Towards Digital Rights Protection in BitTorrent-like P2P Systems

Xinwen Zhang[a], Dongyu Liu[b], Songqing Chen[b], Zhao Zhang[c], and Ravi Sandhu[d]

[a]Computer Science Lab, Samsung Information Systems America, San Jose, CA 95134, USA
[b]Dept. of Computer Science, George Mason University, Fairfax, VA 22030, USA
[c]Dept. of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011, USA
[d]Institute for Cyber-Security Research, University of Texas at San Antonio, San Antonio, TX 78249, USA

ABSTRACT

Both research and practice have shown that BitTorrent-like (BT) P2P systems are scalable and efficient for Internet content distribution. However, existing BT systems are mostly used for distributing non-copyrighted or pirated digital objects on the Internet. They have not been leveraged to distribute the majority of legal media objects because existing BT systems are incapable of copyright protection. On the other hand, existing Digital Rights Management (DRM) techniques are mainly based on a client-server model, and cannot be directly applied to peer-to-peer based BT systems.

To leverage the efficiency and the scalability of BT systems for Internet content distribution, we propose a novel scheme to enable DRM in existing BT systems without demanding infrastructure changes. In our scheme, each file piece is re-encrypted at runtime before a peer uploads it to any other peer. Thus, the decryption keys are unique for both different peers and difference pieces. In addition, any user can take part in the content distribution while only legitimate users can access the plaintext of being distributed content. To evaluate the performance of our proposed scheme, we have conducted experiments on PlanetLab with an implemented prototype and compared with the original BT system. The results show that our proposed scheme introduces less than 10% of system throughput degradation for copyright protection when compared to BT systems without copyright protection.

Keywords: DRM, BitTorrent (BT), El-Gamal

1. INTRODUCTION

BitTorrent-like (BT) systems have attracted considerable attention due to their scalability and efficiency for content distribution.[1–5] As reported in June 2004, P2P traffic has made up 80% traffic on the Internet, in which the share of BT traffic is 53%.[6] The content distributed through BT systems has evolved from relatively small MP3 files to large and huge files.[7] Recently, some open source projects[1,8] use BT systems to distribute newly released software packages.

As an efficient content distribution vehicle, BT systems distribute the file in small file pieces (e.g., 256 KB per piece) with the assistance of a tracker site. In general, a BT system[9] works as follows. Before an object is distributed, a meta file (normally called `.torrent`) is produced. The meta-file includes the object information (e.g., file name, length), a string of hash values of all file pieces based on SHA1, and the URL of a tracker site. When a client (also called a peer) wants to download the file, it first gets the meta file (e.g., from a public server) and then queries the tracker site. The tracker site always maintains the information of peers who are active

Further author information: (Send correspondence to Songqing Chen)
Xinwen Zhang: E-mail: xinwen.z@samsung.com, Telephone: 1 408 544 5642
Dongyu Liu: E-mail: dliu1@cs.gmu.edu, Telephone: 1 703 993 1536
Songqing Chen: E-mail: sqchen@cs.gmu.edu, Telephone: 1 703 993 3176
Zhao Zhang: E-mail: zzhang@iastate.edu, Telephone: 1 515 294 7940
Ravi Sandhu: E-mail: ravi.sandhu@utsa.edu, Telephone: 1 210 458 6398

(downloading/uploading) in the torrent. Upon a client request, the tracker site responds with a list of active peers on which file pieces are available. The client then starts to download different file pieces from these active peers in parallel. After a piece is downloaded, its hash is calculated and compared with that in the `.torrent` file to verify its integrity. Each downloading peer also reports to the tracker site periodically (typically 30 minutes) so that the tracker site can provide updated active peer information to other peers upon a peer request. In a BT system, normally, a peer that is downloading is also uploading to other peers, and a peer often simultaneously uploads available pieces to a limited number (for example, 5) of peers at a time. Peers are encouraged to upload using the *tit-for-tat* incentive scheme. Once a peer finishes downloading, it becomes a *seed* in the torrent. A seed is a peer that has all file pieces in a torrent, and only uploads to others. In a torrent, there is at least one seed that has the entire file at the beginning.

Many studies through modeling and measurements[2,3,10,11] have shown that BT systems are scalable and efficient. However, existing BT systems have not been used to distribute the majority of legal digital objects. As currently most files shared in BT systems are non-copyrighted or pirated, there are a number of lawsuits concerning the copyright infringement. With verdicts from the Supreme Court of the United States,[12] a copyright protection mechanism is desperately demanded before BT systems could be widely leveraged for distributing copyrighted Internet content.

The currently popular mechanism for copyright protection over the Internet is Digital Rights Management (DRM).[13,14] With DRM, typically, an object is encrypted by a server before distribution. A client downloads an encrypted copy, which is encoded with a unique serial number (ID) or encrypted with a unique key by the server. A license is needed to play or view the content, which includes the decryption key and the usage policy (e.g., a user can only play an obtained movie for 5 times), according to other information such as the user's payment. There are different models to integrate the license management and the enforcement mechanism in DRM. For example, with Microsoft's DRM technology,[14,15] each media file is encrypted with a unique key. The media player on the client side must contact a license server and obtain a license file that includes the key ID and the key seed to recover the unique decryption key before playing. The media player enforces the usage policy defined in the license file, and decrypts the content with the decryption key while playing. On the other hand, in Apple's iTunes and QuickTime that use the FairPlay DRM technology,[13] a music file is encrypted with a master key, which, in turn, is encrypted with a unique user key and encoded in the file. Before playing the music, the client side player must obtain the user key from the server, decrypt the master key, and enforce the usage policy.

