Escrow Services and Incentives in Peer-to-Peer Networks

Bill Horne
STAR Lab, Intertrust Tech.
821 Alexander Road
Princeton, NJ 08540
bhorne@intertrust.com

Benny Pinkas
STAR Lab, Intertrust Tech.
821 Alexander Road
Princeton, NJ 08540
bpinkas@intertrust.com

Tomas Sander
STAR Lab, Intertrust Tech.
4750 Patrick Henry Drive
Santa Clara, CA 09540
sander@intertrust.com

ABSTRACT

Distribution of content, such as music, remains one of the main drivers of P2P development. Subscription-based services are currently receiving a lot of attention from the content industry as a viable business model for P2P content distribution. One of the main problems that such services face is that users may choose to redistribute content outside the community of subscribers, thereby facilitating large-scale piracy. Digital Rights Management (DRM) systems typically employ tamper resistance techniques to control this risk. We propose a system architecture that uses economic incentives instead of tamper resistance to motivate users to keep the content within the subscription community. The key technical contribution we make is to integrate a P2P file sharing service with an escrow service that reliably "pays" the party that is serving up the content. The payment itself can be realized in a number of ways, using "actual" money or bonus points such as frequent flyer miles.

Moreover, our architecture facilitates trust between two unacquainted parties by offloading risk to a trusted third party, which can acquire a revenue stream by assuming this risk. To implement the escrow service securely we use cryptographic techniques, such as encryption, hashing, and error correcting codes. Our system motivates users to serve up content of high quality and verifies that users only share legitimate content and not spam, viruses or content that is not part of the subscription. We thereby address other important security concerns in P2P systems and problems like the free-rider phenomenon.

1. INTRODUCTION

1.1 Motivation

Peer-to-peer networks for distribution and sharing of files have enjoyed enormous popularity. Many networks with different characteristics are being developed, and one may argue that P2P file sharing of content has proven to be a "killer app". It is fair to say however that the problem of building a P2P file sharing system for content distribution that simultaneously accomplishes goals desirable for users, such as a high quality of service, *and* goals of copyright holders who may wish to restrict unlicensed copying and distribution of content ("piracy") has been an unsolved problem. In this paper we suggest an architecture that attempts to achieve these goals simultaneously.

EC'01, October 14-17, 2001, Tampa, Florida, USA.

Let us first review some of the issues concerning quality assurance for end users and piracy concerns for copyright holders in P2P networks.

Spam: This refers to the problem of peers sharing files which do not match their description. In a Napster like network this might correspond to files that contain music which is different than what their title suggests, poorly ripped music which is of low quality or simply unsolicited advertisements.

Free riding: Currently users do not have any motivation to serve content in a P2P network. This might lead to "free riding" by many of the users (as was indeed found in a study of the Gnutella network [1], and was suspected to lead to reliability and scalability problems for P2P networks).

A method to counteract free riding is to incentivize users to serve files, e.g. to allow users to get paid for their service (either in real money or by a different commodity, like frequent flyer miles or network based currency). However, the introduction of payments raises the problem of ensuring that the service that is paid for is of high quality, and that payments are valid.

Fighting black markets – attracting users to join "legitimate" networks. To compete with networks that distribute unlicensed content, "legitimate" P2P networks may wish to incentivize users to use them by offering added services. The additional service that our system provides is better quality assurance, and assurance that payments can be exchanged in a reliable way for high quality content.

Content certification: A P2P network might have a policy of providing file sharing for only a restricted class of files (for example, according to a recent ruling in the Napster case the network is not supposed to distribute songs that appear in lists of copyrighted songs provided by the record companies).

1.2 Incentivizing Users to Participate in a P2P Network

Digital rights management (DRM) solutions encode content together with rules that govern its usage (for example a rule might say "This content can be used three times for free. The content must be purchased in order to be used more times"). The content must be kept inside encrypted containers that can only be used by specific clients that enforce the rules. These containers must remain secure as far as possible along the content's usage route (e.g., in the case of music, it would be desirable if the container remained encrypted until it reached the digital to analog converter of the music card so that only an analog version of the content would be accessible in the clear). Until such clients are ubiquitous there will be however locations in which the digital content is available in the clear, and might be stolen. In addition, the security of the clients is based on their tamper resistance which might be broken.

Given this state of the technology and the fact that most content is available in an unprotected format anyway (say on music CDs) it is reasonable to assume that cleartext digital versions of content will become available to pirates. These files can be redistributed at little or no cost, leading to black market models such as the one Napster has been criticized for by major rights holders. A natural solution to this problem consists of creating incentives for users to both join a "legitimate" peer-to-peer content distribution network and also to play by the rules. Most likely such incentives would be financial. Why not allow Bob, who is running a peer-to-peer server on his machine, get paid by Alice who downloads content from his server? If Bob can make money from the system then he will be less likely to redistribute the content in the clear. More generally if one assures that Bob would only get paid as long as he plays by the rules he is much more likely to follow those rules.[1]

One of the problems in a peer-to-peer system where relatively unknown parties participate in financial transactions is a breakdown of trust. Why should Alice trust Bob and vice-versa? How can Alice be sure that Bob will deliver an original copy of the content, not some poorly "ripped" version of it? Similarly, how can Bob be assured that Alice will meet her financial obligation? And finally, how can the original content provider get a piece of the transaction? The system we design attempts to answer all of these questions.

The main contribution of this paper is to design reliable and trustworthy mechanism for enabling payments in exchange for content among participants in a P2P network. We argued earlier how such a mechanism can be used to encourage end users to play by the rules - instead of solely relying on tamper resistance or legal remedies for accomplishing compliance. We find it as an interesting problem to study and assess in more detail the various business models that are enabled by our proposed mechanism. This economical analysis is however beyond the scope of this paper in which we focus on the cryptographic aspects of how such a mechanism can be built.

