Fileteller: Paying and Getting Paid for File Storage

John Ioannidis
ji@research.att.com
AT&T Labs Research

Sotiris Ioannidis
sotiris@dsl.cis.upenn.edu
University of Pennsylvania

Angelos Keromytis
angelos@cs.columbia.edu
Columbia University

Vassilis Prevelakis
vp@drexel.edu
Drexel University

November 12, 2001

Abstract

FILETELLER is a credential-based network file storage system with provisions for paying for file storage and getting paid when others access files. Users get access to arbitrary amounts of storage anywhere in the network, and use a micropayments system to pay for both the initial creation of the file and any subsequent accesses. Wide-scale information sharing requires that a number of issues be addressed; these include distributed access, access control, payment, accounting, and delegation (so that information owners may allow others to access their stored content). In this paper we demonstrate how all these issues are addressed using a micropayment architecture based on a trust-management system. Utilizing the same mechanism for both for access control and payment results in an elegant and scalable architecture.

Keywords: Micropayments, trust management, network storage, access control.

1 Introduction

We have set out to create an architecture for a file storage marketplace, which we have called FILETELLER. We envision that there will be Network Storage Providers (NSPs) that make available file storage and charge for it. These storage providers can be traditional ISPs, new businesses created solely for the purpose of providing large amounts of file storage, cooperating organizations, or even individuals with a fast Internet connection and spare capacity on their home systems who want to make such capacity available for a small fee.

The actual price paid for this service can be in real money, in loyalty points, or even in closed-system currency ("play-money" used only as a bookkeeping device among users of the system). We use the trust-management-based micropayments system described in [3] and summarized in Section 1.3 to handle the very small payments that would accompany use of the FILETELLER service. Furthermore, we use a trust-management system for all the access control as well, thereby integrating the payment and the access control mechanisms; access is granted not only on the basis of who someone is or what credentials they hold, but also on whether they can pay for it.

1.1 Motivation

Probably the most onerous operation for individual computer users is backing up their file systems. Local disks have grown tremendously in size in recent years; it is not uncommon for a home computer to have 50-100GB of disk space. At the same time, the usual backup media (tape and CD-R) have not grown much in capacity, and tape storage in particular is much more expensive relative to the price of disk than it used to be. Commercial endeavors are already offering "network drives" where, for a small monthly fee comparable to what an ISP charges for Internet access, users are entitled to some amount of network-attached storage. We believe that wide availability of such offers, paid for with fixed monthly fees or on a per-use basis at a low per-transaction cost, are going to become increasingly available in coming years. These "network drives" can simply be used as backup storage, or can even be used as the main repository of all user data (backed up by the NSP), while the file system on the user's machine acts as a file cache, much like AFS[5] does.

Off-site backup storage and provisions for disaster recovery, once the purview of large, well-funded organizations, is becoming increasingly important to smaller companies, academic institutions with limited IT budgets, and even individuals. While setting up off-site storage operations for just that purpose may be expensive, groups of similar in nature and geographically separated organizations may want to share their excess capacity. By using a system such as FILETELLER, they can do so in a straightforward way without worrying about one member abusing other members' resources, because of the implicit accounting (even if it is just "play money,") that goes with the system.

Another reason to have network-attached storage is file sharing. Today, if network users wish to make some of their files available to others on the Internet, they can place them on their Web pages and publish the corresponding URL, or in older times "put them up for anonymous FTP." Neither mechanism provides for easy access control or the possibility of financial gain for the owner of the information. Web pages can be password-protected, and FTP directories (even anonymous FTP directories) structured in such a way as to only allow authorized parties to access the information; however, the only users allowed to access the files are those that are already known to the system. This, as is the case with existing network file systems, limits access only between users in the same administration domain and requires considerable involvement of the administrator and the file owner in the access control management. When replication for availability and performance scalability purposes is taken into consideration, the management complexity increases drastically. The WebDAV system [10] solves some of these problems, but its main purpose is group manipulation of shared files, not occasional access.

1.2 Non-Goals

It is tempting when designing a new file storage system to attempt to solve all existing problems and issues in network file storage. We have specifically avoided doing that; we are not providing a new name space, a new file access method, a new cryptographic file system, or even a new network file system (in the sense of NFS or CIFS). We are not guaranteeing either integrity or confidentiality of the stored information. We do require the existence of a transport mechanism, such as HTTP, that can carry all the meta-data necessary to accomplish the management functions of FILETELLER, but we have endeavored to rely on existing protocols for the actual data transfer and user/server authentication.

