Enhancing Data Authenticity and Integrity in P2P Systems

Peer-to-peer systems let users share information in distributed environments because of their scalability and efficiency. However, existing P2P systems are vulnerable to numerous security attacks and lack a mechanism to ensure shared information's authenticity and integrity. A proposed general architecture enhances these aspects by leveraging trusted computing technology, which is built on a trusted platform module and provides a mechanism for building trust in the application layer. Preliminary experimental results show that the proposed scheme can ensure data authenticity and integrity in P2P systems with acceptable performance overhead.

Xinwen Zhang, Songqing Chen, and Ravi Sandhu
George Mason University

Peer-to-peer systems have gained considerable attention because of their global scalability and high efficiency. Although P2P systems are useful for content distribution (Napster, KaZaa, and BitTorrent), computing capability sharing (SETI@home), and collaborative network systems (Friend Troubleshooting Network[1]), various possible attacks threaten these systems.[2,3] At the network level, for example, structured P2P overlay networks are prone to malicious routing.[3] Gnutella and other systems have suffered from denial-of-service (DoS) attacks due to inherent weakness in the protocols. Attacks can be easily mounted at the application level and thus are hard to prevent. For example, in content-sharing systems, a peer can maliciously return false data, or two peers can collude to break the systems' anonymity.

With this article, we focus on the specific problems of data authenticity and integrity instead of discussing P2P security in general. We propose a general architecture that enhances the authenticity and integrity of data shared in these systems by using trusted computing (TC) technologies. (See the "Related Work in Trusted Computing" sidebar for the other work in this area.) Specifically, we propose a trusted reference monitor (TRM) in the platform of each peer beyond necessary trusted hardware and supporting functions. A TRM can monitor and verify the information a peer provides to ensure data authenticity. Using the credentials protected by the underlying

 Published by the IEEE Computer Society 1089-7801/05/$20.00 © 2005 IEEE

Related Work in Trusted Computing

With this article, we propose an architecture in the application layer with abstract underlying trusted computing (TC) technology. Previous work has proposed developing various underlying architectures and improving TC's primitive functions. For example, Ahmad-Reza Sadeghi and Christian Stuble propose[1] a trusted platform architecture in which a micro kernel is applied on top of trusted hardware such as a trusted platform module. Other work[2] proposes a secure-kernel-based approach to enforce fine-grained attestations between applications. Also, researchers have presented a virtual-machine-based attestation to capture a remote entity's behavior, while the language-based virtual machine itself is attested by signed-hash mechanism.[3] All these approaches can be applied as concrete underlying TC mechanisms in our approach.

A TC application for protecting entertainment products against pirates in peer-to-peer (P2P) content distribution networks is also available.[4] The main idea is to use remote attestation to control a peer's joining the network and publishing content. Only certified platforms and applications that the peers in the network trust can use a P2P system's resources. Although the trust is built on the P2P application in our work, we use TRM as a common and compact component, which makes our approach more practical.

References

1. A. Sadeghi and C. Stuble, "Taming Trusted Platforms by Operating System Design," *Proc. 4th Int'l Workshop on Information Security Applications*, LNCS 2908, Springer-Verlag, 2003, pp. 286–302.
2. E. Shi, A. Perrig, and L. Van Doorn, "Bind: A Fine-Grained Attestation Service for Secure Distributed Systems," *Proc. IEEE Symp. Security and Privacy*, IEEE CS Press, 2005, pp. 154–168.
3. V. Haldar, D. Chandra, and M. Franz, "Semantic Remote Attestation: A Virtual Machine Directed Approach to Trusted Computing," *Proc. 3rd Virtual Machine Research and Technology Symp.*, Usenix Assoc., 2004, pp. 29–41.
4. S. Schechter, R. Greenstadt, and M. Smith, "Trusted Computing, Peer-to-Peer Distribution, and the Economics of Pirated Entertainment," *Proc. 2nd Int'l Workshop Economics and Information Security*, 2003; www.eecs.harvard.edu/%7Estuart/papers/eis03.pdf

hardware, a TRM can digitally sign data from a peer and provide verification mechanisms for a remote site. Preliminary evaluation results show that our proposed scheme introduces acceptable overhead to achieve the security goals.

