MSc in Computing (Communications Software)

# A Distributed System for Storage of Trust Management Credentials

*By*
Eric Robson

Waterford Institute of Technology
Department of Computing, Mathematics and Physics

January 12, 2009

# Table of Contents

# 1 Introduction

The purpose of this document is to provide an update on the progress of the dissertation, and to serve as an update to the original dissertation proposal. In particular I will be addressing issues highlighted in the feedback received, specifically concerning how to evaluate the solution.

The dissertation will attempt to address a significant problem identified in the area of trust management. The problem centrers on finding a safe, efficient distributed system for storage of trust management credentials.

# 2 Literature Review

The following section describes a detailed literature review of the application area. This dissertation will use technology from peer-to-peer systems to solve a problem in trust management, this combination of two separate technologies is reflected in the structure of this section. Each technology area is first discussed separately, and then we shall explore previous attempts at combining them.

## 2.1 Overview of Trust Management Systems

Ioannidis and Keromytis [8] give a very good introduction to trust management systems in a chapter of the book 'The Practical Handbook of Internet Computing'. The two authors are very well credited in this field having published papers[3, 1, 4–6] on trust management systems since 1998, many of these papers in collaboration with Matt Blaze and other well know researchers in this area.

The chapter gives a couple of sections to an overview of access control and how traditional authentication systems work, it then proceeds to explain the limitations of these authentication systems. In explaining these limitations the reader gets a good insight as to why and how distributed trust management systems evolved.

The authors proceed to give detailed descriptions of a number of distributed trust management systems, including POLICYMAKER[2], KeyNote[3] and the REFEREE[7] system. The final section in the chapter details application areas where trust management systems are used.

## 2.2 Assessment of Trust Management Systems

**PolicyMaker** PolicyMaker[2] is the first example of a comprehensive distributed trust management system. It was defined as a prototype implementation of trust management. The system that was defined is a query system that when supplied with enough information about a request, can make a recommendation to the calling application.

PolicyMaker defines a protocol in which to specify queries, policies and credentials. Policies and credentials are almost equal, they both delegate trust from one entity to another, but the difference being that policies are unsigned and

are treated like root certificates that are implicitly trusted. Signed credentials have to be signed by an entity that has been delegated trust from a policy or have a hierarchical link to a policy. A PolicyMaker assertion is defined as `'Source ASSERTS AuthorityStruct WHERE Filter'` where source is the identity (public key) of the entity making the assertion, or 'POLICY' in the case of a policy. AuthorityStruct is the key(s) of the identities that trust is being applied to. And Filter is the conditions where this assertion is valid.

AuthorityStruct and Filter can be defined to be quite complex pieces of logic, for example the AuthorityStruct could be defined as 'at least 2 keys must be present and one must be key A' and the Filter could defined as 'grant access where amount requested is less that 5000 dollars' or where 'sender=bob and IP=10.1.1.1'. It is important to note that the PolicyMaker system doesn't need to know anything about the details (I.e. what sender or IP mean) in the assertion, these are application specific key value pairs that are specified with a request. The PolicyMaker specification does not restrict users to program an assertion in a particular language, that detail is left to the implementer. It does however recommend that implementers use a safe I/O and resource limited interpreted language; the prototype includes an implementation of regex and AWK.

A PolicyMaker query is defined as `'key1,key2,....,keyn REQUESTS`ActionString'— where an application on behalf of the specified one or more keys can query the PolicyMaker system to determine an access recommendation for the action defined in ActionString. An example of an action string might be 'sender=bob IP=10.1.1.1' The PolicyMaker system processes the query through all of its assertions until it can find an assertion that confirms the query. If it finds an assertion it will make a recommendation to approve, if it doesn't it will make a recommendation to deny.

Blaze, Feigenbaum and Lacy have been credited with coining the term "trust management' in their 1996 paper "Decentralised Trust Management'[2]. This paper is seen as a key paper in this field of research as it takes a fresh look at application security in decentralised systems. Even though this paper was written in 1996, its review of existing approaches to system security (in particular X.509[9] and PGP[15] certificate systems) is still quiet relevant.

