# A Self-Destructing File Distribution System With Feedback for Peer-to-Peer Networks

Jason Croft
Department of Computer Science
Boston College
Email: croftj@bc.edu

Robert Signorile
Department of Computer Science
Boston College
Email: signoril@bc.edu

***Abstract*—Maintaining control and ownership of data is difficult in any network environment. In peer-to-peer P2P networks, without a central authority to control access to data, other mediations must be proposed. We propose a method to control distribution of data in P2P networks through the use of a feedback-based, restrictive content distribution system. Unauthorized distribution is thwarted through a method of self-destructing, one-time-use data. Transmitted data is encrypted, encapsulated within an executable, and authenticated to a single user and machine. Once accessed, measures are taken to ensure it cannot be used outside the executable (e.g., displayed within a non-selectable, non-editable window) and that the executable cannot be easily decompiled. After a single use, data is destroyed through a method of in-memory compilation of a new executable which, during run-time, overwrites the original. Methods to prevent overwriting, such as removing write privileges, are treated as misuse of the data. Misuse of data, or unsatisfactory transactions, negatively affect a peer's trust on the network. Misuse can include failed authentication, unauthorized multiple uses, or attempted distribution of the data. The executable provides feedback to the sender based on this usage so trust values can be adjusted accordingly. We assume some pair-wise trust, and place emphasis on these pair-wise interactions. To compute trust values, we determiend that the most efficient approach, given the emphasis on each individual pair-wise transaction, is a modified Bayesian approach with weight given towards information gathered first-hand.**

## I. Introduction

Peer-to-peer (P2P) networks present many advantages over typical client-server networks. These networks are more scalable, lack a single point of failure, and are typically less expensive to deploy. However, like any type of network, ownership of data is not easily managed once it is distributed to another user on the network. That is, peer $i$ cannot ensure that a file sent to peer $j$ will not be maliciously distributed to peer $k$. Peer $i$ can use encryption to ensure only $j$ can initially decrypt file, but once decrypted, $j$ can send it to whomever it wishes. This is an essential part of the problem encountered in digital rights management (DRM). Solving this is more difficult in P2P networks than traditional server-client networks because of the lack of a central node with which to regulate file access. Previous research ([17], [6]) has implemented "persistent access control" in client-server networks through the use of watermarks, trusted hardware, and a certification authority. In this paper, we describe a method of first level protection against unauthorized distribution of confidential data for P2P networks that uses a feedback-based content distribution system. Distributed content provides feedback to the data owners based on the usage of the transmitted data, which is used to compute trust values of the data transaction. To compute these trust values, we modified a Bayesian algorithm.

For our architecture, we have focused primarily on applications to large enterprise networks that may require authentication. Universities utilizing LDAP authentication would be one such example, although any environment with the need for sharing of data would be applicable. Those with confidential information, in particular, would benefit most from such control. We define confidential distribution as data that can be distributed to a peer with the assurance that only the receiving peer will have the ability to use the data and cannot redistribute it to an unauthorized one.

We also narrowed our work to the distribution of binary data, such as text, HTML, or PDF files. Our reasoning for focusing on these types of data rather than media (*e.g.*, music or video) is due to the intended use of such an application in a corporate or enterprise network. The initial file transfer is assumed to be amongst trusted pairs of peers in the enterprise. Our protocol is designed to limit confidential data leaking from the trusted peer after this initial transfer. If there is an effort (accidentally or deliberately) to electronically send the file to another peer, the file will be effectively made unusable and feedback of this event will be provided to the original owner of the file. In this event, the trustworthiness of the the original receiving peer will drop (thus, limiting the willingness to share files with this peer in the future).

## II. Why P2P?

The design of our content distribution system is not reliant on P2P networks and can be easily adapted for client-server networks. However, we have chosen to implement it on P2P networks due to their scalability and to demonstrate that our idea is possible without requiring central authority. Additionally, the need to share data amongst multiple peers coincides with the strengths of P2P networks.

