create tighter regulations for privacy and enforce larger penalties for non-compliance [2]. Another solution involves a new programming language that enforces privacy policies by constraining certain variables that correspond to user data [3]. These are good solutions, and they are important steps towards giving the client more control. One of the biggest issues with the current state of privacy, though, is that even if a user edits their privacy settings, often they are still left unclear as to how their data is managed, and frankly can't tell if it was mismanaged anyway. Figure 1 below shows how clients lose their control after sending it o to a service.

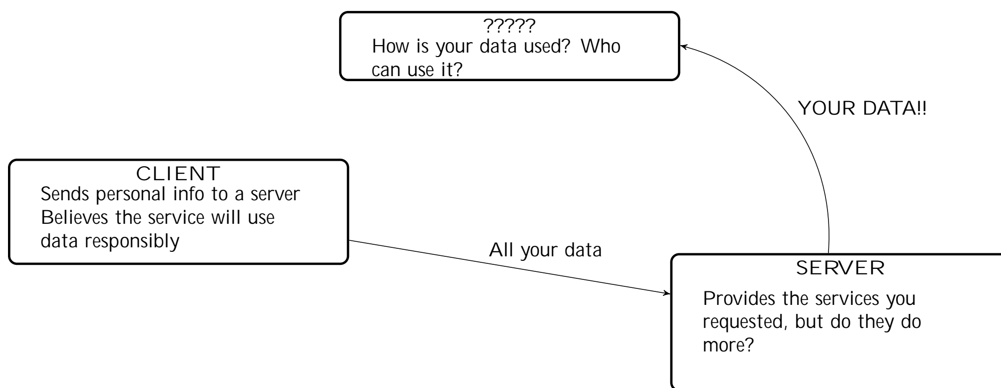

Figure 1: Current Client/Server Relationship

The approach investigated here will not go so far as to ensure the data is used properly, but it will create a `promise' that the service will implicitly make with every instance of the data being used. This scheme uses a type of Key-Policy encryption called Attribute-Based Encryption. It requires a client to share its data with a server, which will perform some action with the data, and a third-party authority to create, store and distribute keys necessary for the data to be encrypted and decrypted. The prototype for such a system has been built as a web app called privateBook. In privateBook, clients create an account and set a privacy policy, then can write posts that they can view if signed in to their own account. The prototype shows a practical way of ensuring data is encrypted for all parties, and can only be seen as a result of the service adhering to a user's privacy policy.

2 Data Encryption

At a high level, At Encryption (ABE) is a form of Public-Key cryptography which encrypts data with an access or policy. A party attempting to decrypt the data must present an atob4bute list along with the private key. This atobibute list must satisfy the access tbee of the encrypted data in order for decryption to occur. We will discuss exactly how ABE uses policies and a to encrypt and decrypt data, and then how the privateBook implementation uses this type of encryption.

2.1 Public-Key Cryptography

In many systems of encryption there is one key that is used to encrypt and decrypt messages, called a cipher. This is analogous to one key that may lock or unlock a door. Public-Key cryptography is also called asymmeob4c cryptography, due to its use of two separate keys, one used to decrypt and one used to encrypt a message. In this form, there is both a public key and private key. The public key is used to encrypt data, while the private key is used to decrypt. The keys are generated with some function that creates a public key which does not allow a foreign party to discover the private key. This is based on certain problems which are essentially impossible to solve computationally, such as large integer factorization problems.

2.2 Atob4bute-Based Encryption

There are four portions of the ABE scheme: the parameoer setup, encryption, key generation, and decryp

Setup The setup simply generates random groups, which are stored as PK, the public parameoers, and MK, a preliminary version of the private key. PK is the ABE version of a public key. The two keys here are bilinear group generators raised to randomly chosen exponents. These are stored as Python dictionaries.

Encryption Encryption uses PK, , the access structure, along with the string message meant to be encrypted, M . A dic E is created, which is the encrypted cipher text. , a string of a boolean expression, is the policy that must be satis ed for decryp

Key Generation   Key Generation uses a list of attributes, A, and MK to generate a decryption key, D. This is the ABE version of a private key, used to decrypt. A is the list of attributes that must satisfy   in order for D to successfully decrypt E.

Decryption   Decryption takes E, the ciphertext containing , PK and D. It applies D to E in order to decrypt the message. If A satis es , in the sense of the attributes ful lling the boolean expression included in D, the message will decrypt.

2.3 Example Usage

An example following the paper describing ABE's ne-grained access capabilities can demonstrate how this system would play out if used in exactly the way described above [5].
If there were a system for storing activity logs on a network, this data would need to be protected by an encryption scheme that would vary for di erent types of information and di erent types of activity. Here, ABE encryption would be useful, as each log could be stored with a speci c access policy: say "user is Bob or Alice AND the date is between September 2010 and May 2014 AND the activity is related to updating or changing the nancial information of projects". This information would then be encrypted with this policy. Anyone investigating the logs could then be given a secret key with a speci c access list with their properties: user name, title, security clearance, activity type, date, etc. ABE would then only allow the analyst to access information if their information t with the policy: all other data, which is not pertinent to their work, will be unaccessible.