1.3 Our Solution

Quality assurance of content in P2P networks can be solved rather efficiently in theory if the content is served for free: Some trusted party can examine the content and certify it using digital signatures. Any recipient that downloads content could verify that the file is signed by a party that it trusts. The problem is much more complicated if the party that serves the content should get paid, and in particular if it is paid for every file that it serves, by the recipient of that file. A payment should be exchanged for a high quality file. The payment cannot be sent before the recipient examines the file (since otherwise a corrupt distributor might not send the required content), but on the other hand the file must not be used before the payment is verified (since otherwise a corrupt recipient might not pay for the files it downloads). The payment and file transfer operations should essentially be run simultaneously, as a single atomic transaction.

The problem we describe is essentially the problem of *fair exchange* of digital goods, which enjoyed considerable attention by the research community (as is described in Section 1.4). The solution we suggest involves a trusted third party, which we call the escrow server. The system ensures that if a dispute arises it involves the trusted party and one of the users, but does not involve two users. We assume that disputes of the former type can be resolved more easily than disputes between two users. The operation of the trusted party is very simple. In particular,

- The operation of the escrow server does not require direct online interaction with the party that is serving the content.
- The escrow server is not required to receive the entire content from the recipient. Rather, the messages that are sent between the escrow server and the recipient are of constant length. Furthermore, the content can be kept secret from the escrow server, if this is desired.
- The escrow server does not need to keep a database of content files that it supports. The operation of the escrow server can be completely stateless, namely its response to a query sent by a recipient depends on the query alone, and does not depend on any other information that is stored in the server.

Traditionally, the use of a trusted third party (TTP) that is involved in every transaction is considered non-optimal. We believe however that this is the right solution for the P2P quality assurance problem, for several reasons.

- Although cryptographic research has shown that the information a trusted third party needs to learn in these types of protocols, and the requirement for the TTP to be online can be greatly minimized, this might not necessarily be desirable for a company that would like to be in the business of selling trust, like our escrow server.
- The simple design of the system and the simple operation of the escrow server make our system more usable and scalable than current systems that do not use a trusted third party.
- From a business perspective, the management of the P2P network might wish to be involved in every transaction for branding purposes, e.g., to demonstrate its relevance, deliver a clearly visible and easy understandable value to consumers, and gain a reputation that would encourage users to participate in the network ("Every day millions of users consider us as their trusted party, come do business with us...").

[1]Theoretically there is the danger that an illegitimate service could try to compete with the legitimate one by also offering payments to Bob. However this does not seem to be much of a threat in the real world, because enabling financial transactions on a large scale typically involves the cooperation of major financial institutions which are unlikely to cooperate in such illegitimate efforts, or could be easily forced not to do so.

- End users in a P2P system typically need to go online and contact a (centralized or decentralized) directory service to find servers of the content they are looking for. Thus requiring them to contact the escrow server (or better piggyback and integrate this additional interaction with the interaction with the directory service) does not seem to be a significant extra cost.

The system can support the distribution of content that is generated by large content providers, where the vast majority of peers operate as distributors but not as contributors. It could also support distribution of content that is generated by small content providers (e.g. "garage bands"). It can support variations in which only specific parties are allowed to distribute specific content, or alternatively enable parties to distribute content even if they care to pay to use it.

Verification of different encryptions of the same content Another contribution of this work, which is of separate interest, is an efficient solution for the following scenario: *One party generates many different encryptions of the same plaintext and sends them to different recipients. A trusted third party knows the original plaintext and the different keys that were used for encryption. Each recipient (who does not have a decryption key) wants the third party's help in verifying that the ciphertext it received is correct.* We describe how to perform this verification without requiring the third party to decrypt each ciphertext. (In fact, the online overhead of the third party is independent of the content size.)

1.4 Related Work

The field of research that is most related to our work is fair exchange of information. There have been many suggested fair exchange protocols, both with and without trusted parties, as we describe below. These protocols, however, are non-optimal for the peer-to-peer scenario, for three major reasons:

- The protocols suffer from scalability problems in large networks, since they require several (or many) public key operations per information exchange, or require several communication rounds between the parties.

- Some protocols are not scalable to handle the exchange of large files, since they require the sender or the recipient to send the entire file to the trusted party. This overhead might be reasonable for a single transaction that involves only two parties. However, a P2P network might have a very large number of parties but only a limited number of trusted parties, and the latter would need enormous bandwidth resources in order to support sharing of large files over the entire network.

- Previous work on fair exchange considered a one-on-one scenario that involves just two parties exchanging information. A P2P network has interactions between many parties and introduces new problems (such as the problem of a party downloading a large encrypted file from one server, that expects to be paid for the decryption key, and then obtaining the decryption key from a different source). We show how to solve this problem in section 4.

The initial work on fair exchange designed two-party and multi-party protocols for secure computation that solve the problem of early termination without a trusted party [5, 16, 7, 13]. These protocols are either generic or discuss the specific problem of fair tossing of a random coin. The main drawback of these protocols is that each exchange requires a considerable number of exponentiations and a number of rounds that is linear in the security parameter (and therefore quite large).

In the scenario that involves a trusted third party there have been many discussions on the design of protocols for certified email, where an email message should be exchanged for a receipt. This work is surveyed in [20], see also [6, 8, 9, 10]. The problem examined by these papers is simpler than the problem we discuss, but nevertheless the protocols suggested are less efficient in the sense that they require several rounds of communication or sending the entire message to the trusted party.

Franklin and Reiter [12] described a protocol for fair exchange using a "semi-trusted" third party, which is semi-trusted in the sense that although it might be corrupt it does not conspire with either one of the other two parties. The protocol is rather efficient, although it requires a few exponentiations and therefore might not be scalable to large networks. However, the protocol only enables exchanging two keys between the parties, and does not solve the problem of checking whether the information that is encrypted with the keys is the desired content.