**KeyNote** KeyNote[3] for all practical purposes is a re-design of PolicyMaker. It follows on from PolicyMaker's fundamental principles, but attempts to fix a number of it's shortcomings. PolicyMaker tries to be too broad and generic; it doesn't define a language to write assertions in and the most popular implemented language, AWK, has been argued to be too heavy weight and complex for a trust management system. To address this shortcoming KeyNote defines a language to define (unsigned) policies and (signed) credentials. This is seen as advantageous as the

> "the security configuration mechanism for one application carries exactly the same syntactic and semantic structure as that on another, even when the semantics of the applications themselves are different."[3]

This assertion language allows an assertion author to specify a range of compliance values that a request conforms to. For example if KeyNote was queried about access to a particular file, a recommendation for one requester might be read only or read/write for another requester. This significantly reduces the amount of assertions that need to be written. A KeyNote assertion is semantically similar to a PolicyMaker assertion, but syntactically very different. A KeyNote assertion takes the form:

```
KeyNote-Version: 2
Authoriser: A single principle (public key who is delegating
authority, or POLICY)
Licensees: Zero or more principles who is receiving authority.
Conditions: clauses, and return values for the clauses e.g:
(@(IP) = 10.1.1.1)  read
(@(IP) = 10.1.2.1)  read/write
Signature: The credential is signed with the authoriser's private
key.
```

KeyNote queries are specified as:

```
 _ACTION_AUTHORISERS = list of principles asking query.
name, value pairs. e.g. IP=10.1.1.1
```

The KeyNote system has been used in many trust management implementations.....

is

### 2.3 Overview of Peer-to-Peer

Peer-to-peer networks come in two flavours, structured and unstructured. With unstructured networks, nodes are arbitrarily connected and data can be stored on any node. This type of network is generally used for file sharing and examples of it are Gnutella and FastTrack. When a user of an unstructured peer-to-peer network wishes to search for a file, the request is flooded to all nodes on the network in an attempt to find as many nodes as possible that share the file. Due to the size and complexity of these networks a response is not necessarily guaranteed.

Structured peer-to-peer networks are less ad hoc than the unstructured variety, while they still provide for nodes joining and leaving of their own accord, each node is assigned an identity and explicitly told what information it can store. They are generally used to distribute objects evenly amongst nodes. The peer-to-peer algorithm decides what object is stored on a node, although depending on the algorithm this can be influenced. Each structured peer-to-peer algorithm provides a routing mechanism to efficiently retrieve a stored object. In this case flooding of the network is not required and a response from the network is guaranteed.

### 2.4 Assessment of Peer-to-peer Systems

Most structured peer-to-peer algorithms employ a distributed hash table (DHT) to provide a unique key to identify objects and nodes. The DHT forms the basis of the routing protocol. Examples of a structured peer-to-peer network are Chord[13], Pastry[12], Tapestry[14] and the content addressable network (CAN)[11]. These four algorithms were released concurrently as proposals for a next generation of DHT based peer-to-peer systems. With the exception of CAN, these algorithms rely on the concept of *consistent hashing* to organise the node space.

Consistent hashing works by assigning an ID to each node, the ID is a hash of a unique attribute of the node (e.g. its IP address), this provides a uniform distribution of node ids throughout the node space. These nodes are then arranged numerically in a circle, giving rise to the peer-to-peer ring. When an object is submitted to the peer-to-peer ring for storage, a hash is computed of the object (using the same hashing function as before) to produce an object id. The storage node is identified by finding a node id that is numerical closets, but greater than the object id.

While each algorithm conforms to a core set of requirements like efficient routing and nodes joining and leaving arbitraryly, the differences emerge in how the routing tables are defined and how they are queried. The following section details how each of the algorithms work.

**Pastry:** Rowstron describes Pastry as "a scalable, distributed object location and routing substrate for wide-area peer-to-peer applications"[12], essentially it is a structured peer-to-peer network overlay based on a distributed hash table (DHT).