Roussopoulos et al., identified three characteristics of these types of networks: self-organizing, symmetric communication, and decentralized control [20]. They also describe five characteristics important in assessing the "P2P-worthiness" of dis-

tributed problems: budget, resource relevance to participants, trust, rate of system change, and criticality. The needs of our network coincide with the three defined characteristics. All peer-related information, including trust values or shared data, is discovered without any global directory. There is indeed symmetric communication, as a peer both provides or requests data, and control is completely decentralized in our environment. Budget is not considered to be an influential factor in our design, though resource relevance to participants is important. We assume that peers join the network to request or share data, so cooperation should evolve without any additional incentives.

However, Roussopoulos cites mutually distrustive peers as a factor against P2P networks because of the additional overhead and need for artificial economies of trading schemes to incentivize trading. While data sharing is fundamental to our environment, we believe there should be no additional incentives to share data, so we do not consider this a factor against using a P2P solution. Need for data should be the only purpose behind sharing. Since we are concerned most with controlled dissemination of needed data, peers should not be punished for lacking data that is relevant to other peers. Since they cannot share received data with others, only misuse of data should be punishable.

Roussopoulos also argues that systems with high rates of change are best deployed as non-critical applications and quick changing systems involving critical information would not benefit from P2P networks. Rate of system change will be variable: large enterprise networks may have higher rates of users entering and leaving than a smaller university network.

## III. Related Work

Distribution of content on our network is managed using a combination of authentication and self-destructing data. We have found one other example of self-destructing data in a patent for self-destructing emails [23]. The authors describe this design as "automatically [destroying] documents or email messages at a predetermined time by attaching a 'virus' to the document or email"'. We focused on a less intrusive method to avoid issues with anti-virus or anti-malware software.

Examples of enterprise software aimed at protecting against "endpoint-data loss" include NextLab's Enterprise DLP Data Protection suite [1] and RSA's Data Loss Prevention Endpoint [3]. However, these applications require a central controller to manage policies and add additional users.

Some work has proposed solutions to preventing spreading of malicious or harmful content [22] using trust-based access control, while other trust-based access control work includes [16], [5]. However, we have found no work on limiting data distribution in P2P networks and consider our work unique. Additionally, we note that our implementation, both the network application (which manages peer presence and available data for distribution) and data view are more than simple proof-of-concepts but fully functional applications. We believe this can be deployed in an enterprise-level network.

## IV. Architecture Overview

We have implemented a two-layer architecture: the first is a content distribution system that allows one peer to distribute data to another that can use, or view it, and the second is a feedback system that provides trust information of peers based on their usage of data. The main focus of our research is the self-destructive distribution system. The distribution system allows a peer to specify the usage of the data, enforcing a one-time-use (*i.e.*, read) policy. Additionally, it restricts usage by requiring authentication based on username and MAC address. Proper use of data increases a peer's trustworthiness, while improper use will decrease it. We define improper use of data as exceeding the set number of uses or a failed authentication, giving three cases:

- A peer exceeds the one allowed use
- A peer consumes the data on different machine or using a different username than authenticated
- A peer distributes the data to a non-authenticated user/machine

The focus of our work is on controlled distribution of confidential data in an enterprise-wide architecture. The data is a one-time use file that can only be accessed on the trusted peer's site (the original receiver of the file). If access is on a unauthorized site, the file self-destructs. To gain feedback on how the data is used (i.e. propagated in the enterprise), we examined several pre-existing reputation systems and methods for computing trust values, including work by Buchegger and Boudec [9], Cornelli et al. [11], Gupta et al. [12], and EigenTrust [14]. We use a modified Bayesian approach for trust feedback.

This gives a narrow scope, but because other P2P-related work can be leveraged to create a fully robust environment (*i.e.,* work related to authentication, data discovery, scalability, and fault-tolerance), this scope is not impractical. We assume some pair-wise trust, especially between the initial pair of peers (the initial sender/receiver pair). Remember, the goal of this system is to allow confidential data to be shared via one hop, but then limit the propagation of this data beyond the initial pairing. For example, peer $i$ may trust peer $j$, but peer $i$ has no understanding of any other peers that $j$ may trust. Thus, a confidential file passed between peer $i$ and peer $j$ should not go further than peer $j$.