Case Examples

The distinguishing concept in P2P systems is *sharing*, by contributing to and benefiting from the peer community.[4] A fundamental problem in P2P systems is how to ensure and verify the authenticity and integrity of shared data or communicated information in open and highly distributed environments. Most existing P2P applications don't have mechanisms to address these problems, as these examples demonstrate:

- *Case 1: Configuration attack in the Friend Troubleshooting Network.* Microsoft developed the Friend Troubleshooting Network (FTN[1]) as a P2P system to diagnose misconfiguration on a sick machine by querying correct configuration information from a network consisting of friends' machines. A malicious peer can contribute either fake configuration information to make the diagnosis fail or generate some configuration information to compromise the sick machine.
- *Case 2: DoS attack in Gnutella.* In Gnutella (www.gnutella.com), a peer sends a query message to all its neighbors, which in turn send it to their neighbors. When a peer with the target object is reached, it responds to the message and builds a connection to the original peer. If a malicious peer exists in the middle of the network, it can capture many queries and simply respond to each one that the target object is available in a victim peer. Then, all the queries' source peers will try to build connections to the victim peer and flood it. The fundamental problem with this DoS attack is that a malicious peer can reply to a query with the peer ID of another peer (the victim) — that is, a peer can return some inauthentic information.
- *Case 3: Pollution attack in BitTorrent-like P2P systems.* BitTorrent (BT) is an efficient Internet-based content distribution system.[5] In BT, an object is split into many pieces (for example, each piece might be 256 Kbytes). To distribute an object, its owner first makes a meta-info file (.torrent file), which includes the object's information (file name, length, and so on) and a tracker site's URL — meta-info is the original name used in BT for metadata. Each object's meta-info also includes a string of Secure Hash Algorithm, version 1.0, (SHA1) hash values of the pieces. The tracker site maintains a list of active peers in the network. When a peer tries to download an object, it first gets the meta-info file (available from a public server) and queries the tracker site. The tracker site responds with a list of peers where the pieces are available. The downloader then connects to

selected active peers from this list and downloads pieces. Each peer reports to the tracker site periodically with its downloaded pieces so the tracker can update its tracking list and other peers can download pieces from this peer.

If a malicious peer in the network modifies the contents before sharing with other peers, many peers can download a piece from this peer at the same time and the faked pieces will be distributed widely and quickly in the network. A peer can verify a piece's integrity with the hash value in the meta-info file after completely downloading the piece, but it can't figure out whether a peer maliciously modified a piece or whether it resulted from a transmission error. Also, a malicious peer can upload incorrect content when requested. A tracker site's current implementation doesn't enforce any mechanism to prevent further distribution of compromised pieces. This attack is more severe in a file distribution's starting period because BT uses a rarest-first strategy for piece selection to speed up distribution. With this strategy, different peers download different pieces of a file before the original peer disappears from the network. If a malicious peer modifies the pieces that are only available on it, the object distribution will fail. Similar pollution attacks also happen in other file-sharing systems like KaZaa.[6] In KaZaa, a peer searches the network to look for an object by file name or keywords. A peer can publish fake or junk files with the names or keywords of some popular files, causing normal users to frequently download the wrong files. This quickly makes peers lose trust and interest in the community.

Given the open and distributed computing architecture of P2P applications, TC has great potential to enhance security in P2P systems.