There have been several recent suggestions of protocols for optimistic fair exchange (e.g. [2, 3]). These protocols require a trusted party but this party is not required to intervene during an "optimistic" scenario in which none of the parties misbehave. The basic idea of these protocols is that the parties send to each other commitments to the values they exchange, and prove that these commitments can be opened by the trusted party. These protocols are rather impractical for large networks.

2. THE BASIC DESIGN OF THE ESCROW SERVER SYSTEM

2.1 Preliminaries

2.1.1 Participating Parties

The system we envision allows various parties to engage in the exchange of digital content, C, for a payment, P. For example C might be a multimedia file including audio or video, or various kinds of documents including books, papers, medical records, business information, email, etc.

The system contains three parties: (1) A Party that serves (or sends) content, denoted by S. (2) A party that receive content, denoted by R. (3) An escrow server, denoted by ES.

We assume that there are many servers and recipients of content, but only a single ES whose identity is known to all parties. The primary assumption that underlines the design of the system is that the escrow server is a trusted and respectable party. We thus assume that ES never tries to maliciously cheat other parties, and therefore disputes that it might have with other parties can be resolved rather easily.

We further assume that users have a way of paying each other. This functionality could possibly be facilitated by the ES[2]. In principle the ES could use payment technolo-

[2]For example this can be done by requiring users to sign

gies, such as electronic cash or Paypal [18], or outsource the payment part completely. The payment might also be informational rather than financial, i.e., the recipient might be willing to fill out a survey or reveal personal information in lieu of a financial payment in order to get access to the content.

2.1.2 *Cryptographic primitives:*

We assume that the parties agree on the following cryptographic primitives:

- A symmetric encryption scheme E, such as DES or Rinjdael, the recent AES standard.
- A collision resistant hash function H. The property that this function should have is that it is infeasible to find two values x and y such that $H(x) = H(y)$. This function could be realized by using SHA-1 or MD5.
- We assume that there are secure and authenticated channels between any two parties. These could be achieved using public key cryptography, such as using SSL or IPSEC.
- Some variants of our system require that parties be able to encrypt information that can be later read by the ES. This must be done non-interactively and therefore SSL and IPSEC cannot be used. The encryption can be done using public key cryptography, or using symmetric keys which are shared between ES and each of the parties.

2.2 The Basic Scheme

We describe here a generic outline of the operation of the system. We later describe how to tune the system to obtain different properties.

Setting terms: We assume that the sender, S, is willing to serve content, C, which is requested by the recipient R. We also assume that through some mechanism R finds out that S is willing to serve C, and these two parties agree on the terms of a transaction in which R should obtain C from S. The terms must include a description of C and the payment that R should pay. They might also bind S to a specific file C, by providing in advance a commitment to C or simply the value $H(C)$.[3] To simplify the initial description of the system we assume that the price of C does not depend on the identity of S and R, and is known to ES. In practice, users willing to exchange content could be matched by the (centralized or decentralized) directory service of a P2P network. Terms of service could be agreed upon when users join the network.

Information at the escrow server: In the protocol that we describe, S encrypts C with a key K and sends the encryption $E_K(C)$ to R. R obtains the decryption key K from the escrow server ES. In order to take part in the protocol ES requires the correct hash of the *encrypted* content, $H(E_K(C))$, and the key K that decrypts the content. There are several ways in which this information can become known to ES. For now we assume that S sent the pair $\langle K, H(E_K(C))\rangle$ to ES in a preprocessing stage, together with a description of C. ES might have also asked to receive the content C itself and verify that C agrees with its description and that $H(E_K(C))$ is the required value. This is depicted in step 0 of Figure 1.

Content delivery: Content delivery is composed of the following steps, depicted in Figure 1.

- Party S prepares an encrypted version of the content, $E_K(C)$, namely C encrypted using a randomly chosen key K (step 0 in Figure 1).
- S sends $E_K(C)$ to R (step 1).
- In order to obtain the key K, R computes the hash of the encrypted content $H(E_K(C))$ and sends it to ES, together with a description of the content C it desires to obtain, and the identity of S. It also sends a payment for the content C, which will only be processed if the content is valid (step 2).
- ES compares the hash it receives from R with the hash it stores (step 3). It also checks whether the description of C which it receives from R is the same as the description of C that it stores. There are two possible outcomes to this comparison:

  If the two hash values and the corresponding descriptions are equal, ES verifies that R's payment is valid. If it is, it informs R that the encryption is legitimate and sends it the key K (step 4). If R's payment is invalid than the protocol terminates.

  R receives K and uses it to decrypt $E_K(C)$ and obtain C. According to predefined agreements, ES forwards part of R's payment to S, and might keep part of the payment as its own fee (step 5).

  If the two hash values and the descriptions are not equal, ES notifies R that the hash values do not match (step 6). (This can happen because S send to R an illegitimate copy of C, or because sent the wrong hash value to ES.) In this case R is not required to pay anything and the protocol ends. (If we wish to enable the system to decide who is to blame we should require the message from S to R be signed. The signature identifies a single message and then this message can be used to verify whether S sent the wrong content or not.)

2.3 Security of the Basic Protocol

The security of the system is based on the trustworthiness of ES. The following two simple propositions state that if ES behaves according to the protocol then both R and S receive their required outputs. Therefore, if any of the peers decides that it did not receive the correct output from the protocol it should challenge the operation of the ES, rather than challenge the operation of another peer. A dispute between a peer and ES is similar to a dispute between a consumer and a business, and should be easier to resolve than a dispute between peers.

---

on to the ES with their credit card number or their bank account. The ES keeps and updates a balance of the payments due among various users and batch processes them once economical thresholds are reached to justify charging or crediting a user's account.