3 ABE Privacy Prototype

The implementation of ABE is provided by Charm, a Python framework providing many di erent crypto systems [4]. It is used to power the encryption in privateBook. This can apply to any service which uses client data to perform some function, whether it be like Facebook, just storing and displaying messages, text or multimedia, or something like Google Maps, which uses client location info to display a map or directions. In a sense, the prototype applies ABE in a backwards fashion: rather than encrypting with a policy, and

users attempt to decrypt by presenting their attributes, here the attribute list is created and clients present their policies. Figure 2 displays the process.

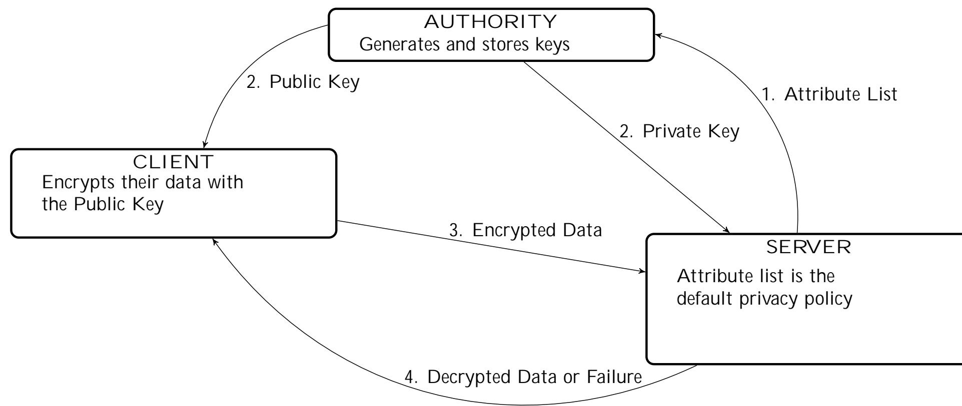

Figure 2: Prototype encryption/decryption process

3.1 Parties Involved

In privateBook, there are three relevant parties involved: the server, which provides the service, a client, who uses the service, and the authority, a third party that creates, distributes and stores the keys for both parties. The server creates default settings for privacy, saving them as an attribute list.

1. Server Registration This is the rst step, and only happens once. The server registers with the Authority, naming itself and providing a list of attributes, A, which is a list of the default values created for a privacy policy. The Authority stores A.

2. Key Distribution This happens once, in three parts. The Authority generates PK and MK, the public parameters and master key, which are used to generate D. A, PK and D are stored by the Authority. Then, D is sent to the server, which will use it to decrypt data, and PK is sent to the client, who uses it to encrypt data.

3. Client Encryption This will happen with every client request. The client encrypts by providing their data, , their personal privacy policy, and PK. This yields E, a cipher text, which is sent to the server. The server then stores the data as a cipher text.

4. Server Response This happens after every client request. The server stores the encrypted data, and attempts to decrypt it by applying the de-

table, and a homepage, where the client can write their updates and have them displayed; each update is stored by the server as a cipher text with a user ID in a Posted Data table. We will demonstrate each of the four steps in the privateBook implementation.

4.1 privateBook Server Registration

The privateBook server registers its name and default privacy settings with the Authority, which saves them as an attribute list, and then creates PK and D, the public parameters and private key. Figure 3 shows the printed values created from the default privateBook privacy values.

```
The Attribute List is:
[illegible]
The public parameters PK are:
[illegible]
-----------------------------------------------
The private key D is:
[illegible]
```

Figure 3: Authority values (Attribute list, PK, D)

These default values are based o the client privacy policy creation page, which is shown in Figure 4.

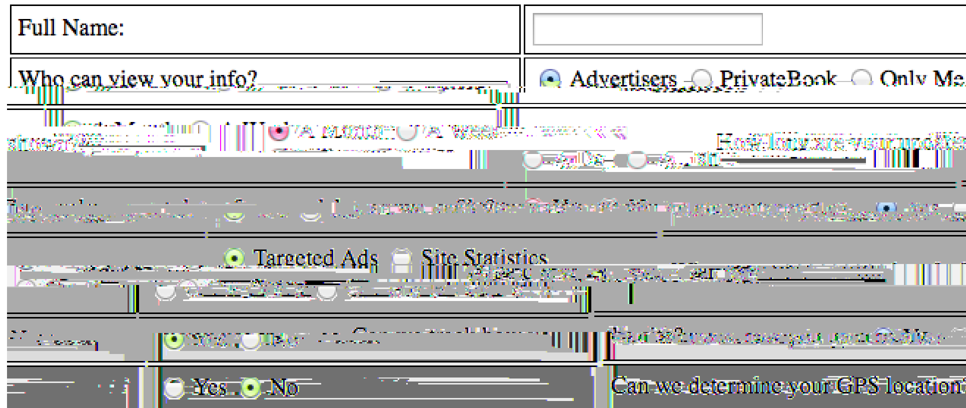

Figure 4: Client Policy creation page

4.2 Authority Key Storage and Distribution

The Authority then serializes the dictionary keys and stores them, shown in Figure 5.