The underlying problem in these examples is that the systems can't authenticate shared information. The goal of authenticity is to ensure that shared data or information from a peer is genuine and that an attacker hasn't maliciously modified it. Generally, the authenticity of the information from a peer depends on valid properties and behavior of the peer or the P2P software running on the peer's machine. In P2P systems, most information sharing and communications happen between peers at the edge of the network, using general platforms such as PCs, PDAs, and smart phones. Besides that, a misconfigured system on a platform can cause a P2P application's unexpected behaviors, deliberate software-based attacks on a client platform present an increasing risk. Malicious software can illegally read or modify sensitive data in persistent storage, memory, and I/O buffers in these platforms as well as change the request for information and actions sent to other platforms. The lack of strong security mechanisms on client platforms in general, and the push for an open environment on computing devices, such as PDAs, smart phones, and notebooks, leave the client extremely vulnerable to software attacks. Due to the dynamics of a peer with frequent arrivals and departures, these attacks are hard to prevent in existing P2P systems.

In P2P systems, in addition to ensuring that data is from a genuine peer and the content is authentic, it's also necessary to make sure that the data hasn't been modified or compromised during transmissions. Generally, integrity verification depends on a peer's authentication. Unfortunately, in most existing P2P systems, no infrastructure exists for identifying peers and providing all peers with digital certificates.

TC and the Trusted Platform Module

The problem of attesting and verifying valid properties and runtime behavior for an entity (platform or application) is related to the concept of trust, which is motivated by emerging hardware-based trusted computing (TC) technologies. Software alone can't provide an adequate foundation for building a high-assurance trusted platform. The quest for finding the correct set of hardware primitives for TC was shaped in the mainframe era of the 1970s by the pioneering Multics system, followed by a number of academic and commercial capability-based computers. The most recent attempts to specify hardware trust primitives began in the late 1990s and continue to be pursued today. TC technologies seek to protect data's creation, processing, storage, and transfer primarily by only exposing the cryptographic secrets required to access the data to software with a ver-

ifiable chain of trust, be it on a single computer or across multiple computers. Modern TC primitives are designed for a distributed and dynamic open environment in which we can execute and protect trusted application software from interference from other software on the same platform. Thus, the trust mechanisms provide for greater security for software execution within a single platform. In addition, direct support exists for platform-to-platform propagation of trust. Reliance on appropriate application software to actually enforce the security policy is an integral part of this approach. Given the open and distributed computing architecture of P2P applications, TC has great potential to enhance security in P2P systems.

The Trusted Computing Group (TCG) defines a set of specifications aiming to provide hardware as the root of trust and a set of primitive functions to achieve trust in high-level software and applications. The root of trust in TCG is a hardware component on a platform's motherboard called the trusted platform module (www.trustedcomputing group.org). TPM provides protected data (secrets and arbitrary data) by never releasing a root storage key outside the TPM. In addition, TPM provides some primitive cryptographic functions, such as random number generation, RSA key generation, and asymmetric key algorithms. Most importantly, a TPM provides a mechanism for measuring, storing, and reporting a platform's integrity, from which we can achieve strong protection capabilities and attestations.

A TPM contains a set of platform configuration registers (PCRs), which store the integrity measurements of a platform's running code, from when the system boots to loading the operating system to launching applications. A measurement of protected data or program code represents the measured object's properties and characteristics, such as integrity, a process' running states, and configurations. The specific PCR's value is updated by applying a SHA1 operation on its current value concatenated with a newly measured value.

A TPM includes a set of credentials, of which the TPM protects the private keys and never releases them, and the public key is certified by corresponding certificate authorities (CAs). The TPM manufacturer issues the endorsement certificate, which uniquely identifies the individual TPM batch and its source. Another credential is a TPM's *attestation identity*, which a privacy CA certifies. The privacy CA uses the endorsement key and other information to verify that the platform has a genuine TPM without releasing the TPM's individual identity. A TPM can have multiple attestation identity credentials certified by different CAs.