[3]The value $H(C)$ could serve in practice as a good commitment to C, although it does reveal some information about the value of C. (For example, $H(C)$ enables the recipient to test whether the promised content C equals some candidate file C', by simply comparing $H(C)$ to $H(C')$. A commitment does not enable this comparison.)

	S		R		ES
0	$E_K(C)$				$H(E_K(C)), K$
1		$E_K(C)$ $\rightarrow$			
2			$E_K(C)$	$H(E_K(C)), S$, payment$\rightarrow$	
3					compare two hashes
success:					
4				$\leftarrow$ K	verify payment
5			decrypt using K		pay S
failure:					
6				$\leftarrow$ "hashes do not match"	

Figure 1: The basic protocol.

Proposition 1. *If R sends a payment to ES and ES behaves according to the protocol, then R is able to obtain the content C, or else ES is convinced that S did not act according to the protocol.*

Proof: In the beginning of the protocol ES obtains a pair of values $\langle \alpha, \beta \rangle$, that should correspond to $\alpha = H(E_K(C))$ and $\beta = K$. Consider the following two cases.

Assume that the values $\langle \alpha, \beta \rangle$ that ES holds correspond to the hash of the encryption of C and to the decryption key. If R receives $E_K(C)$ from S then the protocol ensures that R receives a key K from ES which enables it to decrypt the content. If R receives a different value γ from S then the collision resistance of H implies that $H(\gamma) \neq \alpha$, since it is infeasible to find two different values that are hashed by H to the same value. Since R's payment is processed only after ES verifies that the hash that R sends is equal to α, the misbehavior of S is discovered before R's payment is processed.

Assume now that the values $\langle \alpha, \beta \rangle$ that ES holds do not correspond to an encryption of C and to the corresponding decryption key (we assume that C is defined in the "setting terms" stage). If R receives a value which is not hashed to α then R is not even required to pay, as is described above. However, it might be the case that R receives a value which does hash to α, and later, after paying and receiving K, R finds out that the decryption of this value with K is not C. In this case R can present to ES the value that it received from S (which is hashed to α), and demonstrate that decrypting this value using K does not yield C. Since H is collision resistant, S cannot confront this claim by presenting a different value that is hashed to α and is the encryption of C with the key K. $\square$

Proposition 2. *If R is able to obtain the content and ES behaves according to the protocol, then S is paid.*

Proof: The security of the encryption scheme E implies that R cannot obtain the content C if it is only given an encryption of C with a randomly chosen key, as is the case in our system. The protocol defines that R receives the decryption key K only after ES verifies R's payment. Since ES is assumed to follow the protocol it must then transfer to S the part of the payment that belongs to it. $\square$

A note on hiding the content from ES: The basic protocol does not require ES to have a copy of C. The only information that ES requires is the hash of the encryption of C, and the decryption key[4]. The basic protocol can therefore be used even if it is preferable that ES does not learn the content. For example, R and C might exchange some very valuable content and only require the help of ES in case of a dispute.

Using a universal one-way hash function: It seems tempting to require the hash function to be universal one-way , rather than collision resistant (the former requirement was shown to be a weaker requirement than the latter). A universal one-way hash function should satisfy that given a random x it is infeasible to find a y such that $H(x) = H(y)$. (Collision resistance is a stronger property which means that it is impossible to find any collision, for any x, y of the attacker's choice. The attacker has freedom in choosing both X and y.) The problem with using universal one-way functions in our system is that it might be possible to find a collision pair that satisfy $H(x) = H(y)$. This might ma the system insecure, for example if the encryption scheme is a one-time pad (since in this case given x and content C of the same length it is easy to find a K which encrypts C to x, namely $K = C \oplus x$). This enables a corrupt S to send x and K to ES and send y to R. R then computes $y \oplus K$ which is different than C.

It seems that the above problem with using universal one-way functions cannot be solved unless we put additional requirements on the encryption scheme E. Something like "given a ciphertext and a plaintext it is infeasible to find a key which binds them". An additional problem is that the input to the hash function is not random, but rather an encryption of a specific content. This ciphertext might come from a small subspace for which it is easy to find another input that is hashed to the same value.

2.4 Applying the Basic Scheme in Different Scenarios

The basic scheme can be adapted to many different scenarios in which files are shared. The files can be generated by large content providers that are affiliated with ES, or by small independent content providers, or by the party that shares the file (S). The method by which the recipient searches for the file and recovers it can be arbitrary (see Section 5). We demonstrate the possible options of implementing the system through some illustrative examples.

[4]If the encryption algorithm E is deterministic then knowledge of $H(E_K(C))$ and K releases some information about C (for example, it enables to verify whether C is equal to a certain value). If it is desired to prevent this leakage of information the system should use a semantically secure encryption function E.

2.4.1 Large content providers

In this scenario there are a few large content providers, for example the large music labels or movie studios. These providers distribute their content through a network of small peers. This may be financially advantageous to the content providers, who can offset the costs of setting up and maintaining a high performance server system with the requisite bandwidth requirements to distribute the content on their own. In particular this approach seems valuable for the distribution of video files, since such a system might impose bandwidth requirements which are too demanding for a centralized server. It can be assumed that the content providers have a continuing relationship with ES. At an extreme, each content provider can operate an ES to handle the distribution of its own content.

The system could operate in the following way. Before content becomes available through the peer network, the content provider encrypts it with a random key K and provides the hash of the encrypted content, and the key, to ES. This encrypted copy is therefore certified by the original content provider. The content provider then distributes this copy to peers that are willing to serve it for a fee. (In other words, this is a Napster-like network in which content providers actually pay peers to do the distribution work.) When a user R asks to receive this content from a server S they run the basic protocol described above. ES compares the hash that it receives from R to the hash of the encrypted copy that it received from the content provider. If the hash is correct the transaction takes place. Fractions of the payment can be delivered to different parties, e.g., the content owner is paid for letting R use the content, S is paid for delivering it, and ES is paid for providing the escrow service.