In a simple implementation each node only knows about its immediate neighbours, with this implementation an object might have to pass through every other node in order to be stored at the last node in the ring. However pastry specifies a more comprehensive routing algorithm than this, it claims that the expected number of routing steps in *O(log N)* where N is the number of pastry nodes. Pastry achieves this by maintaining three tables, a routing table, a neighbourhood set and a leaf set, each node is responsible for maintaining its tables.

Each row in the routing table represents a list of nodes that share the same prefix of digits in the node id, for example row one contains a reference to any node which shares the same first digit in its node id, row two shares the first two digits, etc... The structure of this table means that the node maintains a finer grained list of nodes that are (numerically) closer to it than ones that are more distant

The neighbourhood set consists of a set of nodes that are close, according to the proximity matrix (i.e.. exist within the same network or physical location). This set is not used for routing, but more for protection against network partition and redundancy. Pastry has inbuilt functionality for replication of data between

nodes. In order to support this the node maintains a leaf set, which is a reference to nodes that are its numeric neighbours.

When a node receives a request to store and object, the node must first compute the hash of the object to produce an object id. The object id is then compared to the elements in the leaf set, and if no match is found it is compared against the node's node id to calculate the length of the common prefix. This can then be used to find the nearest node in the routing table. When the node has identified the next hop, it sends to the object to this node.

**Tapestry:** The Tapestry system was defined by Zhao et al [14] but is based on a concept known as a *plaxton mesh*[10]. The plaxton mesh is a similar concept to consistent hashing (both systems were published in close proximity in 1997) in that they both have unique identifiers for nodes and objects inorder to locate the correct node to retrieve an object from. They differ in that the plaxton mesh does not specifically use hash functions to generate identifiers for the nodes and object, and that the plaxton mesh was designed to handle multiple copies of the same object and to provide the most efficient routing to the nearest copy.

The plaxton mesh method of routing is similar to that described in the Pastry section above in that each node maintains a prefix based routing table. This table contains an entry for the closest node(based on network distance) sharing $i$ amount of common digits in its node ids prefix (where $i$ is an integer between one and the length of a node id).

Zhao et al noted that the plaxton mesh had some limitations in that "it is a static data structure without node or object insertions and deletions"[14, page 4]. This problem centres on that the plaxton mesh relys on a piece of global information (that is initialised at boot time) to create the node and object ids. If a new node or object is added all the node and object ids need to be regenerated.

Tapestry's improvements remove this dependency on global information, it introduces an algorithm called *surrogate routing* to incrementally compute a unique id. Tapestry also needed to define methods for sharing new routing information and neighbour tables when a node joins (methods previously not required in the plaxton mesh).

**Chord:** Stoica et al[13] specifies the Chord system that centres around the idea of consistent hashing. Similarly to Pastry consistent hashing is used to produce a node space; this node space can be visualised as a ring where each node represents a place on the ring. Each node is then responsible for the segment of the ring between it and the node previous (or anti-clockwise) to it.

The Chord system specifies a routing table called the *finger table*. The finger table comprises of $i$ entries where $i$ is no greater than the length of the hashes generated by the hashing function. Each entry in the table contains the identity of the next node (clockwise) in the ring that succeeds the current node by $2^{i-1}$. This provides the node with a routing table which stores the nodes (numerically) closest to it in finer detail to the nodes that are more distant. This property

allows all routing requests to be completed in O(Log N) where N is the number of nodes.

In order to handle nodes leaving and joining arbitraryly, the node needs to know where in the ring it is positioned. To do this it must maintain two pointers, a pointer to the node immediate before it (the predecessor) on the ring and a pointer to the node immediately after it (successor) on the ring. The successor pointer is a given as it is always the first entry in the nodes finger table. The predecessor pointer is more complicated as the finger table only operates in a clockwise direction around the ring. To populate the predecessor pointer the current node relys on the predecessor node to inform the current node that it is the current nodes predecessor.

**Content Addressable Network (CAN):** Ratnasamy et al describes the CAN system "as a distributed infrastructure that provides hash table-like functionality on Internet-like scales"[11, page 1]. They advertise this system as a replacement for large scale unstructured peer-to-peer systems like Napster[**?**] or Gnutella[**?**].