With the integrity measurement and storage capability, a platform can generate an integrity report and give it to another platform through a challenge-response protocol called *attestation*. During attestation, a platform (challenger) sends an attestation-challenge message to another platform (an *attestor*). The attestor signs one or more PCR values with its attestation identity key (AIK) and gives them to the challenger, along with a list of measurement events in the attestor. The challenger verifies this attestation by verifying the signature and comparing the integrity hash with expected values. Attestation provides the authenticity of a platform's current integrity, state, or configurations.

TPM security enhancement protects sensitive data (such as an application's secrets or a user) with integrity measurement values through *sealed storage*. Not only is a key applied to encrypt the data, but one or more PCR values are stored during the encryption. A TPM releases such a protected object only if the current PCR values are matched with those stored with the protected object. Therefore, a protected object can only be available when the platform is in a particular known state. The TCG specifications don't require the TPM to be tamper resistant. As such TPM-based TC technologies only address software-based attacks.

In addition to a TCG-compliant TPM, Intel's LaGrande Technology (LT; www.intel.com/technology/security) includes extended CPU that enables software domain separation and protection as well as extended chipsets that enable protected graphics and basic I/O devices (such as a keyboard and mouse), which constructs trusted channels between applications and these devices. Beyond the hardware layer, a domain manager supports protected running environments, including separation of processes and memory pages.

Proposed Architecture

Our proposed general trusted platform with a TRM assumes a homogeneous environment — that is, each platform is equipped uniformly with the necessary TC hardware.

Figure 1 shows an abstract picture of our proposed TC platform. The basic trusted component is the trusted hardware including a TPM. A TRM is an application or service component running in the operating system's user space, enforcing access control policies in general client-side platforms.[7] The

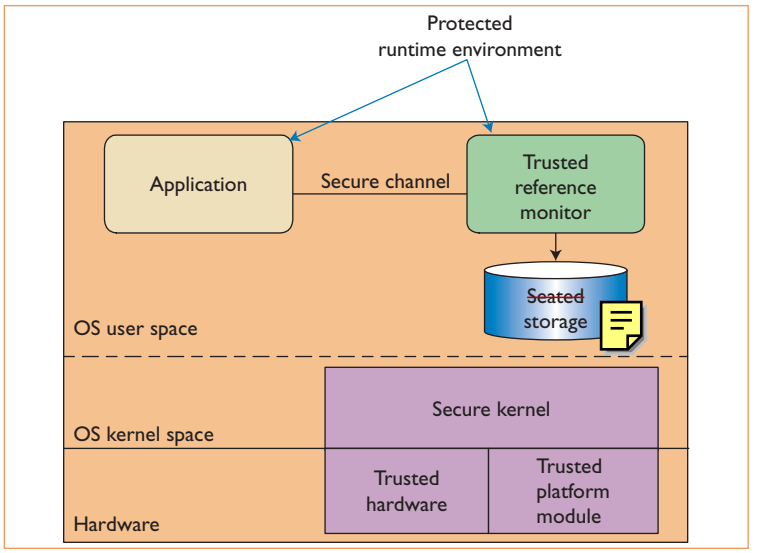

Figure 1. A platform architecture with trusted hardware and a secure kernel. We can verify the trust of TRM by measuring its integrity before loading and running it in a protected environment.

hardware, cooperating with the security kernel, provides necessary functions to the TRM, from basic cryptographic functions to platform and program attestation, and sealed storage for sensitive data.

In our abstract design, the secure kernel can be separated from the operating system's main kernel, similar to Nexus in Microsoft's Next-Generation Secure Computing Base (NGSCB; www.microsoft.com/resources/ngscb), or it can be a special component of the main OS. The secure kernel separates execution between upper layer applications. When an application or process (including a TRM and P2P application) loads, the secure kernel allocates an isolated memory space for it such that no interference exists between any two separated processes, preserving their runtime integrity. For TRM, an isolated runtime environment also ensures that sensitive information such as control policies isn't vulnerable to malicious processes in the platform. A requirement of our trusted platform is the secure communication between processes, either implemented via the secure kernel or directly by individual applications.