A variant of the system might be very useful for distribution of content with a future release date (say, of a new movie). Since the content would probably be very popular in the period immediately following the release date, a centralized server might not be able to supply all the demand for the content during that period. It can distributed an encrypted version of the content before that period, to servers that are willing to distribute the content, but provide the decryption key (via the escrow server) only after the release date.

Sharing without using: Note that the sharing parties need not know the plaintext version of the content. A party S who is interested in making money from distributing the content, but is not interested in obtaining the content for its own use, can keep the encrypted version of the content, $E_K(C)$, and never ask for the key K. This does not prevent it from distributing the content to other parties who are interested in it.

2.4.2 Small content providers

The proposed system can also support many small content providers, such as independent musicians. Each of them can generate content, distribute it (or provide it to other peers for distribution), and use ES in order to reduce the risk in interaction with peers who are interested in receiving the content.

There could be many variations in the design of the system in this case:

The system could support a single content author distributing its creations, or enable the content creator to use other peers as distributors.

ES could check the encrypted version of the content before it begins supporting its distribution, in order to verify that it indeed contains the encryption of the right content. Alternatively it could store the hash of the encrypted content without checking it, and only request the encrypted content in case of a dispute.

The content provider could send the hash of the encrypted content to ES before it begins to use the system. Alternatively, the hash could be delivered to the ES via the receiver R, as is explained in Section 3.

The system might send the same encrypted copy of the content to every recipient, or it might support independent encryption of different copies of the content (as is described in Section 4).

S might choose the encryption key, or the encryption key might be chosen by the content provider.

3. INDIRECT COMMUNICATION FROM S TO ES

The basic protocol assumes that ES has knowledge of $H(E_K(C))$ and K. Assuming that the content is generated by a different party, this information must be communicated from that party to ES. It makes most sense to assume that $\langle H(E_K(C)), K\rangle$ should be sent from the server S to ES, since S is held responsible for the quality of the information it sends to the recipient R.

The simple approach suggested in the basic protocol requires S to send a separate message to ES which contains $\langle H(E_K(C)), K\rangle$ (step 0 in Figure 1). This approach makes sense when S distributes many copies of C to many recipients, and sends only a single message with the hash and the decryption key to the ES. The drawbacks of this approach are that ES has to store a database of values that correspond to different contents and that S has to open a communication channel with ES. The latter problem is especially severe if S sends a different message to ES for every copy of the content that it sends to potential recipients. For example, if S desires to use a different key for each transaction.

The following method does not require S to open a communication channel with ES. It is depicted in Figure 2.

- In the beginning of the protocol S knows the public key PK of ES, and only ES knows its private key PK.
- The message that S sends to R contains, in addition to $E_K(C)$, an additional part which contains an encryption of $\langle H(E_K(C)), K\rangle$ that can only be decrypted by ES (step 1 in Figure 2). (More on the encryption method below.)
- The recipient receives this message and forwards to ES the hash $H(E_K(C))$ of the first part, as well as the second encrypted part, $\langle H(E_K(C)), K\rangle$ (step 2).
- ES decrypts the second part using its private key (step 3), and proceeds as in the basic protocol.

This method has the advantage of not requiring S to send any message directly to ES, and enabling ES to be stateless and do not keep a database of hash values of encrypted contents. The recipient R is essentially used by the system as a communication channel from S to ES. This makes sense since in any case S has to send a message to R, who then sends a message to ES.

	S		**R**		**ES**
0	$E_K(C), \mathbf{PK}$				**SK**
1		$E_K(C)$, $\mathbf{E_{PK}(H(E_K(C)), K)}$ $\rightarrow$			
2			$E_K(C)$	$H(E_K(C)), S$ $\mathbf{E_{PK}(H(E_K(C)), K)}$ $\rightarrow$	
3					decrypt with SK and compare two hashes
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Figure 2: A protocol without direct communication from S to the ES. (The parts which are different from the basic protocol are in bold.)

Which encryption method to use? R essentially serves as a communication channel between S and ES. One should keep in mind that R and S might have conflicting motivations and therefore R might want to tamper with the message that it is required to transfer. It is therefore recommended that S encrypts this message with a non-malleable encryption scheme [11] (also known as chosen ciphertext secure in the post processing mode), such as the Cramer-Shoup scheme [8] or OAEP+[19]. The use of a non-malleable encryption scheme ensures that R cannot change the ciphertext in any way that changes the result of the decryption that ES performs[5].

It is natural to use a public key encryption scheme whose decryption key is only known to ES. The same public encryption key could be used by all servers to encrypt messages to ES.

An alternative approach is to use private key cryptography. ES could generate a basic key K_{ES} and give each potential server S_i a personal key K_i which is generated as $K_i = F(K_{ES}, S_i)$, where F is a pseudo-random function keyed by the key K_{ES}. (This function could be implemented by the using the encryption scheme E and computing $E_{K_{ES}}(S_i)$). The key K_i is only known to ES and to the server S_i, which uses the key to encrypt messages to ES. If the system employs this method then the servers could use private key encryption which is considerably more efficient than public key encryption. The major drawback is that each server should first contact ES in order to obtain its personal key. This is considerably less convenient than publishing the public key of ES in a way that makes it available to all potential servers.

Certifying C by ES: The method described here binds S to a single possible content as the source of the ciphertext $E_K(C)$. This enables disputes to be resolved if R later complains that the decryption of $E_K(C)$ did not result in the required content. It is preferable though to ensure the quality of $E_K(C)$ before R's payment is processed. This can be achieved in the following way: (1) S sends C to ES in a preprocessing stage and ES verifies that $H(E_K(C))$ indeed corresponds to the encryption of C with K. (2) ES should then sign $H(E_K(C))$ and a description of C, and send the signature to S. (3) When S later interacts with R it should send it these signed values. (4) R should forward these to ES together with the other messages it sends it. (5) ES verifies the signature and if it is correct it uses $H(E_K(C))$, K, and the description of C, as if they were taken from its own database

[5]If the scheme that we describe is used in isolation then the only damage that R can do by changing the message sent from S to ES is changing the hash (which results in ES not sending it the key K), or changing the key K. In any case it does not seem that R could benefit from this behavior. If we take a broader view then it is obvious that R could damage the reputation of S, or take advantage of side effects that depend on the context in which the system is used.