1.3 KeyNote Microchecks

The micropayment system introduced in [3] forms the basis of our approach. The general architecture of the microbilling system, is shown in Figure 1. In FILETELLER, the Check Guarantor plays the role of *Provisioning*, the Network User plays the role of *Payer*, and the Network Storage Provider (or another NU acting as an NSP) plays the role of the *Merchant*. *Clearing* is done either by a financial institution if real money is used, or by a selected user of the system in the case of loyalty points or "play money."

In this system, Provisioning issues KeyNote[2] credentials to Payers and Merchants. These credentials describe the conditions under which a Payer is allowed to perform a transaction, and the fact that a Merchant is authorized to participate in a particular transaction. When a Payer wants to buy something from a Merchant, the Merchant first encodes the details of the proposed transaction into an *offer* which is transmitted to the Payer.

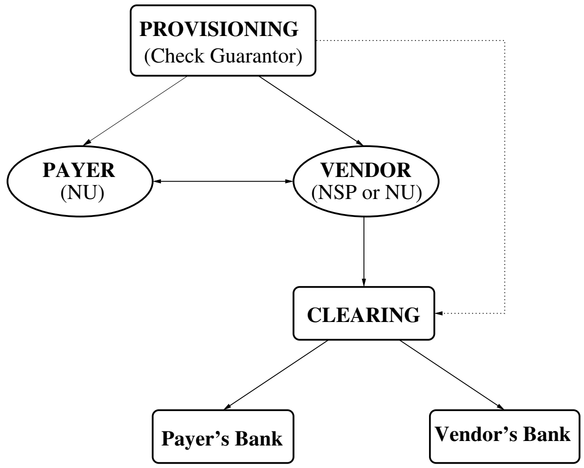

Figure 1: **Microbilling architecture diagram. We have the generic terms for each component, and in parentheses the corresponding players in FILETELLER. The arrows represent communication between the two parties: Provisioning issues credentials to Payers and Merchants; these communicate to complete transactions; Merchants send transaction information to Clearing which verifies the transaction and posts the necessary credits/charges or arranges money transfers. Provisioning and Clearing exchange information on the status of Payer and Merchant accounts.**

If the Payer wishes to proceed, she must issue to the Merchant a microcheck for this offer. The microchecks are also encoded as KeyNote credentials that authorize payment for a specific transaction. The Payer creates a KeyNote credential signed with her public key and sends it, along with her Payer credential, to the Merchant. This credential is effectively a check signed by the Payer (the Authorizer) and payable to the Licensee. The conditions under which this check is valid match the offer sent to the Payer by the Merchant. Part of the offer is a nonce, which maps payments to specific transactions, and prevents double-depositing of microchecks by the Merchant.

To determine whether he can expect to be paid (and therefore whether to accept the payment), the Merchant passes the action description (the attributes and values in the offer) and the Payer's key along with the Merchant's policy (that identifies the Provisioning key), the Payer credential (signed by Provisioning) and the microchecks credential (signed by the Payer) to his local KeyNote compliance checker. If the compliance checker authorizes the transaction, the Merchant is guaranteed that Provisioning will allow payment. The correct linkage among the Merchant's policy, the Provisioning key, the Payer key, and the transaction details follow from KeyNote's semantics[2].

If the transaction is approved, the Merchant should give the item to the Payer and store a copy of the microcheck along with the payer credential and associated offer details for later settlement and payment. If the transaction is not approved because the limits in the payer credentials have been exceeded then, depending on their network connectivity, either the Payer or the Merchant can request a transaction-specific credential that can be used to authorize the transaction. Observe that this approach, if implemented transparently and automatically, provides a continuum between online and offline transactions tuned to the risk and operational conditions.

Periodically, the Merchant will 'deposit' the microchecks (and associated transaction details) he has collected to the Clearing and Settlement Center (CSC). The CSC may or may not be run by the same company as the Provisioning, but it must have the proper authorization to transmit billing and payment records to the Provisioning for the customers. The CSC receives payment records from the various Merchants; these records consist of the Offer, and the KeyNote microcheck and credential from the payer sent in response

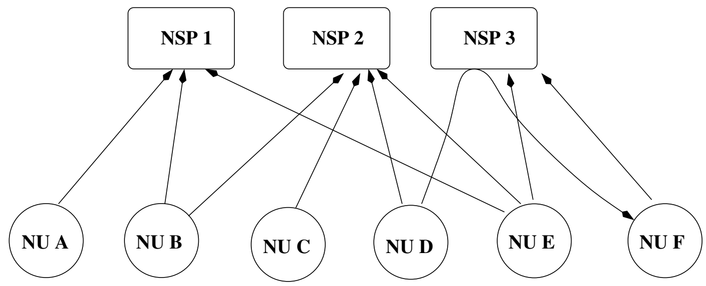

Figure 2: **Flow of payments in our system. Users issue microchecks to the various NSPs as they create and access files, or to other users (through the pay-back scheme).**

to the offer. In order to verify that a microcheck is good, the CSC goes through the same procedure as the Merchant did when accepting the microcheck. If the KeyNote compliance checker approves, the check is accepted. Using her public key as an index, the payer's account is debited for the amount of the transaction. Similarly, the Merchant's account is credited for the same amount.