## V. Content Distribution System

The design of the distribution system relies on data encapsulation. That is, data is stored withing an executable class file that renders the data once per authenticated peer. The owner and sender of the data sets the authenticated peer and machine. In addition to controlling access to this data, the data viewer also notifies the sender of usage of data. If peer $i$ sends a file to peer $j$, $i$ is notified once $j$ views the data, if the authentication failed, or if another peer attempted to use the data. Upon the first usage, the data is destroyed using method of self-destructing data. This combination of authentication and self-destruction is the key to establishing first level protection and limitations on distribution of data on the network.

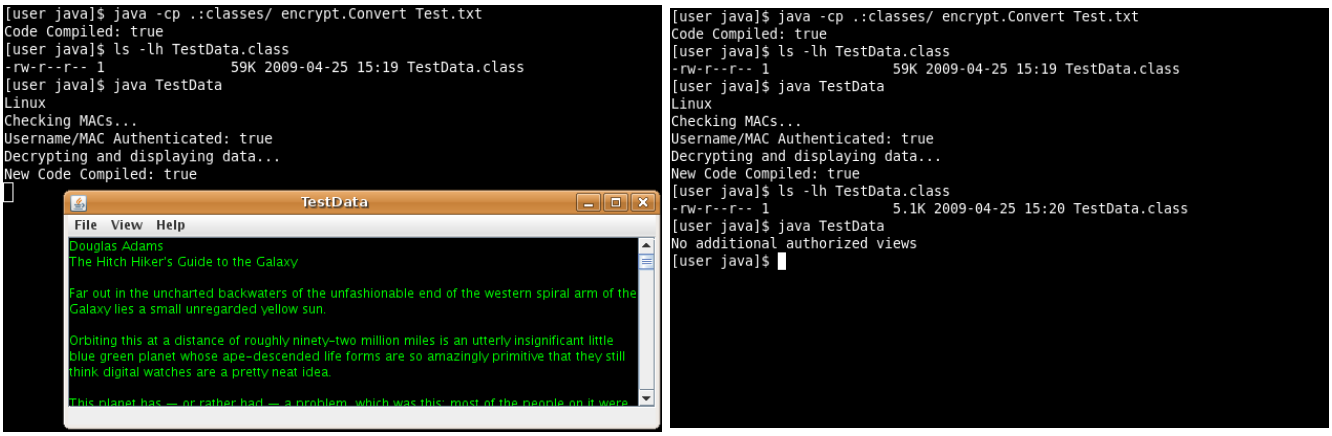


Fig. 1. Screenshots of the self-destructing data/data viewer

### A. Data Authentication

Any data received by a peer will be encapsulated within an executable Java class file. Only the owner(s) of data will have the data in its original format. Data is stored within the class file as an encrypted string (specifically, a base64 string) and the class file is obfuscated to make disassembly or reverse engineering difficult. In our proof of concept, we utilized ProGuard [2]. The data is only displayed after the peer's username and MAC address match those set by the data transaction. This information is gathered once a peer joins the network and is set automatically to prevent spoofing or easily impersonating another peer.

This two-factor authentication limits one peer on one machine to view the data. If authenticated, the data is then decrypted using a hash of these same two factors. If the authentication fails and the peer is on the network, the sender is notified of this failure. The class file is then overwritten to destroy the data using in-memory compilation. The new executable only contains the functionality to notify the sender of subsequent, unauthorized view attempt. This allows the sender to decrease the receiver's level of trustworthiness. This feedback is essential to updating a peer's trust value on the network. Once the data is decrypted, the data is rendered on-screen in a non-editable, non-selectable region to prevent it from being copied.

### B. Self-Destructing Data

The one-time-use policy is enforced through self-destructing data. Our method of self-destructing data is less intrusive than the virus-appended approach in [23]. We use in-memory compilation to overwrite class files. New source code, stored as an encrypted string within the executable, is decrypted then compiled in-memory. The new source contains none of the original data, but only the functionality to notify the sender of subsequent policy violations. Given write privileges on the class file, the Java Virtual Machine (JVM) will allow this file to be overwritten. This process is shown in Fig. 2.

In-memory compilation is vital for security of the application. If we store the code as a string, decrypt it, then attempt to compile it from disk, a malicious peer can potentially use the knowledge of the source code for exploitation. Even attempting deletion of the source file after compilation presents a security risk, as it is nonetheless stored on disk at some point. In encapsulating the data into the executable, the goal is to provide the receiver with the minimum knowledge needed to consume the data. Using in-memory compilation, no additional data must be stored outside of the executable. External information is far more susceptible to modification by a malicious peer than internal data. Data stored information in registry keys, for example, is easier to modify than variables stored within the class file.