4. INDEPENDENT ENCRYPTION OF DIFFERENT COPIES

The escrow method we suggest introduces a new type of problem. It is caused by the fact that servers in the system are willing to serve an encrypted version of the content without being paid, trusting that they will be paid when the recipient obtains the decryption key. The source of the problem is that there is a single encrypted copy of the content. To be more specific we detail two examples in which dishonest parties can use this feature:

Leaking the decryption key: Assume that the content is a video file and that distributing it requires considerable network resources. One user R' can obtain the encrypted version of the file and pay for the decryption key. It might not be economical for R' to distribute the plaintext version of the file to other users, but R' might sell (or give for free) the corresponding decryption key. This enables other users to contact servers, obtain the encrypted version and decrypt it with the key they have obtained from R' for free, without having to contact ES and pay for the decryption key. If this happens then servers might spend considerable amount of resources distributing the encrypted versions of the files, without ever being paid for them.

Fooling ES into paying a different S: Assume that a dishonest server S' is colluding with a user R'. This user can then obtain the encrypted content from a different server S, and then contact ES, present it with the hash of the encrypted content and with a payment, and claim that the content was obtained from S'. In this case S' gets paid although S is the server that did the work.

We suggest a technique to solve this problem by encrypting the content with a different key each time it is sent to a different recipient. . A naive implementation of this approach introduces a new problem: it prevents ES from storing a single hash that can be compared with each possible encryption of the content. We present below two methods that employ error correcting codes to enable ES to check the hash values of different encryptions by checking a fixed set of values, and without having to decrypt each encryption.

The first method is slightly simpler but requires ES to store a copy of the entire content, whereas the second method does not require ES to store any additional information.

Before presenting our solutions we describe two simpler solutions, which have inferior properties, for the problem of verifying the hash of different encryptions of the same content. The simplest solution is to require the *sender* S to send to ES the hash of the current encryption and the corresponding decryption key. ES then compares this hash to the hash it receives from R, and continues in the protocol if they are the same. The problem with this solution is that while ES does compare two hashes, it does not know whether they correspond to the required content C. Furthermore, it does not know whether encrypted versions that were sent to different recipients are indeed different encryptions of the same content. While R can complain to ES in case the decryption of the message does not reveal the required content, there is no way for ES to certify the content before R's payment is processed.

A different solution requires ES to store the entire content, and receive from S the key K that was used to encrypt the content to R. ES should then encrypt the content using K, hash the ciphertext and compare the result with the value received from R. If these are the same then it concludes that S sent R the required encryption. The problem with this solution is that the computation overhead of ES is of the same order as the overhead of S, since ES has to re-encrypt the content every time S sends it to a new recipient. This overhead is not reasonable for any large scale implementation.

4.1 A Straightforward Method Based on Error Correcting Codes

The first step of this method is to divide the content into k blocks (e.g., each block containing one second of music or video). These k blocks are encoded using an error correcting code into n blocks, called the "encoded content blocks". The error correction property should ensure that any t of the n encoded blocks enable the reconstruction of the original k content blocks. This requirement means that the parameters must satisfy $k \leq t \leq n$. (It is desirable of course to have t as close as possible to k.) To be concrete assume that $t = 0.75n$, and therefore if one obtains uncorrupted copies of at least 75% of the encoded blocks it is able to reconstruct the original content.

The system operates in the following way (depicted in Figure 3). First S must encode the content into n blocks, and provide these n blocks to ES, who must store them. When S sends the content to R it picks a random key K and uses it to independently encrypt each block of the encoded content. It then sends these blocks to R together with the key K encrypted with the public key of ES. (Note that the use of error correcting codes means that the length of the communication from S to R increases by a factor of n/k).

After receiving this message R picks a small random set of ℓ of the encrypted encoded blocks (where ℓ is a parameter to be defined below). It sends to ES this set of encrypted blocks, together with their indexes and the encrypted message that is directed from S to ES.

ES decrypts K and uses it to decrypt the ℓ blocks that R sent it. It then compares each of these blocks with the corresponding block of the encoded content which it stores. If any of these comparisons fails ES concludes that R did not receive a correctly encoded content, otherwise it concludes that S behaves according to the protocol and sends the key K to R.

In order to analyze the system, note that it is sufficient for R to obtain t encoded blocks in order to decrypt the content. Since ES checks ℓ randomly chosen blocks, the probability that S can prevent R from retrieving the content while not being detected by ES is at most $(\frac{t-1}{n})^\ell$. The value of ℓ should be set to make this probability sufficiently small. For example, $\ell = 2n/(n-t)$ is sufficient in order to lower this probability to about 10%.

Which code to use? The scheme could either use error correction codes, or use more efficient erasure codes. We first examine the use of error correction codes. The codes have the property that given n blocks, such that at least t of them are correct (although the receiver does not know which ones these are), the receiver can reconstruct the original message. The system could use the list decoding algorithm of Guruswami and Sudan [14] which requires that $t \geq \sqrt{nk}$. In particular if we wish that $t = 0.75n$, then the system should set $n \geq 16k/9$. This means that the message from S to R is less than twice its original size, and R should send to ES 8 blocks ($\ell = \frac{2n}{n-t} = 8$) in order to bound the error probability by 10%.