Thus, to enforce DRM, each user should obtain a unique copy of an object, either encoded with a unique ID or encrypted with a unique key. This is fairly easy to implement in a client-server model that is mainly adopted in current practice. However, in a BT system, encrypting an object before distribution does not work since peers download exactly same pieces from each other (instead of from a single source) and all clients get the same object. Therefore, the decryption key in the license file is same for all clients, and such a system is not immune to compromised peers. That is, a single compromised license file can break the security of the whole system.

Alternatively, it is also difficult to assign unique IDs or attach other meta information to objects downloaded by different peers in a BT system, which is mandatory in most existing DRM applications. For example, after downloading a copyrighted software, a user needs to obtain a license to install and run the software, which uniquely identifies the downloaded object. Unfortunately, existing BT systems cannot support this since a unique license cannot be defined. Therefore, DRM technologies cannot be applied through this approach.

A direct application of DRM for BT systems could work if the following requirements can be satisfied. That is, each file piece is headed with a unique serial number for a peer. When this peer (uploader) uploads this piece to another peer (downloader), the head information of the downloader is provided by the tracker site and updated by the uploader. However, this requires the trusted behavior of each peer, and assumes that BT client software can recognize and update the head information. Such requirements are not realistic because a general peer cannot be trusted to behave in an expected manner. Thus, the unique challenge to enforce DRM in BT systems lies in the conflict between security requirements and the open environment where a peer downloads different pieces from various sources.

To leverage the capability of BT systems for efficient Internet content distribution, we propose a novel scheme to enable DRM without additional infrastructure changes to existing BT systems. In our scheme, the

content distributed via BT systems is encrypted, and different decryption keys are used for different clients and different file pieces. Thus, DRM relies on different keys to identify different copies. In particular, re-encryption is performed while a peer uploads a file piece to any other peer. Therefore, if a BT system is enhanced with our proposed scheme, any user can participate a torrent to speed up the content distribution, but only legitimate users can access the plaintext of the content, since only legitimate users can get the unique decryption key for each file piece. To study the performance of our proposed scheme, we have implemented a prototype system and experimented on PlanetLab. The experimental results show that for the copyright protection, a BT system with our proposed scheme degrades the system throughput by up to 10%.

The rest of this paper is organized as the follows. Our proposed scheme is presented in Section 2. Section 3 presents the performance evaluation results of our proposed scheme. Some related work is discussed in Section 4 and we make concluding remarks in Section 5.

2. OUR PROPOSED SCHEME

Having discussed the security requirements for DRM in BT systems, we first present the principle of the security algorithm for our proposed scheme, followed by the protocol design details.

In our new scheme, the following assumptions are made, which define the trust boundary of our scheme.

- Similar to the current DRM practice, for digital media content (e.g., video and audio media) we assume that on the client side, a player (or a plug-in of a player) is responsible for decrypting and enforcing the usage policy of an object without releasing the decryption key and plaintext of object pieces. The keys and policies are protected in a license file. For any other type of content, we assume there exists a content viewer with similar functions on the client side. Note that our scheme does not consider content leakage on the client side by hacking the player or the viewer.
- We assume that the original seed and the tracker site of a torrent are trusted by the object owner. That is, the original seed will not upload plain pieces of the object to any peer. It is either the object owner or trusted by the object owner. Similarly, for the tracker site, we assume it is either maintained by the owner, or there is a trustworthy relationship between them.

With the trusted original seed and the tracker site, the content distributed via BT systems is encrypted from the beginning, and different decryption keys are used for different clients and different file pieces. While a peer needs to upload encrypted file pieces to other peers, the encrypted file pieces are re-encrypted so that only the designated users can decrypt the file. After describing the principle of our security scheme, we will present the details of our protocol design and discuss other design issues.

2.1 Principle of Our Proposed Scheme

The security algorithm we leverage is based on the discrete logarithm problem and is similar to El Gamal public key system. We briefly review the El Gamal first and then present the formal definition of our scheme.

Definition 2.1 (El Gamal Cryptosystem). *Let $\mathcal{E}_{eg} = (Gen, Enc, Dec)$ be the standard El Gamal public-key encryption scheme.[16] Gen outputs system parameters g and p, a random number a (used as the private key), and the public key $g^a \bmod p$. Enc is the encryption algorithm with input of message m and outputs $(mg^{ar} \bmod p, g^r \bmod p)$ as the cipher message, where r is a random number. Dec is the decryption algorithm with the private key by dividing $mg^{ar} \bmod p$ with $(g^r)^a \bmod p$ to retrieve the plain message m.*

El Gamal is known to be chosen-plaintext attack secure. Based on the proven security of El Gamal scheme, our scheme is defined as follows (we assume all arithmetic to be *mod p* unless indicated explicitly).

Definition 2.2 (Secure BT System). *Secure BT System is an asymmetric system*, composed of a six-tuple $\mathcal{E}_{bt} = (SGen, PGen, TGen, PEnc, PDec, \mathcal{T})$, each of which is a polynomially computable algorithm as follows.*

*With selective encryption, only a portion of the being distributed objects needs to be encrypted (see section 3).