The in-memory compilation is accomplished using the Java package `javax.tools.ToolProvider`'s `getSystemJavaCompiler()` and `javax.tools.SimpleJavaFileObject` [13]. An example of this code is shown in Fig. 3, and screenshots of the self-destructing data and in-memory compilation are shown in Fig. 1. Note after the first execution of the class file, the data is destroyed and the file size reduces from 59KB to 5KB.

### C. Dependencies and Limitations

The data viewer requires several dependencies for full functionality. Any unmet dependency will result in reduced security, so we require all be met for the viewer to execute. First, the client machine requires the current version of the Java Development Kit (JDK), version 6. This release supports the `javax.tools` package, which is needed for the in-memory compilation.

Secondly, write privileges are required on the received class file. In our test cases, the received class files are given write privileges for the receiving peer. Only after intentionally changing the file permissions was write access revoked. We assume a malicious peer would use an attack of this sort to prevent the executable from overwriting itself and delay data destruction.

Thus, given these unmet requirements, the viewer will notify the sender of this issue. In the second case, a peer has the potential to attempt reverse engineering of the class

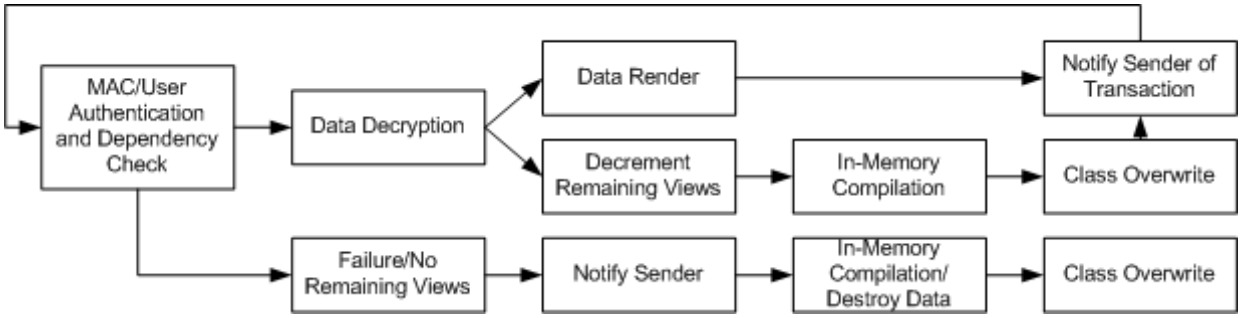


Fig. 2. Data viewer process

```
import javax.tools.*;
import javax.JavaCompiler.CompilationTask;
import javaxtools.JavaFileObject.*;
import java.io.StringWriter;
class Compiler {
  public void compile(String code) throws Exception {
    JavaCompiler javac = ToolProvider.
          getSystemJavaCompiler();
    JavaFileObject file = new
          JavaSourceFromString("NewClass",
          write.toString());
    Iterable<? extends JavaFileObject>
          compilationUnit = Array.asList(file);
    PrintWriter out = new
          PrintWriter(new StringWriter());
    out.println(code);
    out.close();
    CompilationTask task = jc.getTask(null, null, null,
          null, null, compilationUnits);
    task.call();
  }
  class JavaSourceFromString
    extends SimpleJavaFileObject {
    final String code;
    JavaSourceFromString(String name, String code) {
      super(URI.create("string:///"
           + name.replace('.','/')
           + Kind.SOURCE.extension, Kind.SOURCE);
      this.code = code;
    }
    @Override
      public CharSequence getCharContent
           (boolean ignoreEncodingErrors){
          return code;
    }
  }
}
```

Fig. 3. In-memory compilation

file to obtain the data since it cannot be overwritten. However, since each execution notifies the sender, the receiver's trust can be reduced under the assumption some malicious action is being attempted. We must also consider complications during the overwrite. Perhaps the encrypted source code may become corrupted or fails to compile. In these cases, the class file is deleted but the receiver's trust is not affected. Two methods of file deletion are attempted, first a simple `java.util.File.delete()`. This may fail, so `java.util.File.deleteOnExit()` is attempted, which uses Java level shutdown hooks to mark the file for deletion.