Erasure codes are more efficient but require that the recipient identifies which blocks are correct and which blocks are corrupt. The system could use such codes if ES signs each encoded block, and S sends the signed blocks to R. ES must then verify that the blocks it decrypts are correct and contain its signature. This test ensures that with high probability R will be able to recover at least t correct blocks which are signed by ES. The reconstruction operation of R should therefore be to decrypt each block, examine its signature, and if it is correct use the block in the reconstruction of the original content. As for the choice of erasure code to be used, one could use Reed-Solomon codes, for which it suffices to set $t = k$ (meaning that to get $t = 0.75n$ we should set $n = 1.33k$, reducing the communication overhead from S to R). The drawback of Reed-Solomon codes is that the computation overhead is essentially $O(n^2)$ (if one does not use FFT methods which are asymptotically more efficient but less efficient in practice). A better choice is to use the Tornado codes of Luby et al [15]. These codes have a slightly larger t, $t = k(1+\varepsilon)$, but the reconstruction time is linear. For all practical purposes one could set $\varepsilon = 5\%$, increasing the communication overhead by only 5% (again, R should send to ES 8 blocks in order to bound the error probability by 10%).

4.2 An ef£cient method based on error correcting codes

The main drawback of the method described above is that it requires ES to store a copy of the encoded version of every block. The mere requirement of storing a database of information about the various content files is a considerable burden on ES. Furthermore, the database should be very large since the content files themselves are rather large, and their encoded versions are even larger.

A verification method that does not require ES to store any data about the content files can be based on ES comparing hash values of a certified copy of the encoded blocks to the values it receives from R. A first and simple version can be used if ES performs both the initial certification of

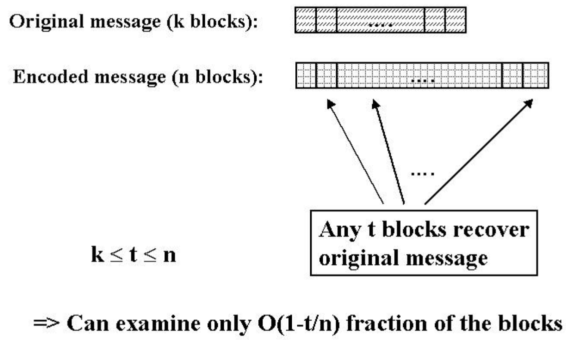

Figure 3: Using error correcting codes for content verification.

the content (namely, running a verification that the encoded blocks represent the description of the content), and the examination of the encrypted content that R receives from S. A slightly more elaborate method can be used if there are different parties running the tasks of certifying the initial encoded versions of the content and checking the encrypted content files that are received by the recipients.

A verification method based on MACs: This method is based on using message authentication scheme, MAC (see [4] for a description of the properties of MAC schemes). ES should choose a random key K_M to key a MAC scheme, and must keep this key secret. If S wants to use the system to share content C, it sends to ES in an initialization stage the encoded blocks of the content C, denoted $B_1, \ldots, B_n$. ES verifies that these blocks encode C. If they do it sends to S, for every $1 \leq i \leq n$, the output of the MAC function, keyed by K_M, on the input $(\hat{C}, i, B_i)$, where $\hat{C}$ is an agreed upon description of the content (such as a name of a movie, etc.). Namely it sends to S the set of values $\{MAC_{K_M}(\hat{C}, i, B_i)\}_{i=1}^{n}$.

When S sends the encrypted content to R it encrypts each block independently, together with the corresponding MAC. R, as in the previous scheme, chooses ℓ random encrypted blocks and sends them to ES, together with their indexes and the description $\hat{C}$ of the content. ES then decrypts these blocks and checks the MAC values using the key K_M. If all the MACs are correct it decides that the encrypted version of the content is correct.

Intuitively, the security of this scheme is based on the fact that the unpredictability of the MAC function ensures that without knowing the key K_M it is impossible for S to change the encoded block without being identified by ES. The overhead of this scheme is negligible since the size of the output of the MAC and the overhead of computing it are negligible.

The drawback of this method is that only ES (and not R) can verify the MACs, since only it knows that MAC key. This property means that the system must use error-correction codes, rather than more efficient erasure codes, since R cannot identify which blocks are corrupt before running the error correction procedure.

A verification method based on efficient signatures: The scheme described above uses a MAC function which is secure as long as its key K_M is kept secret. It is reasonable to use this scheme if the same party performs both the certification of the content and the examination of the encrypted copies. If, however, the system includes several ES servers or other servers that are responsible for certifying content, there is a reasonable chance of one of these servers becoming corrupt. When this happens the key K_M is compromised and the scheme is no longer secure. This threat can be solved if the parties use public key signatures to certify that the content is correct. Namely, each of the ES servers has its own private/public key pair, and it uses its private key to sign each encoded blocks, instead of using a MAC. Each ES should then know the public key of the other ES servers, or the public key of a certification authority, and can use them to verify the signatures. This method enables to revoke the keys of an ES server which is found to be corrupt. An advantage of this method is that R can check the signatures on blocks by itself, and can identify corrupt blocks before running the error correction procedure. This enables the system to use erasure codes which are more efficient than error correction codes.

The main problem is, however, that the overhead of computing the signatures is much higher than that of computing MACs. In order to achieve greater efficiency ES servers could use a hash tree [17]. Namely, the server that certifies the encoded blocks should use a collision resistant hash function H to generate a binary tree in which the ith leaf is the hash of (i, B_i), and each internal node is the hash of its two immediate descendants. ES should then sign the hash of the root of the tree and $\hat{C}$, and send it to S. S should encrypt the ith block together with the corresponding leaf, and the immediate descendents of the nodes in the root from this leaf to the root. It should also send R the signature generated by ES. In order to verify the encrypted content that it receives from S, R should forward this signature and ℓ randomly chosen encrypted blocks to ES. ES can decrypt the blocks, generate the nodes in the paths from the corresponding leafs to the root, and verify that they match the signed value of the root node.