1. *SGen*(1^n) *is an algorithm generating system-wide parameters* g *and* p*, where* p *is an* n*-bit large random prime,* g *is a generator of the multiplicative group* Z_p^* *of the integers modulo* p*.*

2. *PGen*(P_j) *is a key generation algorithm for a peer* (P_j) *resulting in a random number* s_j *as its private key, where* $1 \leq s_j \leq p-2$*, and the corresponding public key* g^{s_j}*.*

3. *TGen*(P_j) *is a key generation algorithm running on the tracker site. After a peer* P_j *subscribes, the tracker site uses TGen to generate a set of* distinct *random numbers* $r_{1,j}, \ldots, r_{N,j}$ *as the tracker site (abbreviated as TS hereafter) keys of* P_j*, where* N *is the number of pieces of a file. These are used by the tracker site to derive re-encryption keys and by* P_j *to decrypt cipher pieces.*

4. *PEnc*(P_j, m_i) *is performed by the original seed to encrypt piece* m_i *with the public key of a downloader* (P_j) *and the corresponding TS key, i.e., PEnc*(P_j, m_i) *outputs* $m_i(g^{s_j})^{r_{i,j}} = m_i g^{r_{i,j}s_j}$*.*

5. *PDec*(P_j, m_i) *is the decryption algorithm of a downloader* (P_j) *with input of* $g^{r_{i,j}}$ *and its private key, by dividing* $m_i g^{r_{i,j}s_j}$ *with* $(g^{r_{i,j}})^{s_j}$ *to get plain piece* m_i*.*

6. $\mathcal{T}(P_j, P_k, m_i)$ *is a re-encryption function that transforms a ciphertext of* m_i *encrypted by the public key and TS key of* P_j *to a ciphertext of* m_i *encrypted by the public key and TS key of* P_k*. Upon receiving cipher piece* $m_i g^{r_{i,k}s_j}$*,* P_j *encrypts it with input* $g^{(r_{i,k}s_k - r_{i,j}s_j)}$ *from the tracker site, and outputs* $m_i g^{r_{i,j}s_j} g^{(r_{i,k}s_k - r_{i,j}s_j)} = m_i g^{r_{i,k}s_k}$*.*

In short, for each torrent, the tracker site generates system-wide parameters through *SGen* before making the downloading service available. When a peer joins the torrent to download a file, it first subscribes to the tracker site. Upon a successful subscription, a peer performs *PGen* to generate its private key and sends the public key to the tracker site. The tracker site generates a set of random numbers as TS keys for this peer, each for one file piece.

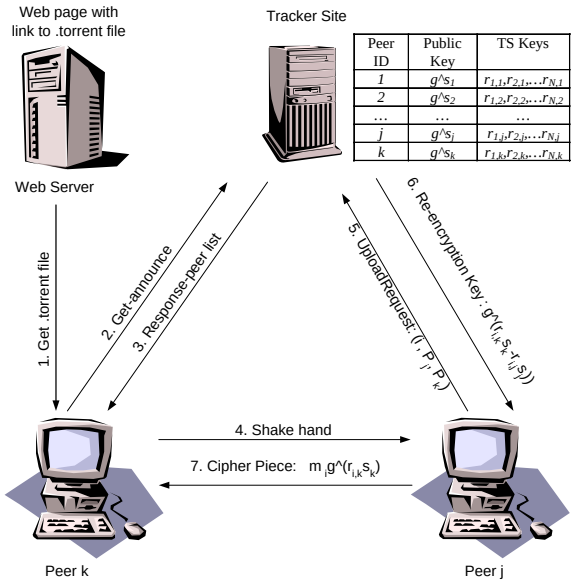

Figure 1. The secure BT system architecture: between two peers

Figure 1 sketches the design of our proposed scheme with an example of data transmissions between two general peers (as stated in *Algorithm 2* shortly). In our proposed scheme, only the original seed (P_o) has all plain pieces. Each peer needs to subscribe to the tracker site before starting to download file pieces. Upon the subscription [†], a peer generates a pair of private key and public key with *PGen*. The private key is kept

[†]The subscription and optional authentication service can be conducted by the tracker site, or a trusted party.

secretly while the public key gets registered with the tracker site. Note that for simplicity of illustration, we do not include a public key authentication mechanism here. Existing approaches such as PKI and SSL can be used for this purpose when a peer subscribes to the tracker site. Corresponding to a peer's public key, the tracker site generates a set of TS keys with $TGen$, each for one piece of the file, and these keys are safely preserved by the tracker site. After a successful subscription, a peer (say P_j) gets a list of active peers that have file pieces available in the system. In the initial step, pieces are only available from P_o. When P_o decides to upload a piece to P_j, it encrypts the plain piece with P_j's public key and the corresponding TS key from the tracker site (with $PEnc$) such that only P_j can decrypt it. When P_j decides to upload an encrypted piece to another peer (say P_k), it first re-encrypts the cipher piece with an input from the tracker site so that only P_k can decrypt it (with $\mathcal{T}$). P_k can also upload this encrypted piece to other peers following the similar procedure. A downloader cannot decrypt received cipher pieces without the decryption keys (provided by the tracker site) included in a license file and enforced by a trusted player. By leveraging the function of the existing tracker site, our scheme does not need additional infrastructure support to distribute and certify public keys of peers.

At a high level, our scheme does not demand any additional infrastructure support from existing BT systems. But the tracker site does assume more responsibility since there is a *peer table* maintained by the tracker site to store active peer ids, and corresponding public keys and TS keys.

2.2 Protocol Design

Encryption Protocols There are two working protocols for downloading copyright protected content in BT systems. The first is for the transmission between the original seed and a downloader, shown as *Algorithm 1* in Figure 2. The protocol works as follows. Initially, the original seed (P_o) has all plain pieces of the file. When P_j joins the system, it gets the `.torrent` file from a public web server (message 1), contacts the tracker site (message 2), and gets a list of active peers (message 3). Suppose P_j wants to download piece m_i from P_o after the "shake hand" step (message 4),