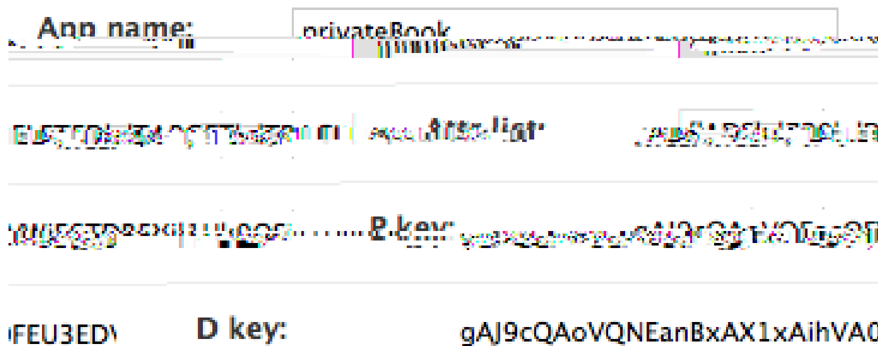

Figure 5: Authority key storage

These keys are then provided to the server and the clients by way of database queries to the Authority table. In a real distributed system, the Authority would send back D to the server, which would store it on their own. The clients would then receive PK upon registering for the service, an added complexity that this prototype does not delve into.

4.3 Client Privacy Policy Creation

The client, to become a privateBook user, registers themselves with a name and their privacy policy, as shown in Figure 4. They are then given their own ID, simply the order in which they were created, as their password, and username of their full name. These design choices are meant to simplify the account creation process: the focus of the prototype is not concerned with creating users as much as maintaining the privacy of their data, so privateBook has foregone security measures that should be in place in practical applications. Two policies are shown in Figure 6. They are strings of boolean expressions. The rst is a policy that has the least privacy restrictions as given by the service, and the second has the strictest settings. The rst includes an `or' for every type of privacy setting which includes all possible attributes; this allows for any default setting in the attributes list to satisfy the policy. The second, stricter policy includes the fewest attributes, making it the most di cult to satisfy.

```
The user policy is:
[illegible]

The user policy is:
((ME) and (1) and (SITE) and (KEEP) and (NOTBACK) and (NOLOCATE))
```

Figure 6: Least strict and most strict user privacy policies

This method of creating a default attribute list was to allow the server to fail. In practice, a service that intends to have wide use should never fail, and thus an early idea was to simply let each attribute be the type of privacy setting it was, and store the values elsewhere. Then the server would still need to decrypt the data and thus inherently read the privacy policy, but that would o er no real di erence from what exists currently, as the actual privacy settings would still be stored somewhere and potentially ignored, as they are now. So a di culty arose in nding an appropriate method to allow one attribute list to successfully satisfy many di erent policies. This method was chosen because it is based on an idea of having di erent levels of privacy, where a baseline is chosen by the service and clients can be more or less strict; more, rendering the service useless for that user, and less, allowing for full use.

4.4 Client Encryption and privateBook Response

The client then makes a status update, much like a Facebook one, and submits that. The data is encrypted with the update, a string, along with the user's privacy policy and the public key PK. This results in a long cipher text, which is stored in privateBook's database as a serialized dictionary. The results of this process are shown in Figure 7. E is stored in the database, then the page is reloaded, displaying all of the user's updates. The server, upon loading of the user's homepage, rst attempts to decrypt their data. If it is successful, the date, time and actual content are shown. If it is unsuccessful, the content is replaced with: \Your status could not be displayed: this service does not support your privacy policy!" Thus this service will not work for the client if their privacy settings are too strict, and therefore their policy is not satis ed by privateBook's default attribute list.

5 Issues with the Prototype

Though the privateBook prototype o ers a solution, there are, as can be expected, some faults. The biggest issue is that the encrypted data, E, is

Acknowledgements

Professor Robert Muller and Stefan Saroiu at Microsoft Research

References

[1] Lawler, Ryan. "QuizUp Sends Personal User Info To Strangers, Company Says Bug Contributed To Weakened Security." TechCrunch, 25 Nov. 2013.

[2] Bajaj, Vikas. "Imagine if Companies Had to Ask Before Using Your Data." Taking Note. The New York Times, 13 Mar 2014.

[3] J. Yang, K. Yessenov, A. Solar-Lezama. A Language for Automatically Enforcing Privacy Policies.POPL 2012.

[4] Akinyele, Joseph A. and Garman, Christina and Miers, Ian and Pagano. \Charm: a framework for rapidly prototyping cryptosystems." Journal of Cryptographic Engineering3.2 (2013): 111-128.

[5] J. Bethencourt, A. Sahai, B. Waters. Ciphertext-Policy Attribute-Based Encryption.