This architecture makes it possible to encode risk management rules for micropayments. Previous electronic systems have focused on preventing fraud and failure, rather than on managing it. Unfortunately, the prevention mechanisms can be too expensive for micropayments, making a risk management approach particularly attractive.

2 Architecture

We start by giving a high-level overview of the FILETELLER architecture, and then proceed with a usage example. We close this section with a brief security analysis of our architecture.

2.1 FILETELLER Operation

The main architectural principle behind FILETELLER is that users with properly guaranteed microchecks should be able to 'purchase' storage anywhere they like without any prior arrangements.

There are three participants in this system: *Network Users* (NUs), *Network Storage Providers* (NSPs), and *Check Guarantors* (CGs). All participants are identified by their public keys. An NU holds one or more credentials issued by a CG indicating the NU's credit line with the CG, as shown in Figure 5. As with [3], CG credentials issued to users are relatively short-lived, in order that there be no need for credential revocation lists. We should point out however that any revocation mechanism can be used with our system, specified on a per-credential basis. For the remainder of this discussion, and without loss of generality, we will assume that credential revocation is time-based.

There are four kinds of credentials used in different parts of the system:

1. Check Guarantor credentials, which specify a user's line of credit.

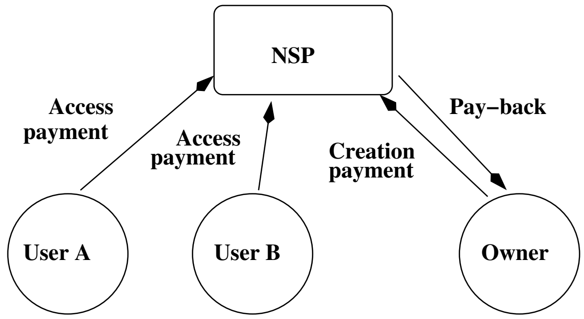

Figure 3: **File owners submit a payment to the NSP at file creation time, and at each access thereafter. Other users also submit payments to the NSP in order to access the file (assuming they have been authorized by the owner). The owner may receive a pay-back from the users through the NSP, if he defines the appropriate file disposition.**

2. Microchecks, which authorize a payment from an NU to an NSP.
3. Server credentials, issued by the CGs, that identify complying NSPs the NUs can use.
4. File access credentials, issued by NUs that own a file, authorizing access to that file.

NUs interact with network storage providers using some file access protocol. The details of the protocol are not important for our architecture; a distributed authoring system such as WebDAV, a file transfer protocol such as HTTP or FTP, or any other data transfer protocol can be used. We assume that the granularity of access is the file; that is, users create, read, delete, append to, or replace whole files. We work with whole files rather than individual blocks for two reasons: to amortize the cost of a check verification over the transfer of an entire file, and to avoid choosing some arbitrary block size and defining block-level operations, which would probably tie FILETELLER to a particular file system philosophy rather than make it a general file-storage service.[1]

NUs may also interact with each other in exchanging files; the role of NSP is an assumed one, and has no physical significance in our architecture. Thus, our architecture can operate in a peer-to-peer environment. Naturally, one would expect that commercial NSPs have better connectivity and storage capacity, and offer better availability guarantees than average users. The data-flows in our system are shown in Figure 4.

After a connection has been established, possibly secured using IPsec or TLS, the client describes the transaction to the server. In particular, the client needs to tell the NSP the name of the file to be accessed, the operation type (create, replace, read, append, delete), the size of the file (for replace, append, or create), the file disposition (ordinary or "pay-back"), the CG credential(s), the ID of the transaction (a random number that the client uses to match requests with responses), and who the owner of the file will be (if someone other than the creator). When the NSP receives this description, it makes sure that the CG is one that it knows about, and sends back an *offer*, consisting of the following fields:

- Transaction ID (echoes what was sent).
- Nonce (for billing purposes).
- Cost, in a real or "play" currency.

[1]That said, a layer can always be added on top of FILETELLER that sends and receives what to FILETELLER look like small, fixed-size files named with sequential numbers, which the layer treats as blocks. We advise against this.