	Algorithm 1	*Algorithm 2*
1	$WS \rightarrow P_j:$ *Get .torrent file*	$WS \rightarrow P_k:$ *Get .torrent file*
2	$P_j \rightarrow TS:$ *Get announce*	$P_k \rightarrow TS:$ *Get announce*
3	$TS \rightarrow P_j:$ *Response peer list*	$TS \rightarrow P_k:$ *Response peer list*
4	$P_j \rightarrow P_o:$ *Shake hand*	$P_k \rightarrow P_j:$ *Shake hand*
5	$P_o \rightarrow TS:$ $UploadRequest(i, P_o, P_j)$	$P_j \rightarrow TS:$ $UploadRequest(i, P_j, P_k)$
6	$TS \rightarrow P_o:$ $(g^{s_j})^{r_{i,j}} = g^{r_{i,j}s_j}$	$TS \rightarrow P_j:$ $(g^{s_k})^{r_{i,k}}/(g^{s_j})^{r_{i,j}} = g^{r_{i,k}s_k - r_{i,j}s_j}$
7	$P_o \rightarrow P_j:$ $m_i g^{r_{i,j}s_j}$	$P_j \rightarrow P_k:$ $m_i g^{r_{i,j}s_j} g^{r_{i,k}s_k - r_{i,j}s_j} = m_i g^{r_{i,k}s_k}$

Figure 2. Encryption algorithms in data transmission

- P_o forwards the request to the tracker site (message 5).
- The tracker site computes the encryption key with the public key of P_j and $r_{i,j}$, and sends it back to P_o (message 6).
- P_o uses $PEnc(P_j, m_i)$ to encrypt the piece with the received key from the tracker site and uploads it to P_j (message 7).

The second algorithm is for data transmission between two normal peers, indicated in Figure 1 and shown as *Algorithm 2* in Figure 2. After the first four normal steps,

- Before P_j uploads cipher piece m_i to P_k, P_j first forwards the request to the tracker site (message 5).
- The tracker site computes the *re-encryption key* which is the division between the public key of P_k with $r_{i,k}$ and the public key of P_j with $r_{i,j}$, and sends the division to P_j (message 6).
- P_j uses $\mathcal{T}(P_j, P_k, m_i)$ to transform the cipher piece with the re-encryption key received from the tracker site and sends the result to P_k (message 7).

Decryption Protocol To decrypt the received cipher piece of m_i, P_k needs $g^{r_{i,k}}$, which is provided by the tracker site. To prevent a peer from sharing plain pieces during downloading, decryption keys are only included in the license file for each user. In particular, after downloading all cipher pieces of an object and before playing that object, the player of a peer contacts a license server and gets a license file. Without loss of generality, we assume the license server is the same as the tracker site [‡]. The license server generates the decryption keys of all pieces, and sends the license file to the user. The decryption process is illustrated as *Algorithm 3* shown in Figure 3. In detail,

	Algorithm 3
8	$P_k \rightarrow TS$ (or license server): $GetLicense(P_k)$
9	$TS \rightarrow P_k : g^{r_{i,k}}$ for $1 \leq i \leq N$, these keys are included in a license file.
10	$P_k : m_i g^{r_{i,k} s_k} / (g^{r_{i,k}})^{s_k} = m_i$ for $1 \leq i \leq N$.

Figure 3. Decryption algorithm in data transmission

- P_k contacts the tracker site (or the license server) by sending a *GetLicense* message (message 8).
- The tracker site generates decryption keys of all pieces, includes them in a license file, and sends back to P_k (message 9). Usage polices are specified in the license file according to related information (e.g., user's payment).
- When playing the content, the trusted player of P_k uses $PDec$ to decrypt cipher pieces with the received keys and its private key. The player enforces the usage policies specified in the license file (message 10).

Note that TS keys are generated by the tracker site upon the subscription of a peer. Thus the decryption keys and the license file can be issued by the license server independently from the content downloading in our scheme. The capability of uploading unique cipher pieces to other peers without decryption enables our scheme to seamlessly work with existing BT systems for efficient content distribution. This feature is enabled by the re-encryption algorithm and centralized TS key management. At a high level, our scheme adds a centralized security architecture above the decentralized content distribution infrastructure of BT systems.

After presenting the details of our proposed scheme, now we show, by the following theorem, that the security of our proposed scheme is guaranteed.

THEOREM 2.3. *Let* $\mathcal{E}_{bt} = (SGen, PGen, TGen, PEnc, PDec, \mathcal{T})$ *be a secure BT Content Distribution System. The problem of recovering a plain piece* m_i *from*

$$(g^{s_j}, g^{s_k}, \ldots, m_i g^{r_{i,j} s_j}, m_i g^{r_{i,k} s_k}, \ldots, g^{r_{i,k} s_k - r_{i,j} s_j}, \ldots, g^{r_{i,j}}, g^{r_{r,k}}, \ldots)$$

is at least as hard as Diffie-Hellman.

Proof Sketch: For simplicity we consider transmitted messages by uploading m_i from P_j to P_k. Due to the unidirectional flow of a single piece in a BT system, m_i is not uploaded from P_k to P_j. Therefore, messages that are publicly available to an adversary are $(g^{s_j}, g^{s_k}, m_i g^{r_{i,j} s_j}, m_i g^{r_{i,k} s_k}, g^{r_{i,k} s_k - r_{i,j} s_j}, g^{r_{i,j}}, g^{r_{i,k}})$.

First, $g^{(r_{i,k} s_k - r_{i,j} s_j)}$ can be derived by $(m_i g^{r_{i,k} s_k})/(m_i g^{r_{i,j} s_j})$ so that it is redundant for cryptanalysis. To find m_i, either $g^{r_{i,j} s_j}$ or $g^{r_{i,k} s_k}$ must be derived. With the knowledge of g^{s_j} and $g^{r_{i,j}}$, the problem to find $g^{r_{i,j} s_j}$ is exactly the Diffie-Hellman problem.[17] The same holds for finding $g^{r_{i,k} s_k}$. This proves that $\mathcal{E}_{bt}$ is at least as secure as Diffie-Hellman [§]. □

[‡]In practice, if the license server is different from the tracker site, the tracker site only needs to send the TS keys of a user to the license server to generate the license file upon the request.