We abstract the underlying hardware and kernel structure by simply assuming that we can achieve these requirements and necessary TC functions by using existing hardware technology (such as LT) and operating systems (such as NGSCB) in a platform. For example, in LT, we apply a TPM v1.2 in the hardware layer with an extended CPU and chipset. A domain manager layer between the kernel and hardware separates application domains. The platform's integrity measurements, including the secure kernel, are stored in specific PCRs in the TPM.

We assume the following credentials and corresponding certificate authorities are available in our architecture:

- TPM storage key(s) to protect credentials and sensitive data with sealed storage, such as secrets and policies. This key must be either from a TPM's storage root key (SRK) or a key protected by the SRK.
- TPM attestation identity key pair (PK_{AIK}, SK_{AIK}). A TPM owner creates an AIK and uses it to sign PCR values, present it to a challenger in an attestation protocol, and sign the secure kernel's public key. Generally, the private part of an AIK is protected by a TPM with a storage key and the public key is certified by a trusted third party such as a privacy CA to verify that the platform has a genuine TPM without releasing the platform's information. A TPM can have numerous AIKs with certificates from different CAs.
- Secure kernel asymmetric key pair (PK_{SK}, SK_{SK}) for signature and encryption. The AIK of the TPM certifies the public key, and the TPM protects the private key with a storage key.
- TRM asymmetric key pair (PK_{TRM}, SK_{TRM}). Each TRM has this key pair for signature and encryption. The private key is protected by the TPM in the platform such that only the TRM on this platform can use it (by checking the TRM's integrity value when using the private key). The public key is in a certificate format signed by an AIK of the TPM.

After booting the platform, the TPM measures the integrity of the booting system and the secure kernel in a platform and stores in particular PCRs. Before starting an application, the secure kernel measures the application code's integrity and stores the hash value locally. The secure kernel also generates a public-private key pair for the TRM, where the public key is certified by the secure kernel (by signing with its private key) and the private key is protected by the TPM. This key pair is generated the first time when the TRM is loaded into the platform. For attestation, the TPM signs the values in the PCRs with its AIK key, and the secure kernel signs the TRM's integrity value with its private key. Both are sent to a remote challenger. The challenger verifies both signatures and the public key certificates of the TPM AIK and the secure kernel. If all are valid, with the expected

integrity values, the secure kernel and TRM are trusted. Also, the TRM can seal and unseal sensitive data (such as secrets and policy information) with a set of PCR values the TPM provides.

Working Protocol

We can use the Friend Troubleshooting Network as an example to explain our architecture and protocol. The policy a TRM enforces in FTN requires that a peer's contributed configuration information is the real configuration entry in its local platform, not fake or compromised information. As Figure 2 shows, the requestor is a sick platform that queries some configuration information from the provider platform. By assumption, each peer is equipped with necessary trusted hardware. Before starting a query, the requestor first sends a remote attestation challenge to the provider's TRM to make sure that the TRM is in a valid state to enforce the policy (A1). The TRM calls primitive functions provided by a secure kernel and TPM software service to collect PCR values of the platform and the TRM (A1′). The TPM signs the PCR values and the secure kernel signs the integrity value of the TRM and both are signed by the TPM's AIK and returned to the TRM (A2′). The TRM responds to the attestation challenge with this signed value and the TPM's AIK credential and the secure kernel's public key certificate (A2). The requestor verifies the signature and compares it with trusted values. If the platform and TRM are trusted, the requestor sends the policy information such as a set of configuration entries the requestor queries (A3).

After establishing trust for the provider's TRM and platform, the requestor sends its query to the provider through the FTN P2P software just like the normal P2P protocol (B1). The FTN software on the provider collects the corresponding data and sends it back to the requestor. The TRM at the provider side captures the content sent out by the FTN, collects corresponding configuration information, and compares it with the content sent out by FTN software to check the authenticity and integrity, according to the original requested information received in A3 (B2). If the data is valid, the TRM authorizes the response from the provider to the requestor (B3). The TRM can then sign the data (with the private key of the provider's TRM) to ensure integrity before sending it out. The FTN software completes the transaction by sending out the response (B4).