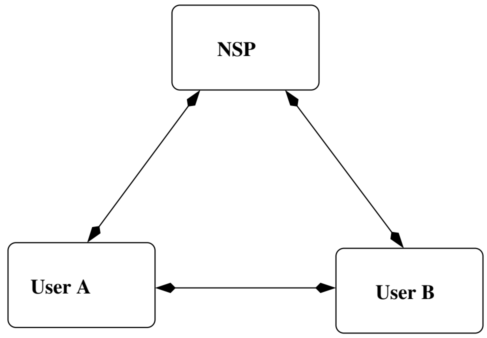

Figure 4: **File data can be stored by "official" NSPs, or by users with spare capacity (*e.g.,* peer-to-peer scenario). Any user in our system can assume the role of an NSP, allowing for a very dynamic storage market, with a low barrier-to-entry.**

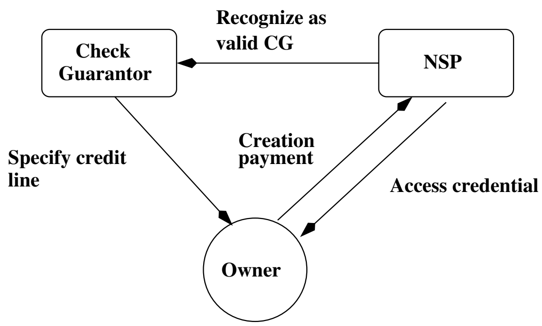

Figure 5: **NSPs issue a credential to each Check Guarantor (CG) authorizing them to act as introducers of users, by issuing them a KeyNote credential. A file owner needs to convince a CG to provide them with a credit line, expressed as a KeyNote credential. The file owner needs to provide these two credentials to the NSP, along with a microcheck conveying payment to the NSP. In response, the NSP returns to the file owner a KeyNote access credential, granting her full privileges in accessing the file.**

The client then writes a microcheck, and sends it along with the credentials necessary to approve the operation: the CG credential(s) and, if this is an access to an existing file, the file ownership chain of credentials, followed by a copy of the offer, the operation, the file disposition, and the attributes, as shown in Figure 6. If the operation was a 'create,' 'replace,' or 'append,' the contents of the file are sent. If the operation was a 'read,' or a 'delete,' nothing else is sent.

When a file is created, the server responds with a file access credential, authorizing the creator of the file to full access (provided the payment has been made). If a file is replaced or appended to, the server responds with a signed 'receipt' for the transaction. Otherwise, the file is sent to the client.

If the client request was for a file with a "pay-back" disposition, two microchecks must be issued; one to the server, and one to the owner. The server will verify that the owner will get paid before releasing the file. The server can either deposit these on the user's behalf, or give them to the owner (in the form of microchecks) for processing. The flow of payments in our architecture is shown in Figures 2 and 3.

There is little difference between processing microchecks drawn in a real currency or in "play money." In the former case, the microchecks must eventually be processed by a financial institution (a bank), as in [3]. In the case of play money, some agent (one of the CGs, or a mutually-agreed-upon party) collects all the microchecks and processes them according to the internal accounting procedures of the group. In the

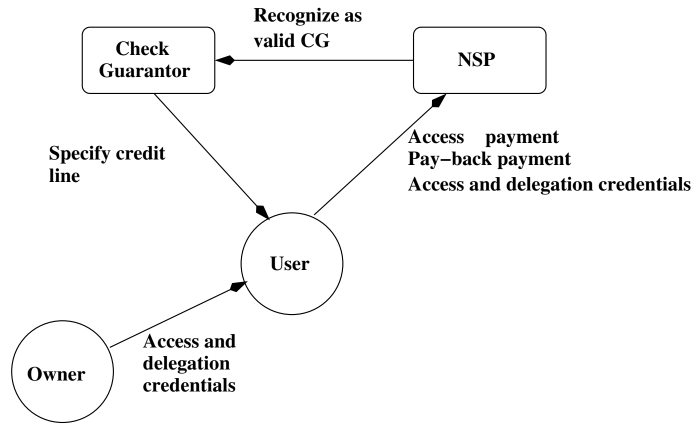

Figure 6: **A user wishing to access another user's file needs to have their own line of credit with a Check Guarantor (CG), as well as a credential from the file owner granting them access to that file. When accessing the file, the user needs to provide the credit-backing credential from the CG, a microcheck to the NSP, and the access credential(s) to the file. If the owner has requested this, an additional microcheck to the owner may also be needed to gain access.**

case of real money, market forces and competition will set the fair price for the service.

File attributes are used in file access delegation credentials to allow subsets of a user's files to be shared. These attributes are meta-data associated with the file by the owner, and can be used to implement easy file grouping, associate security labels with files, or for any other similar scheme.