[§]As the typical piece size in a BT system is 256 KB, we do not consider attacks on the plain El Gamal encryption where a much smaller message size is used.[18]

2.3 Integrity Verification and Vulnerability of Our Scheme

In our scheme, after a peer finishes downloading and obtains all cipher pieces, the integrity of each piece can be checked by the player, using the decryption keys included in the license file and the hash values in the torrent file. Instead of the instant integrity verification in original BT systems, the player-performed verification is largely due to the fact that file pieces that a peer downloads are encrypted and the decryption is not possible without a trusted player and the decryption keys in a license file. From the point view of efficiency, this does not increase the overhead of the system, since a corrupted piece can be found and re-downloaded, no matter when it is detected. Optionally, a client can check the integrity of received pieces anytime during downloading, since the license file can be issued separately from the content distribution in our scheme. For example, it can be issued right after the subscription of a peer.

If a client does not perform any integrity check during downloading, our scheme opens a door for content pollution.[19] As an alternative, a slight extension of our scheme can prevent this type of attacks. Suppose the tracker site has a copy of all plain pieces. Upon the subscription of a peer (say P_k), the tracker site knows its public key (g^{s_k}) and generates its TS keys ($r_{1,k}, \ldots, r_{N,k}$) (refer to Section 2.2). Then the tracker site can compute all expected hash values of the cipher pieces that the peer will download, e.g., $\mathcal{H}(m_i g^{r_{i,k} s_k})$ for m_i, where $\mathcal{H}$ is a hash function. The tracker site sends these hash values to the peer upon its subscription. During the downloading process, the piece integrity verification can be performed with the same way as that in original BT systems. Since the tracker site can compute the hash values for a peer offline when the peer subscribes the service, runtime performance overhead can be avoided.

However, as the tracker site is the central server for storing TS keys and generating re-encryption keys, it has the single point failure problem. Fundamentally, this type of attacks exists in the original BT system since the tracker site maintains all active peers in the system, which can be compromised and results in denial of service (DoS) attacks. Recently proposed trackerless BT protocol[20] can partially solve this problem.

2.4 Further Discussion of Alternative Designs

Given that in our scheme, each peer and each piece are allocated a TS key, some may wonder that whether it is sufficient if only a single TS key (for all pieces in a torrent) is allocated to a peer or only a single TS key (for all peers in a torrent) is allocated to a piece . We briefly analyze these alternatives as follows.

- **Scheme 1**: for a peer, its TS keys are identical for all pieces, i.e., $r_{i,k} = r_{i',k} = r_k$ for any $i \neq i'$. However, the problem of this scheme is that piece $m_{i'}$ can be derived with m_i and the encrypted form of $m_{i'}$. For example, an adversary intercepts message 7 in Figure 2 and obtains $c_i = m_i g^{r_k s_k}$ and $c_{i'} = m_{i'} g^{r_k s_k}$, then $c_i / c_{i'} = m_i / m_{i'}$. That is, all cipher pieces are linkable, and a known single plain piece can infer all other pieces. Thus, this only works when all players and license files are always well protected.

- **Scheme 2**: for a piece, the TS keys are identical for all peers, i.e.,$r_{i,j} = r_{i,k} = r_i$. This scheme cannot prevent peer collusion. Specifically, since the decryption keys of P_j and P_k are the same, they can share a single license file and get the same permission with only one payment. Also, a malicious peer can publish a license that everyone can use it, which breaks the copyright protection mechanism.

- **Scheme 3**: each piece has a random number r_i, and the re-encryption key of P_j for m_i is a function of r_i and g^{s_j}, e.g,, $(g^{s_j})^{r_i} = g^{r_i s_j}$. In this scheme, collusion between P_j and P_k can let P_k obtain its TS key; i.e., $(g^{r_i s_j})^{s_k / s_j} = g^{r_i s_k}$. Thus, a single license file can be used by all collusive peers to play different copies they download.

Above discussion strongly indicates that although these alternatives are simpler, they are flawed in general environments and may only work under certain conditions.

3. PERFORMANCE EVALUATION

In our proposed scheme, there is no additional changes required to the original architecture of BT systems. But peers and tracker sites need to perform additional operations, including data transmission, encryption, and transformation, for copyright protection. In this section, we measure the overhead of these operations in order to verify the feasibility of our scheme based on an implemented prototype system. Particularly, we compare the performance of BT systems with and without our newly proposed scheme.

3.1 Encryption/Decryption Overhead Measurement

In our proposed scheme, the performance overhead is mainly due to the additional security functions. To study the overhead, we implement our proposed mechanism in BitTorrent v.4.0.1[21] and evaluated the performance overhead for a general peer and a tracker site. Both the peer and the tracker site are on machines of Pentium 4 CPU 3.4 GHz with 1 GB memory, running Red Hat Linux 9 with gcc 3.2.2. We study the performance with the Botan crypto library 1.6.1,[22] which implements the El Gamal algorithms (key generation, encryption, and decryption).

Table 1 shows the measured transaction time (seconds) for system parameter generation ($SGen$), piece encryption ($PEnc$) and decryption ($PDec$), re-encryption key generation on the tracker site ($Tracker$), and cipher piece transformation by a peer ($\mathcal{T}$), respectively. These experiments were run repeatedly 10 times with the module size, n (the number of bits of the encryption key), varying from 512 bits to 2048 bits.

Table 1. Performance overhead (in second)

Key size (bits)	$SGen$ (s)	$PEnc$ (s)	$PDec$ (s)	$Tracker$ (s)	$\mathcal{T}$ (s)
512	0.029	5.526	8.862	0.0516	0.029
768	0.047	7.993	10.931	0.105	0.037
1024	0.081	10.657	12.350	0.251	0.043
1536	0.195	16.995	17.473	0.371	0.055
2048	0.381	22.985	21.763	0.442	0.062

It is a reasonable conjecture that the system parameter generation and exponential operations are the main overhead sources in our security scheme. As shown in the $SGen$ column, the parameter generation with a larger key size takes a longer time. However, even with a key size of 1024 bits, the key generation is still very fast. Note that this generation is a one-time operation for each torrent.