The CAN design centres around a virtual d-dimensional Cartesian coordinate space, where the entire coordinate space is partitioned amongst all the nodes in the system. Each node in the system is a wear of its location in the coordinate system and it maintains a reference to nodes in neighbouring sections of the coordinate system. When an object is requested to be stored, the system calculates within which coordinates the object is to be located using a uniform hashing function. CAN then uses this information to route messages to specific coordinates.

This coordinate system also enables a fault tolerant routing system, as each node will have multiple neighbours; if a neighbour node is lost, messages can automatically be routed via another neighbour.

When a node requests to join a CAN network it must bootstrap itself against another active node in the network. The bootstrap node will then send out a message to its neighbours to find out which one of them has the largest section of the coordinate system. The largest section is then split in two and one half is given to the joining node. This enables an even distribution of sections within the node space.

### 2.5 Overview of similar systems

In 1976 Diffie and Hellman[**?**] recognised that one of the problems facing cryptography was securely sending a private key to the recipient of an encrypted file in order for them to decrypt the file. In this paper they proposed public key encryption which drastically revolutionised cryptography. In this paper they also proposed that these public keys could all be stored in a single repository; where anyone, given a name, could look up that persons public key.

While Diffie and Hellmans public key cryptography was a roaring success the public file they proposed was never implemented. Kohnfelder in 1978[**?**] took this public file concept further when he introduced the idea of an identity certificate, which is a certificate that binds a public key to an identity. He proposed

that the certificates should be stored in the public file and that they could be signed by the public file to prove their authenticity. 10 years after Kohnfelders work the X.500 Directory specification was released which again was another system designed for the storage of digital certificate. Due to design flaws this system was never completed and was replaced by the X.509 Public key Infrastructure[**?**]. This system was implemented and there are many X.509 systems in place today, mostly commercial systems like VeriSign[**?**]. However the X.509 system is a certificate signing system, it does not provide a facility for storage of certificates.

As we can see, credential storage has been a problem since the beginning of public key encryption.

**Keypeer:**

**ConChord:**

### 2.6 Literature Review Conclusion

This section gave a detailed review

## 3 Research Question Re-visited

In light of the feedback received from the dissertation proposal I have re-worded the research questions.

1. What are the challenges in building a safe, efficient and distributed system for storage and retrieval of trust management credentials and how can these be overcome?
2. What modifications would be required to expand this system for the efficient storage of credential revocations?

## 4 Research Methodology

In the dissertation proposal I submitted a work plan that largely followed the waterfall methodology, this was criticised in the feedback received. In practice I have learnt that this methodology does not relate to this kind of research project. This project more lends itself to an agile development model, where the requirements are continually changing as new ideas are experimented with. With this in mind a new more iterative project plan is specified here.

### 4.1 WP1: Analysis. Expected Completion: 200901

1. **Action:** Choose trust management system (expected duration: 2 days).
2. **Action:** Choose peer-to-peer overlay network (expected duration: 2 days).
3. **Action:** Choose development language and database system (expected duration: 2 days).
4. **Output:** A clearly defined plan of action.

### 4.2 WP2: Analysis of Third Party Software. Expected Completion: 200902

1. **Action:** Analyse and set-up trust management implementation. (expected duration 2 weeks).
2. **Action:** Analyse and set-up peer-to-peer overlay network (expected duration 2 weeks).
3. **Output:** A working development environment. days).

### 4.3 WP3: Integration. Expected Completion: 200904

1. **Action:** Design bridging software that will allow the peer-to-peer system access to the chain of credentials within the trust management system (expected duration 2 weeks).
2. **Action:** Implement bridging software (expected duration 3 weeks).
3. **Action:** Refine design and implementation of bridging software (expected duration 1 week).
4. **Output:** A basic working implementation to be used for benchmarking software.

### 4.4 WP4: Refinement of Design. Expected Completion: 200907

This task will be run in a number of cycles, each time with new refinements to the design. Note the durations specified below are estimates and some iterations may take longer that others.