The only increase in overhead that is incurred by using this version is the additional $\log n$ hash values that should be sent with every block. This overhead is rather small since for all practical purposes $\log n$ could be smaller than 20 and each hash value can be 16 bytes long, while the blocks themselves are much longer. The overhead can be reduced if S sends the whole hash tree separately from the blocks, encrypting each node separately. The total number of hash values that S sends to R is reduced to $2n$, compared to

$n \log n$ if each block contains the hash values that are needed in order to verify it. That is, the overhead per block is only two hash values, which are of negligible length. R should forward to ES the $\ell \log n$ nodes that enable it to verify the hashes of the ℓ blocks that are chosen by R.

The system could of course use a combination of the first method and the second one. Namely divide the n encoded blocks into m groups of n/m blocks, arrange each group in a tree of depth $\log(n/m)$ and sign the m roots. This combination presents a tradeoff between the number of signatures that has to be computed and sent (which is m), and the number of hash values S should send to R (which is $n \log(n/m)$, if the hash values are encrypted together with the block).

5. ORTHOGONALITY TO SEARCH METHOD

The escrow systems that we suggest are completely orthogonal to the method in which the files are shared and distributed among the peers. The escrow service and the verification of the content can be done by the peer-to-peer service, or by a separate non-affiliated service. To be more concrete we discuss three models of peer-to-peer file distribution:

Centralized directory service: This model includes networks, such as the Napster network, in which files are distributed by peers, but there is a single central directory server that knows which files are available at which locations. Peers who are searching for files go first to this directory server to find the location in which the files are stored. It is natural, but not necessary, to design the ES as part of the directory service.

Distributed directory service: This model includes networks, such as the Gnutella network, where there is no single central directory server that knows the locations of the available files. The search for the files is run as a distributed process that crawls the network. In this case there is no natural candidate to run the ES service, but there is also no reason why such a service cannot be implemented as part of the network or as a separate service. It there is less trust in the parties then is natural to implement a distributed ES service, by using threshold secret sharing methods.

Distributed file storage: In this model files are not stored at a single server. Instead, a file is divided into many blocks encoded with an error correcting code, and each block is stored at a different location (this is the storage method used by Mojonation.com). The use of error correcting codes ensures that the file can be restored even if part of the peers that store parts of the file are unavailable or corrupt. The directory service might be centralized or distributed. Our ES service can cooperate with this storage system, where the use of error correcting codes we suggest in Section 4 might use (or might not use) the same error correcting code as the storage method.

6. OPEN PROBLEMS

The major direction for continued research should be an experiment with implementing the system as part of a peer-to-peer network. This experiment should give evidence about the efficiency of the design, and about the implications of the service of the escrow server.

An interesting technical question is to find other efficient methods for testing whether different ciphertexts correspond to encryptions of the same plaintext with different keys. Another technical question is the design of an escrow system consisting of several servers, in which only authorized subsets of servers can perform transactions, and smaller subsets of servers cannot cheat any one of the users. This type of system reduces the amount of trust that is given to a single escrow server.

7. REFERENCES

[1] A. Adar and B. Huberman, *Free Riding on Gnutella*, manuscript, 2000.

[2] N. Asokan, M. Schunter and M. Waidner, *Optimistic protocols for fair exchange*, ACM Conf, on Computer and Communications Security 1997, pp. 7-17.

[3] N. Asokan, V. Shoup and M. Waidner, *Optimistic fair exchange of digital signatures*, EUROCRYPT '98, pp. 591-606.

[4] M. Bellare, R. Canetti, H. Krawczyk *Keying hash functions for message authentication*, Crypto '96.

[5] M. Blum, *How to exchange (secret) keys*, ACM Tran. on Computer Systems 1:175-193, 1983.

[6] A. Bahreman and J.D. Tygar, *Certified electronic mail*, Proc. of NDSS 1994, pp. 3-19.

[7] R. Cleve, *Controled gradual disclosure schemes for random bits and their application*, Crypto '89, pp. 573-588.

[8] T. Coffey and P. Saidha, *Non-repudiation with mandatory proof of receipt*, Computer Communications Review, 26(1):6-17, 1996.

[9] B. Cox, J.D. Tygar and M. Sirbu, *NetBill security and transaction protocol*, Proc. 1st Usenix workshop on electronic commerce, pp. 77-88, 1995.

[10] R.H. Deng, L. Gong, A. Lazar and W. Wang, *Practical protocols for certified electronic mail*, J. of Network and Systems Management, 4(3):279-297, 1996.

[11] D Dolev, C. Dwork and M. Naor, *Non-Malleable Cryptography*, Siam J. on Computing, 2000.

[12] M. Franklin and M. Reiter, *Fair exchange with a semi-trusted third party*, ACM Conference on Computer and Communications Security 1997, pp. 1-5.

[13] S. Goldwasser and L. Levin, *Fair Computation of General Functions in Presence of Immoral Majority*, Crypto '90.

[14] V. Guruswami, M. Sudan, *Improved Decoding of Reed-Solomon and Algebraic-Geometric Codes*, FOCS 1998: 28-39.

[15] M. Luby, *Tornado Codes: Practical Erasure Codes Based on Random Irregular Graphs*, RANDOM 1998.

[16] M. Luby, S. Micali and C. Rackoff, *How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin*, 25th FOCS, pp. 11-21, 1984.

[17] R.C. Merkle, *A digital signature based on a conventional encryption function*, Crypto '89, pp. 428-446, Springer-Verlag, 1990.

[18] Paypal.com, `http://www.paypal.com`.

[19] V. Shoup, *OAEP reconsidered*, manuscript, 2000.

[20] J. Zhou and D. Gollmann, *Evidence and non-repudiation*, Journal of Network and Computer Applications, London: Academic Press, 1997.