The $PEnc$ and $PDec$ columns show the time for encrypting and decrypting a single file piece of 256 KB. According to the protocols proposed in Section 2, for each piece, the encryption is only performed by the original seed. Thus, this overhead does not cause running performance degradation of other peers. On the other hand, for a general peer, the decryption is performed after the peer completes downloading the entire file. So this decryption does not affect the downloading performance. However, the encryption speed affects the performance of the original seed to upload encrypted pieces, and decryption speed affects the offline playing performance of the client peer. With a key size of 1024 bits, the decryption speed is about 256 KB/12.350 $\approx$ 20KB/s $\approx$ 160Kbit/s, which is slower than the required playback rate (encoding rate) of some Internet videos. However, this is less likely to be an issue in practice since there are two options for us to improve its performance.

- For media objects, such as videos, the encryption of an entire object is commonly unnecessary. Instead, many *selective encryption* schemes have been proposed and well studied. That is, instead of encrypting the whole object, only some critical data, such as I-blocks and relevant micro-blocks for MPEG videos, in the media file are encrypted. For an Internet MP4 (MPEG4) file, its metadata are critical for constructing the scenes during the playback. That is, the player must use the metadata information to access the right raw data during the playback. If the metadata are encrypted, the playback cannot continue. In a media object, the total size of metadata is generally much smaller than the raw data, thus encrypting and decrypting the metadata take much shorter time than those shown in the table. Typically, selective encryption can increase the overhead by as low as 9%.[23] Similar idea can be applied to non-media files with a partial encryption scheme. This is also on the line of using an asymmetric key system to encrypt a very small amount of important data, making our proposed scheme practical.

- Furthermore, for encryption on the original seed side, the tracker can pre-generate encryption keys and send to it before downloading requests are received. The peers to download pieces from the original seed are predictable since the tracker replies a list of peers to a new peer and does have the public key of a possible requesting peer. With this pre-generated encryption key, the original seed can extensively accelerate the encryption speed. This relies on El Gamal's property that the exponentiations in encryption are independent of the message and can be computed ahead of time. Actually the tracker can determine which peers can download from the original seed and generate all encryption keys with pre-processing.

The tracker site overhead is caused by the TS key generation ($TGen$) and maintenance. As a set of TS keys is assigned to a peer upon its subscription, this can be pre-processed by the tracker site. That is, the tracker site can generate TS key sets in advance and assign one set to a peer upon its subscription. Thus, TS key generation can be performed without causing running performance overhead to the system.

During the procedure of content distribution, the tracker site also needs to generate re-encryption keys for uploading peers. The $Tracker$ column shows the average time to generate a re-encryption key on the tracker site. For example, with a key size of 1024 bits, it takes about 0.25 seconds to generate a re-encryption key. For BT systems with a high request rate, this could be a performance bottleneck. Fortunately, the re-encryption key generation can also be pre-processed before the real data transmission, similar to that of the original seed. Particularly, after a peer obtains a list of active peers from the tracker site (message 3 in Figure 1), by predicting the possible transmissions between this peer and the active peers on the list, the tracker site can generate partial or all possible re-encryption keys in advance.

In addition to the computing overhead, a tracker site needs to maintain TS keys of all active peers, which introduces runtime space overhead. This overhead is closely associated with the number of active peers in a torrent, which is comparably small. For example, as we have studied a workload for software distribution (RedHat 9) through BT,[24] although there were 180,000 clients joining the torrent during 5 months (from April to August, 2003), the number of active peers in every 8 hours is about 150. For an object of 1GB, there are about 4000 pieces with a piece size of 256 KB. If a key size of 1024 bits is used in our proposed scheme, the total space overhead for key storage is $150 \times 4000 \times 1024/8 = 75\ MB$. This overhead is acceptable with a modern machine.

As indicated in Figure 2, in our new scheme, there is extra overhead to the uploading peer for transforming cipher pieces and this transformation has to be conducted online. During the transformation, the new cipher piece is obtained by multiplying the re-encryption key received from the tracker site. As this is a multiplication operation, it is expected to introduce trivial overhead. The $\mathcal{T}$ column in Table 1 shows this overhead with a piece size of 256 KB. The resulted transformation overhead is trivial.

3.2 Communication Overhead Measurement on PlanetLab

In our new scheme, in addition to the overhead caused by security related operations that we have evaluated, there is also additional communication overhead. For example, for each piece downloading, the uploading peer needs to get a re-encryption key from the tracker site. Since each peer only connects to a limited number of other peers, this communication overhead is trivial and would not result in significant performance degradation.

However, for the tracker site, since it is the central point in the system, frequent communications may delay its response to downloading requests and affect the throughput of the system. It is important to ensure that it is not a performance bottleneck.

To evaluate the performance overhead on the tracker site in our scheme, we have implemented a prototype system and experimented on PlanetLab. In the experiments, 4 dedicated seeds are set up with an uploading speed of 200 KB/s. Randomly selected 120 PlanetLab nodes are used as downloaders, from Asia, Europe, and United States. The object is a 640-MB file. Both the seeds and the tracker site are running on dedicated machines of Celeron CPU 2.4 GHz with 1 GB memory, and Linux Fedora 2.6.9 and Python 2.3.4.