1. **Action:** Research new concepts and existing solutions (expected duration 4 days).
2. **Action:** Refine design of system, targeting bottlenecks identified from previous result sets (expected duration 3 days).
3. **Action:** Add refinements to code base (expected duration 3 days).
4. **Action:** Run benchmarking trials against modified system and identify bottlenecks (expected duration 3 days).
5. **Action:** Compile report based on trials. (expected duration 1 days).
6. **Output:** Summary documentation on new concepts with reports of how they effected the system.
7. **Output:** An improved implementation.

### 4.5 WP5: Final Write Up. Expected Completion: 200908

1. **Action:** Compilation of notes and result sets. (expected duration 4 days).
2. **Action:** Dissertation write up. (expected duration 26 days).
3. **Output:** A Taught Masters dissertation.

### 4.6 Testing and Validation

## 5 Achievements To Date

Significant progress has been made on this project both in terms of the research and software development and as a whole this project is on schedule as per the revised project plan.

### 5.1 Research Milestones Acheived

Extensive research has been done so far in this project in the areas of trust management and peer-to-peer systems and in systems where both technologies are utilised. This research has given a firm basic understanding of the technologies being used. As a result the following decisions have been made:

1. KeyNote has been chosen as the trust management system to work with. This decision is based on a number of factors:
   (a) The complexity of the KeyNote credential is sufficient enough.
   (b) There is a good freely available java implementation of KeyNote called JKeyNote[**?**].
   (c) A similar system to this has not been attempted before with a KeyNote credential
2. Pastry has been chosen as the peer-to-peer overlay network. This decision is based on a number of factors:
   (a) Pastry has direct support for storage of object in a distributed system with in built support for replication to sustain the required levels of redundency.
   (b) A specfication for a file based storage system (PAST[**?**]) has been designed for Pastry
   (c) A java based implementation is freely available called FreePastry[**?**] which includes an implementation of PAST.

### 5.2 Software Milestones Acheived

## References


1. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The keynote trust-management system, version 2. *IETF RFC*, 2704:164–173, 1999.
2. M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *IEEE Proceedings of the 17th Symposium on Security*, 1996.
3. Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. Keynote: Trust management for public-key infrastructures (position paper). In Bruce Christianson, Bruno Crispo, William S. Harbison, and Michael Roe, editors, *Security Protocols Workshop*, volume 1550 of *Lecture Notes in Computer Science*, pages 59–63. Springer, 1998.
4. Matt Blaze, Joel Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming: Issues in Distributed and Mobile Object Systems*, volume 1603 of *Lecture Notes in Computer Science*, pages 183–210. Springer Verlag, Berlin, July 1999.
5. Matt Blaze, John Ioannidis, and Angelos D. Keromytis. Trust management for ipsec. In *NDSS*. The Internet Society, 2001.
6. Matt Blaze, John Ioannidis, and Angelos D. Keromytis. Experience with the keynote trust management system: Applications and future directions. In Paddy Nixon and Sotirios Terzis, editors, *iTrust*, volume 2692 of *Lecture Notes in Computer Science*, pages 284–300. Springer, 2003.
7. Yang-Hua Chu, Joan Feigenbaum, Brian LaMacchia, Paul Resnick, and Martin Strauss. Referee: trust management for web applications. In *Selected papers from the sixth international conference on World Wide Web*, pages 953–964, Essex, UK, 1997. Elsevier Science Publishers Ltd.
8. John Ioannidis and Angelos D. Keromytis. *Practical Handbook of Internet Computing*, chapter Distributed Trust. Chapman Hall & CRC Press, Baton Rouge, 2004.
9. Kohnfelder. *Toward a Practical Public-Key Cryptosystem.* PhD thesis, MIT, Cambridge, Mass., 1978.
10. C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 311–320, New York, NY, USA, 1997. ACM.
11. Sylvia Ratnasamy, Paul Francis, Scott Shenker, and Mark Handley. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, pages 161–172, 2001.
12. Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Lecture Notes in Computer Science*, pages 329–350, 2001.
13. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. pages 149–160, 2001.
14. Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, 2001.
15. P. R. Zimmermann. *The Official PGP Users Guide.* MIT Press, Cambridge, MA, USA, 1995.