Encrypting User Data with Attribute-Based Encryption Using Privacy Policies

Milo Watanabe
Boston College

April 2014

Abstract

As the internet and cloud services have pervaded our lives in nearly every aspect, one of the biggest issues facing the populace today is finding the best ways to protect our privacy. An important part of protection for the average user of a service is their privacy policy: essentially the only way today to let a client define how their data is used by the server. But often a client is not given a choice, and sometimes their policy is not even followed. We present here an way of encrypting user data such that they first create a privacy policy, and their data is protected from even the service unless their privacy requirements are met by the service.

1 Introduction

Protecting user privacy is a huge problem facing any internet company, and a huge issue for every client using those technologies. Tech giants, like Facebook and Google, and smaller app developers, like QuizUp publisher Plain Vanilla, have faced a large amount of backlash based on their sometimes-shady usage of client data.[1] More and more solutions are emerging for giving users finer control over their data, though.
Recently in the EU, a push has gone through the European Parliament to

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

create tighter regulations for privacy and enforce larger penalties for non-compliance.[2] Another solution involves a new programming language that inherently puts constraints, defined by privacy policies, on certain variables that correspond to user data.[3] These are good solutions, and they are important steps towards putting the client first. 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.

The solution presented 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 and store the 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 which clients create an account and set a privacy policy, then can write posts that they can view if signed in to their own account. This simple demonstration serves to show 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, Attribute-Based Encryption (ABE) encrypts data with an access tree, or policy, and a party attempting to decrypt the data must present an attribute list, which will successfully allow decryption provided the attributes satisfy the tree. We will discuss exactly how the service uses policies and attributes to encrypt and decrypt data, and then how the privateBook implementation uses this type of encryption.

[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.*

2.1 Attribute-Based Encryption

The implementation of ABE is provided by Charm, a Python framework providing many different crypto systems. There are four portions of the ABE scheme: the parameter setup, encryption, key generation, and decryption.

Setup The setup simply generates random groups, which are stored as PK, the public parameters, and MK, the private key. These are later used for encryption/decryption, and are stored as Python dictionaries.

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

Key Generation Key Generation uses a list of attributes, $\mathbb{A}$, and MK to generate a decryption key, D. $\mathbb{A}$ is the attributes that must satisfy γ in order for D to successfully decrypt E.

Decryption Decryption takes E, the ciphertext containing γ, PK and D. It applies D to E in order to decrypt the message. If $\mathbb{A}$ satisfies γ, in the sense of the attributes fulfilling the boolean expression included in D, the message will decrypt.

An example following the paper describing ABE's fine-grained access capabilities can demonstrate how this system would play out if used in exactly the way described above.[4]
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 different types of information and different types of activity. Here, ABE encryption would be useful, as each log could be stored with a specific 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 financial 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 specific access list with their properties: user name, title, security clearance,

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

activity type, date, etc. ABE would then only allow the analyst to access information if their information fit with the policy: all other data, which is not pertinent to their work, will be unaccessible.

2.2 The privateBook Prototype