Mutual trust is necessary in some scenarios — for example, to prevent a requestor from maliciously collecting information on a target platform and analyzing its configuration, which is generally considered a platform owner's private information. For such cases, a provider also sends an attestation challenge to the requestor's FTN software. The provider can contribute its configuration information only if the requestor's platform, TRM, and P2P software are trusted and some properties are satisfied — for example, the FTN software isn't infected by a virus or compromised by malicious processes in the requestor's platform.

In some applications, the TRM needs to protect the policy information. For example, in BT, a malicious client software in a peer platform can modify the pieces and corresponding hash values in the meta-info file to make the verification fail. A TRM can seal this information with a set of PCR values such that only the platform and TRM with a correct running state can unseal and access this information. This state, in turn, ensures that this information isn't vulnerable to malicious software.

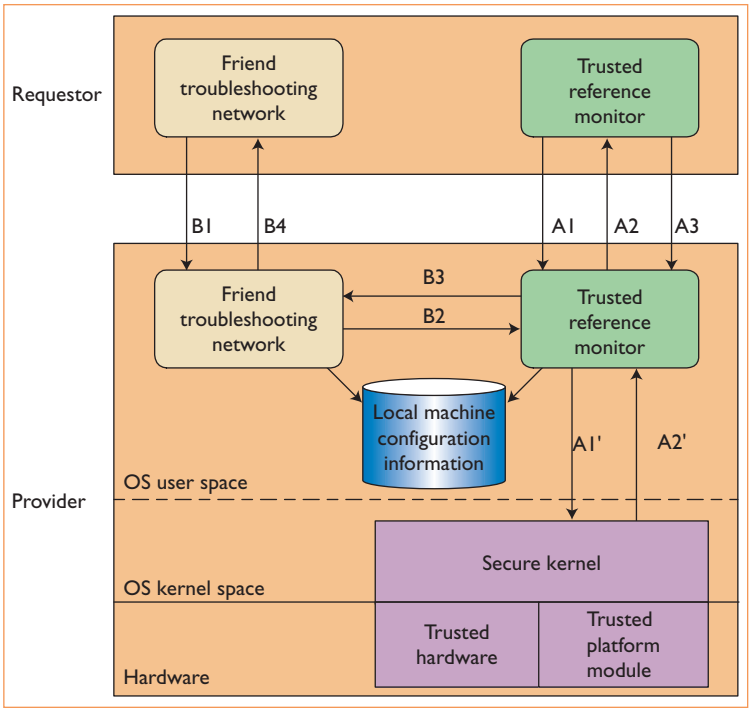

Figure 2. Working protocol to monitor shared data. Before sharing, the requestor challenges the provider through an attestation protocol between the TRMs. The provider's TRM ensures the authenticity of the shared data.

Authenticity Verification Mechanism

Corresponding to various P2P application systems, there are different authenticity verification processes in the TRM. Generally, authenticity verification is performed by the TRM via comparing

the target object with known valid content or the target object's integrity with a known integrity hash or digital signature. We can show this with different applications:

