[15] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich. How to win the clonewars: Efficient Periodic n-times Anonymous Authentication. In *CCS*, 2006.

[16] J. Camenisch, R. Leenes, and D. Sommer, editors. *Digital Privacy: PRIME - Privacy and Identity Management for Europe*, volume 6545. Springer, 2011.

[17] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT*, pages 93–118, 2001.

[18] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *Security in Communication Networks*, 2002.

[19] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In B. Kaliski, editor, *Advances in Cryptology âĂŤ CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. 1997.

[20] D. Chappell. Cardspace. `http://msdn.microsoft.com/en-us/library/aa480189.aspx`.

[21] A. Dey1 and S. Weis. PseudoID: Enhancing privacy for federated login. In *HotPETS*, 2010.

[22] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 303–320, 2004.

[23] J. R. Douceur. The sybil attack. In *IPTPS*, pages 251–260, 2002.

[24] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. Odlyzko, editor, *Advances in Cryptology âĂŤ CRYPTOâĂŹ 86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. 1987.

[25] R. Henry, K. Henry, and I. Goldberg. Making a Nymbler Nymble Using VERBS. In *PETS*, pages 111–129, 2010.

[26] IBM Research Zurich Security Team. Specification of the Identity Mixer Cryptographic Library v. 2.3.2. `https://prime.inf.tu-dresden.de/idemix/`, December 2010.

[27] T. Kohno, A. Broido, and K. C. Claffy. Remote Physical Device Fingerprinting. In *IEEE Symposium on Security and Privacy*, pages 211–225, 2005.

[28] B. N. Levine, C. Shields, and N. B. Margolin. A Survey of Solutions to the Sybil Attack. Tech report 2006-052, University of Massachusetts Amherst, 2006.

[29] Z. Lin and N. Hopper. Jack: Scalable Accumulator-based Nymble System. In *WPES*, pages 53–62, 2010.

[30] F. Paci, E. Bertino, S. Kerr, A. C. Squicciarini, and J. Woo. An Overview of VeryIDX - A Privacy-Preserving Digital Identity Management System for Mobile Devices. *Journal of Software*, 4(7):696–706, 2009.

[31] M. O. Rabin and J. O. Shallit. Randomized algorithms in number theory. *Communications on Pure and Applied Mathematics*, 39, 1986.

[32] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, pages 161–174, 1991.

[33] I. Teranishi, J. Furukawa, and K. Sako. k-Times Anonymous Authentication (Extended Abstract). In *ASIACRYPT*, pages 308–322, 2004.

[34] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. PEREA: Towards Practical TTP-free Revocation in Anonymous Authentication. In *CCS*, pages 333–344.

[35] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. Blacklistable anonymous credentials: Blocking misbehaving users without TTPs. In *CCS*, pages 72–81, 2007.

[36] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. Blacklistable Anonymous Credentials: Blocking Misbehaving Users without TTPs. In *CCS*, pages 72–81, 2007.

[37] P. Verhaeghe, K. Verslype, J. Lapon, V. Naessens, and B. D. Decker. A mobile and reliable anonymous epoll infrastructure. In *Mobisec*, 2010.

[38] K. Verslype, P. Verhaeghe, J. Lapon, V. Naessens, and B. De Decker. Priman: a privacy-preserving identity framework. In *Data and Applications Security and Privacy XXIV*, pages 327–334, 2010.

[39] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA: Using Hard AI Problems for Security. In *EUROCRYPT*, pages 294–311, 2003.