2.2 Example Usage Scenario

Having seen the overall system architecture, let us look at a particular example. *Alice* is a user who wants to store some of her files in *Nick's* Network Storage Provider servers. Every morning Alice contacts her banker and obtains a fresh *Check Guarantor* credential, which allows her to write KeyNote microchecks. The CG credential (most of the hex digits from the keys have been removed for brevity) allows Alice to write checks for up to 5 US Dollars, and she can do so until March 24th, 2002.

```
Keynote-Version:  2
Local-Constants:  ALICE_KEY = "rsa-base64:MCgCIQGB0f8lSVZfHDwdck\
      ESR/Dh+ONPMrYvd0QlU9QdKbKbRQIDAQAB"
      CG_KEY = "rsa-base64:MIGJAo..."
Authorizer:  CG_KEY
Licensees:  ALICE_KEY
Conditions:  app_domain == "FileTellerPay" && currency == "USD"
      && &amount < 5.01 && date < "20020324" -> "true";
Signature:  "sig-rsa-sha1-base64:QU6SZtG9R3IXXAU9vRDBgu..."
```

Alice now wants to store last night's backup, which took up 250 Mbytes. At the rate of $0.001/Mbyte, she knows that she has to pay 25 cents to store it. She writes a check for this amount:

```
Keynote-Version:  2
Local-Constants:  ALICE_KEY = "rsa-base64:Mcg..."
      NICK_KEY = "rsa-base64:MCgQGB..." Authorizer:  ALICE_KEY
Licensees:  NICK_KEY
Conditions:  app_domain == "FileTellerPay" &&
      currency == "USD" && amount == "0.25" &&
      nonce == "eb2c3dfc860dde9a" && date == "20020123" -> "true";
Signature:  "sig-rsa-sha1-base64:Qpf..."
```

The `nonce` is a random number that must be different for each check, guaranteeing that there will be no double-depositing of checks. She also constructs a request to Nick telling him that she is about to send over a 250 Mbyte file, gives the file name, and sends it along with the microcheck and the Check Guarantor credential. Note that Alice uses the same key to identify Nick in his function as a recipient of money, and in his function as the storage server. Actually, a simple string such as "NICK'S DISK FARMS" would have sufficed for the purposes of writing the microcheck. However, it makes bookkeeping easier to know each principal by their public key.

Nick receives these credentials, validates the microcheck to make sure that he will get paid, and waits for the file to arrive. If the check is not good, Nick will say so, and refuse to accept the file. Of course, Nick may keep the check and refuse the file anyway, which just means that Alice will probably accept the loss of 25 cents, and never do business with Nick again. If Nick is honest, he will respond by sending back a *file access credential:*

```
Keynote-Version:  2
Local-Constants:  NICK_KEY = "rsa-base64:MCgQGB..."
      ALICE_KEY = "rsa-base64:MIGJAo..."
Authorizer:  NICK_KEY
Licensees:  ALICE_KEY
Conditions:  app_domain == "FileTellerAccess"
      && filename == "/home/alice/backups/fulbak-20020122"
      && ( (access == "R") || (access == "RP") || access == "A" ||
             (access == "D" ) ) -> "true";
Signature:  "sig-rsa-sha1-base64:QU634865a88..."
```

Alice can use this credential to access the file, by sending it along with proper payment. The credential grants her Read, Replace, Append, and Delete access; that is, all the operations she may want to perform. If Alice wants to read the file, she will have to send both a microcheck (along with the CG credential) and the file access credential. Nick will verify that he will get paid, and then will evaluate the access credential on the file access credential he got from Alice, in conjunction with his own policy:

```
Keynote-Version:  2
Local-Constants:  NICK_KEY = "rsa-base64:MCgQGB..."
Authorizer:  POLICY
Licensees:  NICK_KEY
Conditions:  app_domain == "FileTellerAccess" -> "true";
```

This policy says that anything that Nick's key authorizes is allowed. Since the file was paid for, and a path can be found from POLICY to ALICE_KEY (the requestor of the operation), the operation is allowed.

As a matter of business practice, Bob may require periodic payments from Alice in order to keep her files around. Alice must know that and send microchecks at the appropriate intervals.

Alice may also grant access to Bob to just read one of her files (if, *e.g.,* the file were a picture and not a backup). She can *delegate* to Bob by issuing the following credential:

```
Keynote-Version:  2
Local-Constants:  ALICE_KEY = "rsa-base64:MIGJAo..."
      BOB_KEY = "rsa-base64:MCgQGXW..."
Authorizer:  ALICE_KEY
Licensees:  BOB_KEY
Conditions:  app_domain == "FileTellerAccess"
      && filename == "/home/alice/pics/DSCN1033.JPG"
      && access == "R" -> "true";
Signature:  "sig-rsa-sha1-base64:QU63486532a..."
```

Bob will have to first pay for the access, and send both Alice's original file access credential (that Nick issued) and the one that Alice issued him. In the standard KeyNote way, since a path will be found from Nick's POLICY to BOB_KEY (the requestor), the file will be given to Bob.