- In FTN, a requestor first informs the provider's TRM when configuration entries are queried (A3). The TRM collects this information directly from the local platform's configurations. For example, for the Windows operating systems, the entries in Windows registry are the valid objects that a TRM must verify. We assume that the local configuration information is authentic.
- We can apply a similar verification mechanism in Gnutella. A peer's TRM in Gnutella can check the response message replied by this peer to ensure that the target resource's peer ID (such as the IP address) is the same as this local peer. This policy can be loaded to the TRM whenever it joins the network.
- In BT (and BT-like P2P systems), a peer's TRM can obtain each piece's SHA1 hash value from the object's meta-info file, which the object's owner makes publicly available (possibly on a Web server). When a BT client software at the provider side shares a piece, it uploads to the requestor side. The TRM intercepts the traffic and generates the outgoing piece's hash and compares it with that in the meta-info file. If the integrity is maintained, the TRM can allow the upload . Generally, a piece is split into several chunks, and the provider uploads to the requestor chunk by chunk. Because a hash value in the meta-info file corresponds to an entire piece, the TRM verification at the provider side is an accumulative process that collects all chunks of a piece before generating its hash value. Because the piece length in a particular implementation of BT protocol is generally fixed (for example, 256 Kbytes), a TRM can do this by checking outgoing IP packets' header information.

Authenticity verification in KaZaa is different than BT's. In KaZaa, any peer can publish files in the network, and a peer searches an object with file names or key words. Therefore, a shared object's authenticity depends on many aspects other than a single hash value. Researchers have proposed some mechanisms[2] addressing this problem such as expert-, voting-, and reputation-based approaches. How trusted computing can leverage these mechanisms is an open problem.

TRM's Design and Trust

Instead of verifying the trust behavior of a peer based on the P2P application software, our approach leverages the TRM to enforce security policies. The main motivation for this is that we can build the TRM with minimum functions and a compact size and can conduct software verification on it. This ensures that a TRM's behavior can be verifiable.

On the other hand, P2P applications are generally complicated and developed with various techniques, where the code verification is difficult to conduct. Furthermore, as a common component in client platforms, a TRM can enforce security policies for various P2P applications. For trust verification, a TRM's expected integrity value can be certified by a trusted third party. In a collaborative environment such as a P2P peer group, a group's owner or administrator can specify what state of a TRM is acceptable and can enforce this through attestation before a peer joins the group, for example, through admission control.

Performance Evaluation

Our design introduces performance overhead from two aspects: TRM's integrity attestation between two platforms and the data authenticity verification between the TRM and P2P software on a local platform. The attestation requires communication between the TRM and the underlying operating system. Unfortunately, no such equipment is available today. Therefore, we can't quantify its performance overhead. We conjecture that the hardware-related operation doesn't contribute significant overhead because we use the signed-hash mechanism inherent in TPM, which is a typical function that related software services provide.

To study the security verification overhead between the TRM and the P2P software on a local platform, we implemented a simple prototype of FTN using Microsoft Windows 2000 and the Java 2 Standard Edition (J2SE) version 1.4. In the simulation, the provider process (provider P2P application) reads keys and corresponding values in the local Windows registry. Before sending this to a requestor process (requestor P2P application), the provider process queries the TRM process with the parameters of key names and values. The TRM process reads the keys and values from the Windows registry and compares them with the received parameters. We implemented the interprocess communication between the TRM and the provider process with Java Socket and carried out the experiment on a desktop computer with an Intel Pentium

II 600 MHz CPU and 256 Mbytes of memory.

Figure 3 shows the time for a FTN provider to respond to various queries from a requestor. The figure shows that, without security verification, it takes approximately three seconds to process 2,000 queries; with TRM verification involved, it takes about 6.4 seconds (Case 1 in Figure 3), more than double. This is reasonable because in addition to communication and verification, the TRM process does the same operation as the provider process (collecting key values from the local Windows register) and then compares them with the message received from the provider. The experimental results find similar trends for 500, 2,000, and 4,000 queries. In real applications, the TRM might have the authentic data — for example, by caching a key value that it collects beforehand — or the data's hash value to be verified (from a meta-info file in BitTorrent). This saves the extra overhead of TRM. In this case (Case 2 in Figure 3), the result shows that the overhead is on average 25 percent more than without verification. This demonstrates that our approach doesn't significantly degrade the original P2P applications' performance.