Another potential requirement is network access. We chose not to enforce network access to allow offline usage of the data. Requiring access would severely limit the availability of the data and would reduce the viewer to nothing more than an online browser application. Our goal is to limit distribution of data without any significantly limiting requirements. Remember, the intent of our protocol is to protect third party access to confidential data. The original transfer is based on a trusted pair peer session. We protect the integrity of the data from being transferred beyond this pair-wise trust relationship. One mediation we propose is a time-to-live policy that can be set by the sender and enforced by the data viewer. We assume each TTL can be customized for each transaction, or set to an infinite amount of time. If the peer does not return to the network and consume the data prior to the TTL, his trustworthiness is decreased. For each TTL interval in which a peer does not respond, his trustworthiness is further decreased. An infinite TTL can be used in the case that a peer plans to consume the data offline, which can be arranged with the sender during the transaction. We assume that a peer will only request data when he requires it, or before he leaves the network, so a TTL is not a severely limiting requirement.

For the in-memory compilation, few restrictions exist. First, class files have a size limitation. The value of the `code_length` attribute in the `code_attribute` structure must be less than 65536 [15], as per the JVM class file specification. Data files larger than this can be split into multiple strings, stored in separate class files, and then coalesced at run-time. Additionally, given this 64KB limit, there are no significant restrictions due to memory limitations when compiling the source files in memory.

### *D. Security Considerations*

Keeping in mind that our system is designed to protect accidental or malicious electronic propagation, the goal of our design is to provide some level of ownership over data once an owner has distributed it onto the network. As such, the decrypted data will be in memory for the duration the viewer is running. Since data is neither selectable nor editable, a peer cannot copy/paste the data, but could use a core dump to retrieve the current on-screen data. A screenshot or screen-capturing software could also be used to retrieve the data. For large amounts of data, this is a tedious task.

Reverse engineering of the class file is another consideration. We used ProGuard to obfuscate the class files containing

the data, making reverse engineering a more difficult task. More advanced techniques include [10], [21]. The one-time-use policy also makes reverse engineering difficult. Since the sensitive data is overwritten during use, the malicious peer has one attempt at this task.

The use of virtual machines (VM) complicates the use of self-destructing files. For example, multiple snapshots of a VM would seem to allow multiple views of the data. This may seem to circumvent a safeguard of the system (the one time use restriction); it does not circumvent the goal of our protocol which is to restrict the propagation of confidential files in a network. The trusted peer receiver may have created a way to view the file several times, but if the file is transmitted to another peer (accidentally or not), the file is unusable (self destructed) and the confidentiality is in tact.

## VI. Reputation Management System

Trust information is propagated on the network using a reputation management system. All feedback is performed automatically–peers do not have the ability to vote on the type of transaction. To compute trust values, we focused two different algorithms: EigenTrust [14] and Buchegger and Boudec's Bayesian approach [9], with some modifications to the second to meet the needs for our network. Though each has its strengths and trade-offs in our environment, we expect the Bayesian approach, with weight given towards first-hand information, to yield the best results.

### A. Bayesian Approach

*1) Background:* Buchegger and Boudec's approach stores two ratings about peers: a *reputation rating* that represents the opinion of a peer's behavior "as an actor in the base system" and a *trust rating* that represents a peer's behavior "as an actor in the reputation system." Peer *i* stores reputation ratings about peer *j* as $R_{i,j}$ and trust ratings as $T_{i,j}$. *First-hand information* is stored as $F_{i,j}$. The distribution Beta($\alpha$, $\beta$) is used for the prior in the Bayesian framework, and first-hand information is computed using *s* as observed misbehaviors and *u* as the discount factor:

$$\alpha := u\alpha + s \tag{1}$$

$$\beta := u\beta + (1 - s) \tag{2}$$

Buchegger and Boudec propose that $u = 1 - \frac{1}{m}$ where *m* is "the order of magnitude of observations...to assume stationary behavior." The reputation rating $R_{i,j}$ is defined by ($\alpha'$, $\beta'$), with $\alpha'$ and $\beta'$ calculated in a similar manner as above. For reputation ratings published by other peers, a peer *i* updates first-hand information $F_{i.j}$ from a trustworthy peer *k* with small positive constants *w* [7], [8]:

$$R_{i,j} := R_{i,j} + wF_{k,j} \tag{3}$$

Peer *k* is determined to be trustworthy according to Eq. 8. If peer *i* determines *k* is untrustworthy, the second-hand information is not used. If untrustworthy, a deviation test is computed given $F_{k,j}$ = $(\alpha_F, \beta_F)$ and $R_{i,j} = (\alpha, \beta)$:

$$|\mathbb{E}(Beta(\alpha_F, \beta_F)) - \mathbb{E}(Beta(\alpha, \beta))| \geq d \tag{4}$$

with *d* serving as the deviation threshold. The trust rating $T_{i,k}$ is calculated using:

$$\gamma := v\gamma + s \tag{5}$$

$$\delta := v\delta + (1 - s) \tag{6}$$

with discount factor *v* and *s* = 1 if the deviation test succeeds. Finally, behavior can be classified using:

$$\begin{cases} \text{normal} & \text{if } \mathbb{E}(Beta(\alpha', \beta')) < r \\ \text{normal} & \text{if } \mathbb{E}(Beta(\alpha', \beta')) \geq r \end{cases} \tag{7}$$

and trustworthiness using:

$$\begin{cases} \text{trustworthy} & \text{if } \mathbb{E}(Beta(\gamma', \delta')) < t \\ \text{untrustworthy} & \text{if } \mathbb{E}(Beta(\gamma', \delta')) \geq t \end{cases} \tag{8}$$

with *r* and *t* serving as expressions of tolerance.

*2) Application:* We propose several modification to Buchegger and Boudec's work to tailor it to our environment. First, we eliminate decay of values during periods of inactivity (using $\alpha := u\alpha$ and $\beta := u\beta$). However, we retain the use of the discount factor as a fading mechanism to give weight to more recent transactions. We chose to eliminate decay during inactivity for the same reason we do not leverage any trading schemes to increase trading, but with a small rate of decay. We reason that data is only transferred on an as-needed basis, which may include large periods of inactivity for some users, so a decay during periods of inactivity is not practical. Untrustworthy peers should only become more trustworthy with proper use of data, not time.

We also keep the reputation and trust ratings as continuous variables, giving each user some real-valued rating, as opposed to the discrete ratings with thresholds *r* and *t* in the original algorithm.

### B. Analysis

Though both EigenTrust and Buchegger and Boudec's work are aimed at P2P networks, they have noticeably different approaches. EigenTrust converges to the left principal eigenvector so peer *i* will have the same view of any other peer on the network as peer *j*. The Bayesian approach, with more emphasis on first-hand information, provides different opinions of peers for each based on their interaction with other peers. Given the emphasis on pair-wise interactions in our network, we believe this be the best algorithm for our design.

To prove this, we ran a simulation of 1000 data transfers between three peers. Comparing a larger number of users may be preferable to determine the effects of malicious collectives or other attacks, but to analyze the effectiveness of the algorithms for our design, we chose to focus on a smaller user base. Additionally, since our design emphasizes pair-wise interactions, three users is appropriate as we can note the differences in views of different peers based on first- or second-hand information.

The results confirm several of the weaknesses in the algorithms. The most important being that in EigenTrust, unsatisfactory transactions are not counted when normalizing trust values and could skew reputation values (malicious peers can

seem less so). However, in the Bayesian approach, this is not the case. Thus, the modified Bayesian approach is appropriate for our system.

## VII. Future Work

Future work for our design includes further refinement of the trust algorithms, comparisons to other trust algorithms (such as those in [4], [12], [19], [18]), and increased flexibility for the data viewer.

While our current implementation may succeed in preventing data distribution on a P2P network, it does so with a read-only limitation. That is, if peer *j* sends the file to peer *i*, *j* cannot modify the file and return it to *i*. This is one area we would like to address in future work. Allowing outright write-access is difficult due to a method of storing these changes locally. One idea we propose is annotations and requests for deletion. If peer *i* sends a file to *j*, *j* would have the ability to add annotations to the data and mark areas for deletion. This must be done with care, as the data viewer must still be non-selectable and non-editable to prevent copy-and-pasting of the data outside the viewer.