In another mode of operation, Alice may want to get paid every time Bob (or anyone else) accesses her files. She indicates this by giving Bob a slightly different credential, and telling him that he must pay 50 cents to access the file:

```
Keynote-Version:  2
Local-Constants:  ALICE_KEY = "rsa-base64:MIGJAo..."
      BOB_KEY = "rsa-base64:MCgQGXW..."
Authorizer:  ALICE_KEY
Licensees:  BOB_KEY
Conditions:  app_domain == "FileTellerAccess"
      && filename == "/home/alice/pics/DSCN1033.JPG"
      && access == "R" && value >= "0.50" && currency == "USD" -> "true";
Signature:  "sig-rsa-sha1-base64:QU63486532a..."
```

Bob must then issue his request to Nick by also including a microcheck payable to Alice. When Nick gets that microcheck, he includes in the KeyNote *action environment* the price paid, which will enable that last credential to evaluate to `true`. Note that by simply setting the value paid to 0 when he does not receive payment for Alice, Nick can use the exact same mechanism for both kinds of access, and at the same time Alice can have some people get the file for free (by issuing a credential without the payment requirement) or for pay (by issuing a credential with the payment requirement). In either case, Nick will include the amount paid; if Bob does not send payment for Alice but has the latter kind of credential, he will not get access because the `value >= "0.50"` condition will not be satisfied. As part of the service to Alice, Nick collects all the microchecks issued for her and gives them to her at a later time.

Alice does not have to issue individual credentials to each potential customer. She can simply publish the following credential on her Web page:

```
Keynote-Version:  2
Local-Constants:  ALICE_KEY = "rsa-base64:MIGJAo..."
Authorizer:  ALICE_KEY
Conditions:  app_domain == "FileTellerAccess"
      && filename == "/home/alice/pics/DSCN1033.JPG"
      && access == "R" && value >= "0.50" && currency == "USD" -> "true";
Signature:  "sig-rsa-sha1-base64:QU63486532a..."
```

In this case, since no Licensees are listed, anyone can use this credential to access the file, provided of course that they pay for it.

Alternately, Alice may wish to pre-pay some fixed amount to a CG on a monthly basis and get some number of transactions for free. In that case, the CG can issue a credential of this form:

```
Keynote-Version:  2
Local-Constants:  ALICE_KEY = "rsa-base64:MCgCIQGB0f8lSVZfHDwdck\
      ESR/Dh+ONPMrYvd0QlU9QdKbKbRQIDAQAB"
      CG_KEY = "rsa-base64:MIGJAo..."
Authorizer:  CG_KEY
Licensees:  ALICE_KEY
Conditions:  app_domain == "FileTellerPay" && currency == "USD"
      && (&amount < 5.01 || amount == "prepay") && date < "20020424" -> "true";
Signature:  "sig-rsa-sha1-base64:QU6SZtG9R3IXXAU9vRDBgu..."
```

In this case, Alice will tell the NSP that she is using a pre-paid plan when accessing her files, and issue a microcheck authorizing charging of the pre-paid plan. While that microcheck does not directly convey any value transfer, it acts as a proof of the transaction that the NSP can later use to collect payment from the CG, or that a pre-set limit was exceeded.

Finally, Alice may wish to be charged for all accesses to her files, instead of having the users be charged; this may be the case for a company that uses exclusively off-site storage for its operations. There are two ways Alice can achieve this:

- By using a pre-paid plan, as described above.
- By becoming a Check Guarantor and establishing agreements with one or more NSPs. In this case, Alice issues her own CG credentials to her users. If desired, she can require that any files created using these credentials be assigned to her: in that case, the access credential returned by the NSP will specify Alice as the file owner. She can then issue an access credential to the user who created the file, granting them the relevant access. If the user leaves the company, Alice remains the owner of the file, and can authorize someone else to access it.

2.3 Security Analysis

Similar to [3], our system has four types of communication: provisioning, reconciliation, transaction, and delegation. We shall not worry about any value transfers to banks, as there already exist well-established systems for handling those. All communications between CGs, NSPs, and NUs can be protected with existing protocols such as IPsec or TLS.

The security of the stored data itself is not within the purview of our system, nor is it a responsibility of the NSP; if the user does not trust the NSP not to look at their data, they need to take appropriate measures, such as encrypting their files before storing them at the NSP. Existing mechanisms, such as CFS, SFS, or manually-encrypted files can be used with our architecture to address such concerns.