The communication overhead on the tracker site would increase with the increasing number of concurrent downloading peers. But there are a limited number of accessible nodes on current PlanetLab. As we cannot leverage many (e.g., 1000) peers at the same time, we instead change the file piece size. With a fixed total size

of a file, different piece sizes result in different numbers of file pieces. As the tracker site has to be involved for each piece downloading, decreasing the piece size can increase the communication overhead on the tracker site. Thus, if a regular piece size is 512 KB with 120 downloaders (we set this as the baseline in our experiments), when the piece size is 32 KB, it is equivalent to increase 16 times communication load on the tracker site. That is equivalent to have 1,920 concurrent downloading peers.

In the experiments, we start all peer downloading almost simultaneously (within one minute) such that all peers are active during our experimental period. We run each experiment with varying file piece sizes for one hour. At the end of the one hour, all peers are downloaders. Therefore, the communication load on the tracker site is close to the peak.

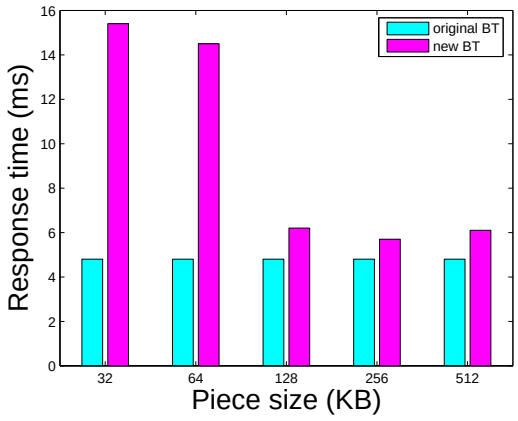

Figure 4. Tracker's response time with different piece sizes

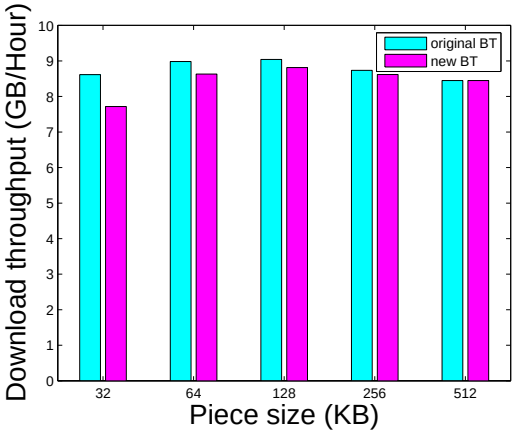

Figure 5. System throughput with different piece sizes

Figure 4 shows the average message response time of the tracker site for a single downloading request. It is not surprising that in all cases the response time of the tracker site with the original BT system is close to each other, since a peer only contacts the tracker site for the peer list and reports its status. The response time difference between our new scheme and the original one decreases as the piece size increases, due to the decreasing load on the tracker site with a larger file piece size from 32 KB to 512 KB. With a piece size of 128 KB and 120 concurrent downloaders (equivalent to 480 concurrent downloading peers with a piece size of 512 KB) or higher, the average response time is slightly increased. Even when the piece size is 32 KB (equivalent to 1,920 concurrent downloading peers), the average message response time is still less than 16 ms averaged over 25,000 measured values within one hour. Because of the traffic variance along time in PlanetLab, the average response times with a piece size above 256 KB are slightly different in our new scheme.

Thus, when the concurrent active peers are at a few hundred, which is the case according to existing studies,[10, 24] the security cost in our new scheme does not affect the response time to client requests and the system scalability much.

To evaluate how much the delayed response time affects the system throughput, particularly when the file piece size is small, we also evaluate the entire system throughput after the system has run for one hour. Figure 5 shows the system throughput comparisons of our new scheme with the original BT scheme. The result shows that the difference of the system throughput decreases as the piece size increases. Even with a 32-KB file piece, the difference between the system throughput is less than 10%. We believe that such throughput degradation is acceptable for most practical systems, and would not affect the system scalability. Again, the system throughput with the original BT protocol is slightly changing with varying file piece sizes because of uncontrollable network traffic variances in PlanetLab.

4. RELATED WORK

Recently, a lot of studies have been performed on the measurement and modeling on BT systems. In 2004, Izal et al. analyzed a five-month workload of a single BT system for Redhat source distribution[24] and concluded that

the BT system is scalable upon flash crowds. To investigate the feasibility of BT systems for data distribution, in Ref. 1, authors studied BT traffic of thousands of torrents over a four-month period. The work in Ref. 2 studied an eight-month BT workload and found that the arrival, aborting, and departure processes of downloaders do not follow the Poisson distribution, which were assumed in the previous modeling work.[3] This modeling work used a simple fluid model to characterize the overall performance of BT systems. It verified the scalability of BT systems and analyzed the effectiveness of BT incentive mechanism based on game theory. In Ref. 5, authors analyzed service capacity of BT systems, and found that multi-part downloading helps P2P systems to improve the performance during flash crowd period. Guo et al. found that BT systems have service stability and availability problems after flash crowds.[10] Bindal et al. studied the impact of neighbor selection on the traffic locality in BT systems[25] while the effectiveness of the incentive mechanism used in BT is analyzed in Ref. 26.

In terms of security mechanisms, several proxy re-encryption schemes and applications have been proposed. Blaze et al.[27] proposed *atomic proxy cryptography* in which a proxy can divert a ciphertext of Alice to Bob with a delegated key. Ivan and Dodis[28] improved this scheme with unidirectional proxy re-encryption. Proxy re-encryption schemes based on El Gamal algorithm have been extensively studied in Ref. 29. A similar scheme based on multi-key RSA has been proposed in Ref. 30 for video-proxy systems. An implied assumption in most of these schemes is that the proxy is trusted or semi-trusted by the server. A major difference of our scheme from these proxy-based schemes is that in BT systems, a peer is both a client and a proxy, thus the peer cannot be trusted to have any re-encryption keys (i.e., TS keys) due to possible collusion between peers. In fact, the collusion attack between the proxy and general clients has been pointed out in Ref. 31 for the video-proxy scheme. In addition, since each peer in a BT system can be a proxy to distribute content, our scheme eliminates the performance bottleneck as in the proxy-based scheme, where the re-encrypting operation is performed in a central proxy.