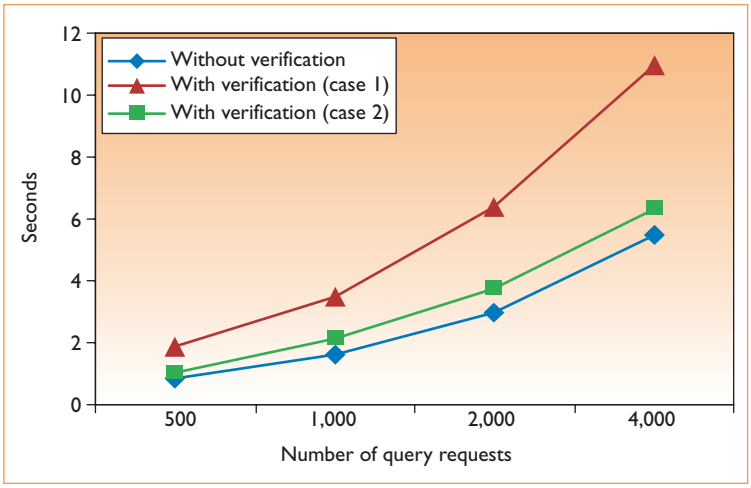

Figure 3. Experimental study of applying our architecture to FTN. The overhead with authenticity and integrity verification is 25 percent beyond that without verification.

The work we discuss here is an initial exploration of the role of TC in P2P systems. Because TC can provide strong assurance in common client platforms, it can enhance or complement traditional security policies and models. We plan to explore general security considerations in P2P with TC technologies, such as anonymity, accountability, and access control.

Acknowledgments

We thank Kumar Ranganathan, Carlos Rozas, and Michael J. Covington of Intel for valuable discussion and input.

References

1. H.J. Wang et al., "Automatic Misconfiguration Troubleshooting with Peerpressure," *Proc. Usenix Symp. Operating System Design and Implementation*, Usenix Assoc., 2004, pp. 245–258.
2. N. Daswani, H. Garcia-Molina, and B. Yang, "Open Problems in Data-Sharing Peer-to-Peer Systems," *Proc. 9th Int'l Conf. Database Theory*, LNCS 2572, Springer-Verlag, 2003, pp. 1–15.
3. D.S. Wallach, "A Survey of Peer-to-Peer Security Issues," *Proc. Int'l Symp. Software Security*, LNCS 2609, Springer-Verlag, 2002, pp. 42–57.
4. D. Milojicic et al., *Peer-to-Peer Computing*, tech. report HPL-2002-57R1, HP Labs, 2002.
5. B. Cohen, "Incentives Build Robustness in Bittorrent," www.bittorrent.com/bittorrentecon.pdf.
6. J. Liang et al., "Pollution in P2P File Sharing Systems," *Proc. IEEE Infocom*, IEEE Press, 2005, pp. 1174–1185.
7. R. Sandhu and X. Zhang. "Peer-to-Peer Access Control Architecture using Trusted Computing Technology," Proc. 10th ACM Symp. Access Control Models and Technologies, ACM Press, 2005, pp. 147–158.

Xinwen Zhang is a PhD candidate in the Lab for Information Security Technology (LIST), George Mason University's Department of Information and Software Engineering. His research interests include access control models and technologies and security issues in distributed computing systems. Zhang has a ~~master~~ engineering from Huazhong University of Science and nology, China. Contact him at xzhang6@gmu.edu.

Songqing Chen is an assistant professor in the Department of Computer Science at George Mason University. His research interests include resource management and security in operating systems, high-performance computing and distributed systems, and Internet content distribution systems. Chen has a PhD in computer science from the College of William and Mary. Contact him at sqchen@cs.gmu.edu.

Ravi Sandhu is a professor in the Department of Information and Software Engineering at George Mason University; director of the Lab for Information Security Technology (LIST); and chief scientist of TriCipher. His research areas are access control models, database security, network security, and distributed system security. Sandhu has a PhD in computer science from Rutgers University. Contact him at sandhu@gmu.edu.