User provisioning must also be protected; although the credentials themselves are not confidential and are integrity-protected (by virtue of being digitally signed), the mechanism by which the user pays the CG (or somehow establishes trust) must be secured. Such mechanisms however already exist, *e.g.,* online payments using a credit card, and can be used for provisioning.

The user needs to authenticate with the NSP before any file operation can take place. The details of the authentication protocol used are not important, and existing security protocols can be used for that purpose. The requirements for the authentication protocol are that it provides strong authentication and, optionally, lets the user piggy-back credential delivery to the NSP; conveniently, most security protocols in use today (IPsec, TLS) meet these requirements.

At a higher level of abstraction, the NU has to worry about the NSP providing service; this is something that can be easily verified by the NU. Furthermore, the NU can use multiple NSPs to achieve better reliability through replication. Protecting against over-charging NSPs or CGs is also straightforward: the details of each transaction can be verified at any point in time, by verifying the credentials and the offer. Since only the NU can create microchecks, a dispute claim can be resolved by running the transaction again. The NU is safe even from a collusion between CGs and NSPs.

Since the value of the individual transactions is small, losses can be tolerated on either side; in particular, if the NU requests access to a file for which she does not have permission, the NSP can keep the payment[2]. Charging for access to non-existant files depends on the NSP's policy.

The NSP must also faithfully enforce the access control policy of the file owner. This is of particular interest in the case of the "pay-back" scheme. There is no way of guaranteeing this to the NU, except by introducing an encryption layer; in that case, however, the owner will have to be involved in every transaction involving the file. This also raises digital rights management (DRM) issues, which are outside the scope of our architecture. We believe that such issues are better addressed via real-world means (*e.g.,* contracts, legal action, *etc.*).

The NSP must make sure they are paid for the services offered. Since it has a copy of all transactions (the CG credential, the microcheck, and the offer), it can prove to the CG, or any other party, that a transaction was in fact performed.

The CG also needs to be paid for the services offered. Since the CG does the clearing of the microchecks, the NSP has to provide the transaction logs to the CG. Using those, the CG can verify that a transaction was done, and at what value. A collusion between the NSP and an NU is somewhat self-contradicting: the NU's goal is to minimize cost, while the NSP's is to maximize revenue, each at the expense of the other. The function of the CG is to verify each transaction (perhaps sampling, for very large numbers of transactions), debit the NSP and credit the NU (presumably keeping some commission or small fee in the process): if the NSP does not give any credentials to the CG, then no work was done as far as the CG is concerned (and no payments are made, which benefits the NU); claiming more transactions than really happened is not in the best interest of the NU (so no collaboration could be expected in the direction), and the NSP cannot "fabricate" transactions.

Since value is not stored in either the NSP or the NU, only a reliable log of the transactions is needed at the NSP (and, optionally, at the NU).

3 Implementation

We have constructed one demonstration system and are working on a second; each implements credential-based access to on-line information using a different protocol. The existing system uses the ftp protocol, while the other will use a Java applet to implement the client, thus providing Web-based access to FILETELLER.

Our prototype implements the functions we described earlier in the Section 2, namely the backup and retrieval of files stored at the remote server, the creation of new files, and the delegation of access rights to other users.

When a file is first introduced to the remote server, a credential allowing its future retrieval is stored on the local machine. To avoid these credential files from cluttering up the user's directories, we place them in a special subdirectory (called FT) in the directory where our file is stored.

[2]To our knowledge, this is the first system where illegal file access results in a "fine" for the misbehaving user.

Other information such as the user's public and private keys and the keys of remote servers and check guarantors are placed in the *.fileteller* subdirectory in the user's home directory.

To retrieve a file, the user must specify the name of the file (on the local machine) and the directory it was stored in (so that the file "FT/filename,cred" may be located). Other information, such as the name of the server and the name of the file on the server are stored in the credential file).

The most complex operation is that of delegation. Using the DELEGATE command, the user can grant access rights to another user. The delegate command must include the file where the public key of the other user is stored, the permissions (read, write, append, *etc.*) that the other user will be granted, the file name and its directory (on the local machine) and, optionally, the price of the file. This price is the pay-back fee that the local user will receive each time the file is accessed by the remote user, and is in addition to the access fee charged by the storage provider.

This prototype is highly portable and runs on all the platforms that support the KeyNote library. Nevertheless, in order to make the FILETELLER system truly platform-independent, we are working on a Java implementation that will run on all popular Web browsers. The capabilities of the FILETELLER-Java prototype will be similar to those of the existing prototype and a lot of code will be simply ported to Java.