5. CONCLUSION

The emergence of BT systems on the Internet has attracted significant attention. Plenty of research and practice has shown and demonstrated its good scalability and efficiency for large file distribution. However, to date, it has not been leveraged to distribute the majority of copyrighted digital content over the Internet. In this paper we propose a security mechanism based on the existing BT infrastructure to enable copyright protection. To the best of our knowledge, our study is among the first attempt for this purpose. To evaluate our proposed strategy, we have implemented a prototype system and conducted real experiments in PlanetLab. The evaluation results show that our scheme can achieve comparable content distribution efficiency to the original BT system. That is, to enable the copyright protection in such P2P systems, our proposed scheme causes less than 10% degradation of the system throughput.

6. ACKNOWLEDGMENTS

We would like to thank anonymous reviewers for their helpful comments on this paper. The work is supported by NSF grants CNS-0509061 and CNS-0621631.

REFERENCES

1. A. Bellissimo, P. Shenoy, and B. Levine, "Exploring the use of bittorrent as the basis for a large trace repository." Technical report 04-41, University of Massachusetts, Amherst, 2004.
2. J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The bittorrent p2p file-sharing system: Measurements and analysis," in *Proc. of the 4th International Workshop on Peer-to-Peer Systems (IPTPS'05)*, 2005.
3. D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *Proc. of the ACM SIGCOMM*, 2004.
4. K. Skevik, V. Goebel, and T. Plagemann, "Analysis of bittorrent and its use for the design of a p2p based streaming protocol for a hybrid cdn." Technical Report, Department of Informatics, University of Oslo, 2004.
5. X. Yang and G. Veciana, "Service capacity of peer to peer networks," in *Proc. of IEEE INFOCOM*, 2004.
6. A. Parker, "The true picture of peer-to-peer filesharing." CacheLogic Research, 2004.

7. "Study of file formats traversing the peer-to-peer networks." CacheLogic Research, 2005.
8. "A tracker for fedora core and other linux distributions." http://torrent.dulug.duke.edu/.
9. B. Cohen, "Incentives build robustness in bittorrent," in *Proc. of the 1st Workshop on Economics of Peer-to-Peer Systems*, http://www.bittorrent.com/bittorrentecon.pdf, 2003.
10. L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurements, analysis, and modeling of bittorrent-like systems," in *Proc. of ACM SIGCOMM Internet Measurement Conference*, 2005.
11. T. Karagiannis, A. Broido, k. c. N. Brownlee, and M. Faloutsos, "Is p2p dying or just hiding?," in *Proc. of the Globecom*, (Dallas, Taxas), 2004.
12. "Supreme court rules against file swapping." CNET News.com, June 27, 2005.
13. "Fairplay." http://en.wikipedia.org/wiki/FairPlay.
14. "Microsoft's digital rights management scheme - technical details." http://cryptome.org/ms-drm.htm.
15. "Windows media digital rights management (DRM)." http://www.microsoft.com/windows/windowsmedia /drm/default.aspx.
16. T. Gamal, "A public key cryptosystem and signature scheme based on the discrete logarithm," *IEEE Transactions of Information Theory* (4), 1985.
17. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.
18. D. Boneh, A. Joux, and P. Q. Nguyen, "Why textbook ElGamal and RSA encryption are insecure," in *ASIACRYPT 2000, Lecture Notes in Computer Science*,
19. J. Liang, R. Kumar, Y. Xi, and K. Ross, "Pollution in p2p file sharing systems," in *Proc. of IEEE Infocom*, 2005.
20. "Bittorrent - trackerless." http://www.bittorrent.com/trackerless.html.
21. "Bittorrent website." http://www.bittorrent.com/.
22. Botan crypto library, "http://botan.randombit.net/."
23. J. Wen, M. Severa, W. Zeng, M. Luttrell, and W. Jin, "A format compliant configurable encryption framework for access control of video," in *IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on Wireless Video*, pp. 545–557, June 2002.
24. M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. A. Hamra, and L. Garc'es-Erice, "Dissecting bittorrent: Five months in a torrent's lifetime," in *Proc. of the 5th Passive and Active Measurement Workshop, Antibes Juan-les-Pins, France, LNCS Vol. 3015*, 2004.
25. R. Bindal, P. Cao, W. Chan, J. Medval, G. Suwala, T. Bates, and A. Zhang, "Improving traffic locality in bittorrent via biased neighbor selection," in *Proceedings of the the 26th International Conference on Distributed Computing Systems (ICDCS)*, (Lisboa, Portugal), July 2006.
26. M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bittorrent?," in *Proceedings of the 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI)*, (Cambridge, MA), April 2007.
27. M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Proc. of Eurocrypt'98, LNCS 1403*, (Helsinki, Finland), May 1998.
28. A. Ivan and Y. Dodis, "Proxy cryptography revisited," in *Proc. of the 10th Annual Network and Distributed System Security Symposium (NDSS)*, (San Diego, California), February 2003.
29. G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," in *Proc. of the 12th Annual Network and Distributed System Security Symposium (NDSS)*, (San Diego, California), February 2005.
30. S. Yeung, J. Lui, and D. Yau, "A case for a multi-key secure video proxy: Theory, design, and implementation," in *Proc. of ACM Conference on Multimedia*, 2002.
31. Y. Wu and F. Bao, "Collusion attack on a multi-key secure video proxy scheme," in *Proc. of the 12th annual ACM international conference on Multimedia*, 2004.