Interim Report

Henry Mortimer

January 2017

1 Introduction

A public key infrastructure (PKI) is a system designed to create, manage, distribute and authorise certificates for cryptographic public keys. There are currently two predominant approaches to designing a PKI.

Certificate Authorities A certificate authority based system uses hierarchical approach where certain trusted entities are designated as certificate authorities (CA) who can verify the identity of a user (Alice) then issue a certificate attesting to this. A separate user (Bob) can then verify the identity of Alice if the certificate is signed by a CA Bob trusts or a CA Bob trusts has delegated authority to the CA who signed Alice's certificate.

Web of Trust A web of trust opts for a more decentralized approach. Any user in the system can create certificates for any other user in the system. If a user Alice wants to verify the identity of a user Bob and Alice has a certificate for Bob signed by Charlie if Alice trusts Charlie to identify users then Alice trusts the identity of Bob.

Problems There are problems with both of these approaches. With certificate authorities there is a single point of failure in the system. If a certificate authorities private key is compromised by an attacker then they are able to issue certificates as if they are the certificate authority enabling them to impersonate the identity of anyone to people who trust the compromised CA.

In a web of trust users can be faced with the problem of new users to the system to be able to get enough other users to endorse their identity as authentic. This leaves them unable to participate in the system as no other users will trust their public key.

Solution My proposed solution is a hybrid of these two systems. A certificate authority will be a well known entity that many users will trust. Users can request the CA to issue a certificate in the same way as normal. Similarly users can create certificates attesting to the identity of other users.

The system will be backed by a distributed hash table (DHT) which will be used to store and retrieve certificates and public keys. Every application using the system will also be a node in the DHT. As the responsibility of storing and locating data is distributed amongst the nodes gives good fault tolerance and scalability properties. Whenever a user or a CA creates a certificate it will be stored in the DHT.

Each user will have two lists of public keys they trust. One of CAs and another of users they trust to verify the identity of other users. When verifying the identity of a public key the system will retrieve all certificates for the key and first see if any where issued by another trusted user before trying to form a certificate chain to a CA.

2 Completed Work

2.1 Design

I have designed the system to be fairly modular. The DHT is responsible simply for storing and retrieving values indexed by keys. It makes no effort to parse the data retrieved from the network, this is left to other modules. Then there are two modules concerning keys and certificates. Each will have a local storage mechanism and will be able to insert and retrieve data from the DHT and parse the data from the DHT into python objects. One module will verify the identity of a public key and can use the key and certificate modules to retrieve data as needed. Finally there will be a control module which listens locally on a socket so other applications can use the system to verify the identities of public keys.

2.2 Dev Ops

Due to the nature of distributed systems to adequately test and evaluate one you need multiple machines. To overcome this I have a virtual machine image I can clone so I have as many nodes as I need. I also have an ansible script to automatically download and install the project and dependencies on all the VMs. The code is all under version control at `https://github.com/hennersz/finalYearProject`. The project is being implemented python and I have setup the project to use the python packaging system so the project and dependencies can be easily installed by others.

2.3 Implementation

So far I have implemented the DHT using a python library for kademlia. I have created my own storage module for the DHT such that if more than one item is inserted with the same key a list of values are stored for that key rather than overwriting the old value. This allows multiple certificates to be stored for one public key for example. I have then created a wrapper around the DHT server object that sets up the connection and automatically deserialses the data extracted from the DHT.

2.4 Testing

For testing I am using the pytest library. I have written unit tests for the modules completed so far. I use tox to manage running the tests and handling the python virtual environment. I also run the travis CI system.

3 Remaining Work

3.1 Implementation

3.1.1 Certificate Module

- Create certificates - 1 week
- Insert certificates - 1 week
- Retrieve and parse certificates - 1 week

3.1.2 Key Module

- Create keys - 1 week
- Add GPG support - 2 weeks
- Insert keys - 1 week
- Retrieve and parse keys - 1 week

3.1.3 Verification Module

- Find key identities - 2 weeks
- Verify a specific identity for a Key - 3 weeks
- Verify using web of trust - 3 weeks
- Retrieve and verify certificate chain - 4 weeks

3.1.4 Control Module

- Local socket listening for connections - 2 weeks
- Syntax for commands - 3 weeks
- Interface to verification and key modules - 3 weeks

3.2 Evaluation

To evaluate the project I am going to test both the performance and security of the system. For the performance evaluation I will test the time it takes to verify the identity of a key on local networks and across the Internet, identify any bottlenecks an suggest ways to improve it. For the security evaluation I will assess the feasibility and potential damage of attacks on the system such as if attackers control nodes in the DHT. I expect to complete the evaluation within the next 8 weeks.