4 Related Work

A number of protocols have been deployed for users to share files across the network, the most commonly used being FTP and HTTP [7, 4]. Anonymous FTP, where there is no need for authentication, offers great flexibility since any user can download or upload files to FTP servers. Similarly in the Web architecture, access is either anonymous or subject to some sort of ad-hoc authentication mechanism. This configuration is useful only in the case where file content is non-critical. Where authentication is required, flexibility is greatly reduced since the only users allowed to access the server, in that case, are users who are already known to the system. Furthermore, these schemes lack the ability to charge for file usage which is what FILETELLER is designed to do.

Network file systems (*e.g.,* NFS and AFS [8, 5]) are also popular and widespread mechanisms for sharing files within the same administration domain. Crossing administrative boundaries creates numerous administrative problems (*e.g.,* merging distinct Kerberos realms or NIS domains).

Encrypting file systems like CFS [1] place great emphasis on maintaining the privacy of the user information by encrypting the file names and their contents. The limitation of such systems is that sharing is particularly difficult to implement; the file owner must somehow communicate the secret encryption key for the file to all the users that wish to access it. Even then, traditional access controls must still be used to enforce access restrictions (*e.g.,* read-only, append-only, immutable file, *etc.*).

WebFS is part of the larger WebOS [9] project at UC Berkeley. It implements a network file system on top of the HTTP protocol. WebFS relies on user-level HTTP servers used to transfer data, along with a kernel module that implements the file system. Access control lists (ACLs) are associated with each file that enumerate users who have read, write, or execute permission on individual files. Users are uniquely identified by their public keys. We have taken a more general and scalable approach in that there is no need for ACLs since each credential is sufficient to identify both the users and their privileges.

The secure file system (SFS [6]) introduces the notion of *self-certifying pathnames*: file names that effectively contain the appropriate remote server's public key. In this way, SFS needs no separate key management machinery to communicate securely with file servers. However, since SFS relies on a trusted third party to mutually authenticate server and client, collaboration is possible only if the client and the server have a common root for their Certification Authorities.

FILETELLER differs from the file system solutions proposed above in two very significant ways. It uses credentials to identify both the files stored in the file system and the users that are permitted to access them, as well as the circumstances under which such access is allowed. Also, users can delegate access rights simply by issuing new credentials, providing a natural and very scalable way of sharing information. Furthermore, it departs from all previous approaches by incorporating a micropayment scheme that allows users to buy file system space on remote servers and subsequently sell access to their files to anyone on the network.

Finally, WebDAV [10] is a system that addresses the issue of collaborative authoring. WebDAV defines the HTTP extensions necessary to enable distributed web authoring tools to be broadly interoperable. It leverages the success of HTTP in being a standard access layer for a wide range of storage repositories – standard HTTP is used for read access, while DAV provides write access. FILETELLER is not limited to a specific access layer and is able to work on any file transport substrate, providing support for flexible file charging and access control at the same time.

5 Summary and Concluding Remarks

We have presented FILETELLER, a system for flexible global file sharing, made possible by using a credential-based network file storage system with provisions for paying for file storage and getting paid when others access files. Both the file access control and payment are implemented using the KeyNote trust-management system,

Using a trust-management system as the access-control mechanism relieves the servers from having to keep access control lists or other authorization data, since all the necessary information is kept in a distributed fashion by each individual user in the form of KeyNote credentials. The use of a micropayments system allows each request to be paid-for as it occurs, and the only bookkeeping that has to happen is the clearance of the microchecks; this further reduces overheads on the server.

Our plans for future research include the integration of FILETELLER with the WebDAV protocols, with the ultimate objective to create an academic research portal site acting as a vehicle for collaborative authoring of research papers and repository of published work.

References

[1] M. Blaze. A Cryptographic File System for Unix. In *Proc. of the 1st ACM Conference on Computer and Communications Security*, November 1993.

[2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote Trust Management System Version 2. Internet RFC 2704, September 1999.

[3] M. Blaze, J. Ioannidis, and A. D. Keromytis. Offline Micropayments without Trusted Hardware. In *Proceedings of the Fifth International Conference on Financial Cryptography*, 2001.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, 1997.

[5] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[6] David Mazieres, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. In *Symposium on Operating Systems Principles*, pages 124–139, 1999.

[7] J.B. Postel and J. Reynolds. File Transfer Protocol, 1985.

[8] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and implementation of the sun network file system. In *Proceedings of the 1985 Summer Usenix Conference*, Portland, OR, June 1985.

[9] Amin Vahdat. *Operating System Services for Wide-Area Applications*. PhD thesis, University of California, Berkeley, December 1998.

[10] E. Whitehead. World wide web distributed authoring and versioning, 1997.