## VIII. Conclusion

Controlling ownership and access to data in a network environment is difficult, though nevertheless important for confidential data. Through self-destructing data and trust values based on feedback from data usage, we have developed a method of first-level protection against unauthorized distribution in P2P networks. We have addressed the security issues related to this type of self-destructing data and shown that a Bayesian approach works best for the needs of our environment.

## References


[1] Endpoint Data Loss Prevention. http://www.nextlabs.com/html/?q=endpoint-data-loss-prevention.

[2] Proguard. http://proguard.sourceforge.net/.

[3] RSA Data Loss Prevention (DLP) Endpoint. http://www.rsa.com/node.aspx?id=3429.

[4] Karl Aberer and Zoran Despotovic. Managing trust in a peer-2-peer information system. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, pages 310–317, New York, NY, USA, 2001. ACM.

[5] W.J. Adams and IV Davis, N.J. Toward a decentralized trust-based access control system for dynamic collaboration. pages 317–324, June 2005.

[6] Alapan Arnab and Andrew Hutchison. Persistent access control: a formal model for DRM. pages 41–53, 2007.

[7] James O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer, Second Edition, 1985.

[8] Sonja Buchegger. *Coping With Misbehavior in Mobile Ad-hoc Networks*. PhD thesis, 2004.

[9] Sonja Buchegger and Jean Y. Le Boudec. A robust reputation system for p2p and mobile ad-hoc networks. In *Proceedings of the Second Workshop on the Economics of Peer-to-Peer Systems*, 2004.

[10] Jien-Tsai Chan and Wuu Yang. Advanced obfuscation techniques for java bytecode. *J. Syst. Softw.*, 71(1-2):1–10, 2004.

[11] F. Cornelli, E. Damiani, De Capitani, S. Paraboschi, and P. Samarati. Choosing reputable servents in a p2p network. In *Eleventh International World Wide Web Conference*, Honolulu, Hawaii, May 2002.

[12] Minaxi Gupta, Paul Judge, and Mostafa Ammar. A reputation system for peer-to-peer networks. In *Proc. of the 13th Int. workshop on Network and Operating Systems support for digital audio and video*, pages 144–152, 2003.

[13] JavaCompiler (Java Platform SE 6). http://java.sun.com/javase/6/docs/api/javax/tools/JavaCompiler.html.

[14] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-molina. The eigentrust algorithm for reputation management in p2p networks. In *In Proceedings of the Twelfth International World Wide Web Conference*, pages 640–651. ACM Press, 2003.

[15] Tim Lindholm and Frank Yellin. *Java(TM) Virtual Machine Specification, Second Edition*. Prentice Hall, 1999.

[16] Xiaoning Ma, Zhiyong Feng, Chao Xu, and Jiafang Wang. A trust-based access control with feedback. pages 510–514, May 2008.

[17] Paul B. Schneck. "Persistent Access Control to Prevent Piracy of Digital Information". *Proceedings of the IEEE*, 06/1999.

[18] Asad Amir Pirzada and Chris McDonald. Establishing trust in pure ad-hoc networks. In *ACSC '04: Proceedings of the 27th Australasian conference on Computer science*, pages 47–54, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[19] Uwe Roth and Volker Fusenig. Position paper: How certain is recommended trust-information. 2006.

[20] Mema Roussopoulos, Mary Baker, David S. H. Rosenthal, Tj Giuli, and Jeff Mogul. 2 p2p or not 2 p2p. In *In IPTPS 04*, pages 33–43. Springer, 2004.

[21] Yusuke Sakabe, Masakuzu Soshi, and Atsuko Miyaji. Java obfuscation approaches to construct tamper-resistent object-oriented programs. *IPSJ Digital Courier*, 1:349–361, 2005.

[22] Huu Tran, M. Hitchens, V. Varadharajan, and P. Watters. A trust based access control framework for p2p file-sharing systems. pages 302c–302c, Jan. 2005.

[23] Howard R. Udell, Cary S. Kappel, William Ries, Stuart D. Baker, and Greg M. Sherman. Self-destructing document and e-mail messaging system". US Patent Number 7191219, April 12, 2002.